NexusFile: A Protocol for Economically Sustainable Information Permanence

Sam Williams
sam@nexusfile.co

Viktor Diordiiev
viktor@nexusfile.co

Lev Berman
lev@nexusfile.co

India Raybould
india@nexusfile.co

Ivan Uemlianin
ivan@nexusfile.co

DRAFT-1
C0bQOaBVoznjkrabyM4TPs5Rf5-ZthrY8MYDr-FU6DRle4yBhaMlNphglTrb06Hp

Abstract

Blockchains have been used as mechanisms to store memoised rep-resentations of history since the very rst Bitcoin block [47]. Despite blockchain technology's clear potential in the area of resilient archive construction without single points of failure, advances in on-chain data storage techniques have remained elusive. This paper addresses this problem through the introduction of the NexusFile protocol: a new mechanism design-based approach to achieving a sustainable and permanent ledger of knowledge and history. As well as outlining incentive mechanisms for achieving sustainable data permanence, this paper outlines key technologies to allow scalable on-chain storage.

Contents

1 Introduction

1.1 Motivation

In this work we present the NexusFile protocol, a new blockchain-like data structure called the blockweave. The protocol is designed to provide scalable and permanent on-chain data storage in a sustainable manner. The block-weave forms the underlying data structure of the permaweb - the array of data, websites, and decentralised applications hosted on the blockweave, accessible on normal web browsers.

In this paper, we will introduce several technological innovations that, together, allow NexusFile to o er unique utility in the on-chain data storage space, including: blockweave, blockshadows, AIIA, decentralised content policies, and their mechanism design.

Our collective ability to store and share information between individuals and across time to new generations has been essential to humanity's successes. Despite our best e orts, however, throughout history, our methods of storing knowledge have been vulnerable to destruction and the information subject to loss or alteration { sometimes with intentional malice, and more often accidentally. Just as in the ancient world, modern history is full of examples of the destruction, alteration, and loss of vital information, from res at libraries and archives [41, 22, 35], to book burning in authoritarian states [57].

Today, with a wealth of digital information surrounding us, we can easily begin to assume that because information is readily available online today, it can't be altered or lost. Unfortunately, this is foundationally untrue [21, 37]. Although the internet is a wildly successful system of distributed information dissemination, it currently lacks a complementary system of decentralised, permanent knowledge storage.

Almost all of the pages making up the web today are housed within centralised data stores, each typically controlled by one organisation or even one individual. This means that when accessing information online, we are wholly reliant on these centralised organisations and individuals continu-ing to allow us to do so. Access can be revoked at any time, or the data can simply be unintentionally lost or degraded. Serving information on the internet incurs server and other upkeep costs, meaning whole websites, ap-plications, and information stores can simply disappear when funds are no longer available for maintenance. Another signi cant risk of this centralised data storage model is that the data is vulnerable to manipulation by these individuals or organisations. Such manipulation tactics typically include the modi cation of documents during their storage [34, 48]. NexusFile o ers a solution to this problem.

Further still, a number of governments are increasing their e orts to censor and remove access to politically sensitive information on the internet [68, 6, 1]. Similarly, with media and news organisations, while we once held physical and irrevocable copies of their publications, we now simply access the information digitally and then immediately discard it. It has become commonplace for media organisations to update the contents of their articles over time. This can cause vital context and content to be lost or obscured.

In order for an information store to be truly permanent, it must be both re-silient and decentralised. Blockchain technology has much obvious promise in the area of resilient, decentralised information preservation [14], as a key feature of the technology is that all data inside the blockchain is immutable, and cannot be altered once it is stored. However, traditionally, such tech-nology severely lacks scalability which clearly limits its utility for storing signi cant quantities of data.

High transaction fees also inhibit typical blockchains' ability to scale to accommodate large amounts of data. For instance, with the Ethereum net-work, although it is technically possible to store data on-chain, the high fees make this impractical for most real-world use cases.

As the demand for data storage grows exponentially [56], the need for a decentralised, low-cost data storage protocol that can scale is a necessity.

Once a piece of data is stored in the data structure, it is cryptographically entangled with every other previous block in the network. This ensures that any attempt to change the contents of the document will be automatically detected and consequently rejected by the network. Therefore, the NexusFile o ers a robust way of permanently storing data on-chain, beyond the reach of accidental or intentional data loss or manipulation.

Owing to the decentralised and cryptographically veri ed nature of the per-maweb, it is beyond the reach of any organisations or groups that might conspire to censor its contents [36]. As such, the NexusFile drastically re-duces the possibility of an Orwellian 'memory hole' [51] from occurring. In this way, we envisage the NexusFile protocol as a useful tool in the preserva-tion of the freedom of information, and subsequently in the strengthening of institutions and democratic processes that depend upon it.

However, in order for the NexusFile protocol to ful l its potential and enable the permaweb to become the basis of the new, decentralised web, widespread developer adoption of the protocol is essential. As such, the protocol is designed to make it simple, and both cost and time e cient for developers to store content on the network. NexusFiles HTTP API makes it extremely simple to build decentralised web applications on top of the blockweave. In order to make permaweb development as simple as possible, developers can use all of their favourite familiar web technologies (including HTML, Javascript, and CSS) and deploy to the permaweb in minutes.

For the end-users themselves, the permaweb built on the NexusFile protocol o ers several major advantages. Firstly, as we have touched on previously in this paper, users are guaranteed reliable, immutable access to permaweb content. Of especially great importance is users ability to reliably maintain access to all permaweb applications and websites themselves, not simply the content they display, forever. This means that, once published, a permaweb app cannot be unpublished, helping to maintain strict application integrity
[5]. This allows users to exercise meaningful choices based on their personal preferences.

The speed of access to permaweb content is also incredibly important for the user experience, and additionally for adoption of the NexusFile protocol and permaweb itself. The NexusFile protocol is the only system to o er truly permanent and decentralised data storage. The protocol is designed to robustly incentivise NexusFile nodes to share data around the network quickly, a key aspect of the protocols mechanism design explained in section 6 describing the AIIA meta-game, and its implementation in the wild re mechanic.

The NexusFile protocol and its family of technologies have all been engineered to ensure that each node's behaviour is likely to contribute positively to the utility of the network itself. In this way, the protocol design is Dominant Strategy Incentive Compatible (DSIC, [58]), a vital facet of a healthy, long-term decentralised and incentive-driven network such as NexusFile.

As such, the NexusFile protocol is designed to address many of the short-comings in both traditional archiving systems and also the scalability issues found with many typical blockchain protocols. The protocol is designed to maximise the quality and e ciency of both developer and end-user experi-ences, which is vital when aiming for wide-scale adoption.

1.2 Structure of This Document

1. Introduction
2. NexusFile: Key Contributions

This document is split into eight sections, plus an appendix section. Here in section 1, The Introduction, we establish the main motivations and goals of creating the NexusFile protocol: providing permanent, resilient storage.

Section 2 highlights the novel key contributions the NexusFile protocol o ers to the decentralised data storage space. The speci c innovations described include: the blockweave, the Proof of Access consensus mechanism, the recall block, memoisation of state, blockshadows, the Adaptive Interacting Incentive Agents (AIIA) meta-game, and wild re, an implementation of an AIIA game. This section explains how the blockweave di ers from a traditional blockchain data structure, as well as the implications and bene ts of these di erences.

Section 3, The Network: Mechanism Design, establishes that distributed, decentralised storage is necessary to meet the requirements of permanence and resilience. Taking a game-theoretic and mechanism design approach, this section describes how the NexusFile protocol is carefully engineered to balance incentives and constraints to produce a network that is Domin-ant Strategy Incentive Compatible (DSIC, [58]), ultimately maximising pro-social utility of the network's overall output.

Section 4, The Node: Behaviours for Protocol Compliance, looks at the network and the protocols from the perspective of how nodes behave within these systems. This section covers in detail the main activities of a node: mining blocks, receiving and validating blocks and transactions, and serving blocks and transactions. To illustrate how the NexusFile protocol promotes pro-social behaviour, this section includes some examples of what happens when a node tries to \cheat".

Section 5, Democratic Content Policies, describes the mechanism by which NexusFile miners - the data storage providers in the network - can accept, store, and share only the content they choose to. This section describes the three primary aspects of content policies: the voting phase, the storage phase, and the incentive (or, mechanism) design. Here, you will discover how the blockweave enables miners to democratically decide which storage content they collectively do or do not wish to store inside the network.

Section 6, covering the phenomenon of Adaptive Interacting Incentive Agents (AIIA), describes how the mechanism design and the technical implementa-tion of the NexusFile protocol combine to create a powerful web of mutually bene cial and counter-balancing incentives. The powerful emergent e ects of this interaction are dissected, demonstrating how the protocol produces high levels of positive externalities { including pro-sociality between nodes and users alike { without forcing compliance on agents in the system.

Section 7, NexusFile Protocol Interoperability and the Permaweb, describes all of the key technical and infrastructure components that together make up the NexusFile protocol, and consequently, the permaweb itself. This section describes how fundamentally, the protocol operates in a trustless, serverless, and distributed manner. Additionally, the speci c permaweb developer- and miner-oriented technologies of the protocol are enumerated, including: the HTTP API, the client-server, gateway nodes, hybrid architectures, ArQL, NexusFile DNS & TLS, and a number of live permaweb applications.

Section 8, Future Work, explores the core NexusFile development team's views on what additional technologies may be added to the NexusFile protocol to further enhance its unique o ering. Firstly, the section describes succinct proofs of access, whereby data storage remains resilient, veri able, and se-cure, but the amount of data contained in transactions passed around the network is reduced, increasing e ciency. Secondly, a proposed reduction in the storage space dedicated to maintaining full wallet logs is considered. Finally, a proposal for a potential fast nd mechanism, is discussed in de-tail, which would ultimately accelerate how quickly data is located in the network when requested.

Section 10, The Appendices, enumerates the speci c contents of a range of novel NexusFile data structures, including block, blockshadow, and transac-tion data structures, as well as providing key datasets.

2 NexusFile: Key Contributions

This section looks at unique innovations engineered for inclusion in the Ar-weave protocol, how these impact usability, resilience, and permanence of data storage on the NexusFile network.

2.1 The Recall Block, Proof of Access, and the Blockweave

In NexusFile's blockweave data structure each block is linked to two prior blocks: the previous block in the 'chain' (as with traditional blockchain protocols), and a block from the previous history of the blockchain (the 'recall block'). Consequently, the NexusFile blockchain data structure is not strictly a chain (i.e., a singly linked list) but it is a slightly more complex graph structure that we call the blockweave.

The recall block is selected based on a hash of the previous block and the previous block's height. This results in a deterministic but unpredictable choice of block from the weave's history.

In order to mine or verify a new block, a node must have that block's recall block. Demonstrating proof that the miner has access to the recall block is part of block construction (and conversely, verifying this proof is part of validating a new block). Proof of Access (PoA) is an enhancement of Proof of Work (PoW) in which the entire recall block data is included in the material to be hashed for input to the proof of work. For further details of the block construction process, see section 4.1.1.

Requiring PoA incentivises storage as miners need access to random blocks from the blockweave's history in order to mine new blocks and receive mining rewards.

The PoA algorithm also incentives miners to store 'rare' blocks more than it incentivizes them to store well-replicated blocks. This is because when a rare block is chosen, miners with access to it compete amongst a smaller number of miners in the PoW puzzle for the same level of reward. As a consequence of this, miners that prefer to store rarer blocks on average receive a greater reward over time, all else being equal.

PoA takes a probabilistic and incentive-driven approach to maximising the number of redundant copies of any individual piece of data in the network. By contrast, other decentralised storage networks specify an exact number of redundant copies that should be provided for a given piece of data, and mediate this using a system of 'contracts'[12]. The NexusFiles competition-based approach is tailored such that it is simpler and more versatile, performing exibly in both well-functioning and poorly-functioning environments.

For example, in a network storing 1 PB of data where only 2 PB of data storage capacity is available (an ‘unhealthy’ network) the PoA incentive structure pushes miners to make sure they have copies of data that few other miners are storing, reacting exibly to the situation. In a well-functioning network where, for example, only 1PB of an available 100 PB storage capacity is in use, the NexusFile also pushes miners to make larger quantities of redundant copies of data in the network. As well as increasing security and stability in such a situation, this approach to redundancy maximisation through incentives also leads to dramatically faster lookup and access times.

2.2 Memoisation of State

In the standard blockchain paradigm (e.g. in the Bitcoin blockchain[47]), the blockchain is a concrete object replicated entirely on every full node in the network. Each full node must store the blockchain in its entirety. As explained in the previous subsection, with NexusFile this is not necessary. The NexusFile’s novel data structure, the blockweave, does not require miners to store every previous block. In order to achieve this, all data required to process new blocks and new transactions is memoised into the state of each individual block (see section 10.2.1 for details of the block data structure).

As a consequence of this memoisation technique, new users are able to join the network by downloading only the current block from its trusted peers, or verifying some backward portion of proofs of work (the larger the quantity veri ed the lower the trust required to join the network). This block data structure includes, among other things, the Block Hash List (BHL) and the Wallet List (WL). Possession of the Block Hash List allows old blocks to be requested and/or veri ed. Possession of the Wallet List allows new transactions to be veri ed without possessing the block in which the wallet’s last transaction was included. The Block Hash List and the Wallet List are kept up to date by miners and synchronised by the network when mining and validating new blocks (see section 4 for details of node behaviours during NexusFile mining).

This reduces barriers to entry for miners { barriers of storage space, processing power, and time { allowing the blockweave to scale to sizes larger than the capacity of any individual miner. As the NexusFile network in-tends to scale past the size, scope, and volume of the traditional web, this mechanism is vital to allowing miners to practically join the network.

Parties interested in verifying the full blockweave from the rst block to the current block, and in reconstructing the entire blockweave locally can do so in a number of ways. For example, by following the link in each block to that block’s previous block, or, by requesting a Block Hash List from a trusted peer, verifying it against the unbalanced Merkle tree hash list in the current block, and downloading the blocks directly.

Once joined and active, there is no need for a node to store the entire blockweave at all, however, they are rewarded by the Proof of Access mech-anism relative to the proportion of the blockweave they store. The Domin-ant Strategy Incentive Compatible (DSIC[58]) nature of the network gives network-level guarantees of storage and replication. This frees each indi-vidual node to prioritise and optimise their own storage according to their own preference and resources.

A blockchain is typically a very large distributed data structure, and down-loading the full blockchain can take a long time and consume a great deal of computing resources[13]. Unlike traditional blockchain systems, NexusFile does not have a typical notion of full and light clients { merely clients that downloaded more or less of the blockweave. With NexusFile, full synchron-isation is not a risk or an obligation, but an optional upgrade path for which miners receive higher rewards.

2.3 Blockshadows

In a traditional blockchain network, when a new block is mined, each entire block is distributed to every node in the network, no matter how many of the block's transactions that node already possesses. This signi cantly limits the amount of data that can be included in a block, as all of the data needs to be gossiped around the network during consensus. If too much data is transferred during block acceptance, the time required to achieve consensus becomes too large and forks emerge in the network. The probability of a fork emerging in a blockchain network during consensus is as follows:

$$P\ (\text{fork}) = \frac{B_{\text{dist\_time}}}{B_{\text{time}}} \qquad (1)$$

Subsequently, as the block distribution time is linear with the block's size, so too is the size linear with the likelihood of forks emerging. As the NexusFile protocol stores data inside the blocks themselves, the normal blockchain trade-o between P (fork) and $B_{\text{max size}}$ is not acceptable.

Therefore, the NexusFile protocol takes a new approach to data distribution: blockshadows, building on the work of Graphene[52] and compact blocks (BIP-152)[19].

Blockshadowing works by decoupling transactions from blocks, and only sending a minimal ’blockshadow’ between nodes which allows peers to re-construct a full block, rather than transmitting the full block itself. These blockshadows contain a hash of the Wallet List and Block Hash List, and list of transaction hashes (instead of the transactions inside a block). From this information (typically a few kilobytes), a node that already holds all of the transactions inside the block, plus an up-to-date Block Hash List and Wallet List can rapidly reconstruct an entire block of almost arbitrary size. Using this mechanism, the bottleneck for block size becomes the number of transaction IDs that can be included, and the length of time required to reconstruct a block from its constituent transactions.

In order to facilitate block distribution and lower communication overhead, nodes immediately share transactions with one another, but only attempt to place transactions inside a block once they have a high degree of certainty that other nodes in the network also have a copy of the transaction. Further improving on the work presented in Graphene[52] and BIP-152[19], block-shadows introduce a mechanism design-based AIIA game for calculating the likelihood that other nodes in the network already have local access to a block. Because nodes do not attempt to fetch missing transactions from one another if they do not have them locally, nodes are incentivised to act in the following way:

1. Not to mine transactions into blocks too early, as this leads to their blocks being rejected.
2. Not to mine transactions into blocks too late, as other miners in the network are likely to mine them beforehand.

The result of this blockshadowing system is a fast and exible block distribution process that allows transactions to be mined into a block as fast as they can be distributed around the network, and consensus about blocks to be achieved at near network speed.

2.4 Content Policies and Censorship Resistance

The NexusFile protocol avoids making it an obligation to store everything, which in turn allows each node to decide for itself which blocks and transactions to store. This increases the censorship resistance of the network[3] as nodes are not forced to store material they don’t want to.

The default behaviour of the network is a large number of replications of each single accepted transaction. The current replication rate in the network exceeds 97%. As a necessary part of block validation, transactions are stored on-chain and replicated throughout the network, so the data is distributed widely in geographic terms. Storage is guaranteed probabilistically and at the network level (rather than on a per-miner basis), consequently strength-ening these guarantees, making them robust, and resilient to interference. These advantages are bene cial to both end-users, for whom data access is faster and more reliable than other alternatives, and also for miners, who are able to select which blocks and transactions they wish to store, allowing them to implement content policies that meet their preferences (see section 5 for more information on such policies).

2.5 Wild re

In the wild re mechanic, a form of AIIA game (see section 6), each node in the NexusFile network ranks its peers based on two primary factors. Firstly, the peer's generosity - sending new transactions and blocks, secondly, the peer's responsiveness - responding promptly to requests for information, in a similar mechanism to Bittorrent's optimistic tit-for-tat algorithm[18]. The node then gossips preferentially to higher-ranked peers. This allows a node to rationalise its bandwidth allocation. It also has the e ect of promoting pro-social behaviour on the part of nodes generally, given the practical implications of how every peer interacts with every other peer.

See section 3.4 for further information about the wild re mechanic. See also section 6 on the Adaptive Interacting Incentive Agents (AIIA) meta-game, of which wild re is an example agent.

3 The Network: Mechanism Design

The network is organised into two categories of actors: miners and users. Users pay tokens (AR) to add data to the network. Miners in the network receive these tokens for mining new blocks, which requires them to store and serve data. It is possible for a single wallet owner to be both a user and a miner. The miners receive these rewards indirectly from users in the system that pay tokens to add data to the network. The assignment of tokens from users to miners is mediated by the network as a whole through mining and validation of new blocks.

Data intended for addition to the network is encapsulated in transactions, which are mined into blocks. A block is both a container for a set of trans-actions, and a memoised representation of the state of the network after the inclusion of the accepted transactions. From the user’s perspective, there are two types of transactions in the network: data transactions and value transactions. A user can initiate a data transaction to store data in a block. Value transactions contain only the change of AR balances in two wallets: a decrease in the wallet that initiated the transaction, and an increase in the wallet that received the transaction. In terms of technical implementation, all transactions in the NexusFile network are the same. Every transaction can have an optional recipient eld, and an optional data eld. This means that transactions can be highly exible, for example, they can be used to send mail between two participants (see section 7.5 for a range of example permaweb applications, including Weavemail). See section 10.2 for full de-tails of block and transaction contents, as well as section 4 for details of how nodes manipulate these data structures.

3.1 Di culty: Regulating the Block Generation Rate

In order to regulate the block generation speed in the network, the Proof of Access (PoA) algorithm allows for variable di culty settings. PoA challenges with higher di culty take longer to compute. ‘Di culty’ is a network statistic, included in each block[47].

If the generation rate of blocks in the network exceeds the target frequency, the di culty of the PoA puzzles for future block generation is increased. Similarly, as the block generation speed in the network decreases, the di - culty setting is adjusted downwards.

In this way, the decentralised network of miners is able to regulate the block generation rate (and subsequently, various reward emission rates { detailed below) regardless of the number of nodes in the network and the amount of computational power and storage available to solve the PoA puzzles.

3.2 Token Economy

3.2.1 Paying the Network, Rewarding the Miners

The NexusFile network uses a token, the scarcity of which is enforced through the consensus mechanism of the blockweave data structure. The token’s main unit is the AR, with sub-unit Winston, where 1 AR = 1,000,000,000,000 Winstons.

As the token in the system is scarce and is used for two valuable functions, that of encoding data into the system and of rewarding miners, the token itself has a non-zero nancial value. Although the token primarily derives its utility from being the only instrument of paying for permanent data storage, it can also be used as a means of value exchange.

55 million AR were created in the genesis block at network launch on the 8th June 2018. A further 11 million AR, an additional 20% of the genesis block supply, are being introduced into circulation gradually as block mining rewards. Consequently, the maximum circulation will be 66 million AR. AR tokens in circulation are held either in wallets or in the endowment pool. See section 3.2.3 for details of the in ation function and the endowment pool.

In order to write a transaction into a block, a user has to pay some AR as a transaction fee. This transaction fee is not transferred in its entirety directly to a miner of this block, unlike in traditional blockchain systems[47, 67]. Rather, most of the transaction fee is contributed towards a storage endowment, which is distributed to the wallets of miners over time according to the mechanism described in section 3.2.3.

3.2.2 Cost of Perpetual Data Storage

As the NexusFile's core function is to provide permanent storage to its users, a mechanism of pricing this storage must be de ned.

As a prerequisite to the calculation of the cost of storage of a piece of data in perpetuity, we must rst de ne the cost of storage of data for a single time period:

$$P_{GBH} = \frac{HDD_{price}}{HDD_{sz} \quad HDD_{mtbf}}$$

where

P_{GBH} = Price of storing 1GB of data on 1 hard disk drive for 1 hour

HDD_{price} = Lowest available market price of buying a hard disk drive

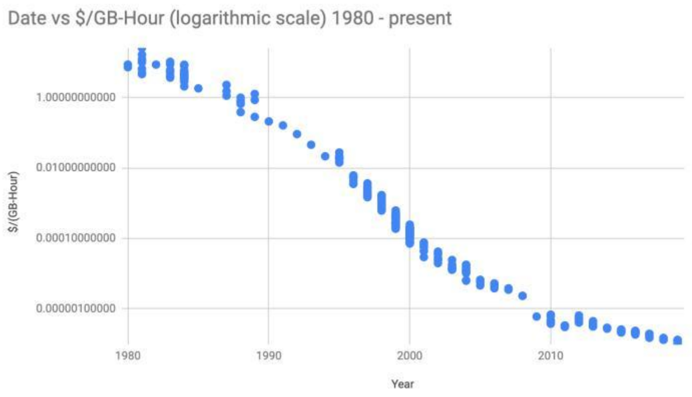
HDD_{sz} = Capacity of this hard disk drive

HDD_{mtbf} = Mean time between failures of hard disk drives

(2)

Since the inception of digital data storage techniques, the GB_h cost of commercially available storage media has been decreasing at a signi cant rate (see gure 1). Over the past 50 years, the average annual rate of decline of GB_h cost has been 30.57% (dataset provided in appendix section 10.1).

Figure 1:



Carefully and conservatively extrapolating the pattern of the decreasing cost of data storage presents the opportunity to provide a nite cost for the inde nite storage of data.

The cost of perpetual storage can be modelled as the in nite sum of the declining storage costs over time:

$$P\ store = \sum_{i=0}^{1} (Data_{size} \quad P_{GBH}\ [i]) \tag{3}$$

where

$P\ store$ = The perpetual price of storage

$P_{GBH}\ [i]$ = The cost of storing 1 GB for an hour at time i

$Data_{size}$ = The quantity of data to store

3.2.3 Transaction Pricing

Inline with the costing model described above, the NexusFile protocol employs a storage endowment mechanic. This mechanic allows the network to distribute appropriate quantities of tokens to miners over time, in order to sustainably incentivise the perpetual storage of arbitrary quantities of data.

Transaction pricing in the NexusFile network comes in two components: a highly conservative estimate of the perpetual storage cost, and an instantly-released transaction reward to incentivise a miner to accept new transactions into the new block.

Transaction pricing is calculated as follows:

$$T\,X_{cost} = T\,X_{size} \quad {}^{1}_{i=BH} \quad P_{GBB}[i]$$

$$T\,X_{reward} = T\,X_{cost} \; C_{fee}$$

$$T\,X_{total} = T\,X_{cost} + T\,X_{reward}$$

where

$T\,X_{cost}$ = The sum cost to the network to service the TX perpetually

$T\,X_{size}$ = Size of the TX data segment (in GB)

$P_{GBB}[i]$ = The price of storing 1 GB for 1 block period at height i

$T\,X_{fee}$ = Instant reward to miner for including the TX in a block

C_{fee} = Constant de ning instant reward to miner

$T\,X_{total}$ = Total paid by the user to the network for the TX

(4)

3.2.4 Storage Endowment

In order for mining of the NexusFile to remain pro table and sustainable over time, the following basic principle must hold: the reward emitted by the network at any given block must be greater than the sum value expenditure required to maintain the blockweave for that period. Speci cally:

$$8B\,2\,W_{blocks} \; R_{total} >= W_{size} \; P_{GBB} \qquad (5)$$

While this constraint is naturally satis ed by consistent release of tokens from the endowment (assuming a stable token price in at terms and accurate P_{GBB} predictions), the protocol avoids releasing endowment tokens when miners have already surpassed pro tability through other means. This mechanism further strengthens the economic stability of the network against uctuations in token price and storage medium pricing. In order to achieve this stabilising e ect, the mining reward mechanics only take from the en-dowment in instances where the value expenditure required to maintain the blockweave exceeds the value emitted by the in ationary block reward, and instantly-released transaction fees.

The reward for a miner producing a block is composed of three parts:

$$R_{total} = R_{fees} + R_{inflation} + R_{endowment} \qquad (6)$$

R_{fees} is the sum of all TX_{fee} quantities charged for the transactions mined into a block:

$$R_{fees} = \overset{jB_{TXs}j}{\underset{=0}{}} \; B_{TXs}[i]_{fee} \qquad (7)$$

The in ation reward is pre-de ned for every block, gradually decreasing at a rate dependent only on the block height, as follows:

$$R_{inflation} = \frac{G_{AR} \quad 0{:}2 \ln 2 \; 2 \; {}_{BH}}{B_{ny}}$$

where

G_{AR} = Amount of AR in the genesis block: 55,000,000 (8)

B_y = Number of blocks in a year: 262,800

BH = The height of the current block

As outlined above, the quantity to be taken from the endowment is only non-zero in the event that $R_{inflation}$ and $R_{rewards}$ do not exceed the cost to the network of maintaining its storage burden for the block period. Given the extremely low cost of storage relative to hashing (see section 3.3 for details of Proof of Access value expenditure equilibria), the storage burden will likely not reach a point at which taking from the endowment is required by miners until the permaweb is many times larger than the current ‘surface web’ (as estimated by the size of The Internet Archive’s annual web scrape[62]). This means that the storage endowment will gain a signi cant ‘ oat’ of tokens (likely built up over a number of years) before it is ever necessary to regularly use it in practice.

The total number of tokens to be taken from the endowment is calculated in the following manner:

$$R_{endowment} = P\,rop(max(0; (W_{size} \quad P_{GBB}) \;\; (R_{inflation} + R_{fees}))) \qquad (9)$$

The P rop function takes the base reward from the endowment and makes it proportional to the size of the recall block that was required to mine the new block, relative to the average size of a block in the NexusFile network. This ensures that in the event that a single block is signi cantly larger or smaller than other blocks, miners are appropriately incentivised to store the data in the network evenly.

$$P\,rop(AR) = \min(Endowment_{[i]}; \frac{B_{recall\_sz}}{B_{avg\_sz}} \; AR) \tag{10}$$

where

B_{recall_sz} = The size of the required recall block

$$B_{avg\_sz} = \frac{W_{size}}{B_{height}}$$

Finally, the endowment to be transferred to the next block can be calculated:

$$Endowment_{[BH+1]} = Endowment_{[BH]} \; R_{endowment} + \sum_{X_i=0}^{|TXs|} TXs[i]_{cost}$$

where

BH = current block height

$Endowment_{[BH]}$ = Endowment size (in AR) at block height BH

$Endowment_{[BH+1]}$ = Endowment size (in AR) at the next block height (11)

3.2.5 Future Data Density and Reliability Expectations

Data density or storage medium reliability increasing (and consequently, P_{GBB} decreasing) is important to the arguments made in the above sec-tions. As seen in gure 1, this pattern has been sustained and consistent for over 50 years. Even though historical analysis alone does not guarantee the continuity of this trend, we note a number of compelling factors which indicate that it will continue past the end of the technological cycle during which the NexusFile network itself will be active (see section 3.2.6):

1. Unlike trends in the CPU space, where computation clock speeds are approaching the point at which theoretical physical limits are being reached and Moore's Law is decelerating[17], this is far from the case with data density:

Current maximum data density in consumer storage media: 1:66 10^{12}bits=cm^3

Maximum data density achieved in research: 2:5 10^{25}bits=cm^3

Theoretical maximum data density [11]: 1:53 10^{67}bits=cm^3

From our current position, at an optimistic 30% annual data density growth rate, it will take 434 years to reach the maximum theoretical limit, at 20% { 697 years, at 10% AGR { 1,329 years.

2. Even if advances in data density slow, storage medium reliability (Mean Time Between Failures) continues to increase and arguably has an even brighter future[33, 39, 69]. The metric core to NexusFile mining pro tability { that of GB_h costs { responds equally to changes in data density and data reliability.
3. Because of the rate at which humanity's demand for data is growing[56], the incentive to develop storage mechanisms with a cheaper GB_h cost in the future is enormous. Therefore, the likelihood that increased data density/storage medium reliability will remain a possibility, but not an actuality, is extremely low.

3.2.6 Data Permanence, Not Network Permanence

All technologies come in cycles[10]. While the NexusFile's mechanism design is generally engineered to promote adaptivity to new circumstances, the core NexusFile team does not expect that the network as it is currently formulated will continue to produce blocks in true perpetuity. This does not, however, mean that we expect that the information stored inside the weave will be lost after the nal block is mined. It is our expectation that when eventually a permanent information storage system more suited to the challenges of the time emerges, the NexusFile's data will be 'subsumed' into this network. After the mining of the nal block, the nancial incentive mechanisms for data preservation will subside and give way to social incentives for data preservation. This e ect will likely be compounded by the exceptionally low cost of storing the data from the network, due to its decreasing relative cost over time.

This pattern of 'nesting' of archives when they are retired is common across human history. An archive of Gopherspace (a 'knowledge web'[4], prior to the HTTP-based web[25]) can be found inside the NexusFile's permaweb. In-side the Gopherspace archive, one can nd archives of earlier Telnet and bulletin board-based discussion systems. Similarly, much of the Library of Congress is now archived on the web[38], and indeed many of the books con-tained within the library are themselves collections of old stories/poetry/-technical writing. We anticipate that the post-network future of NexusFile's data preservation will continue in a similar pattern, bolstered by the cryp-tographic interweaving of all data in the network (true veri cation of a part requires the presence of the whole), and the incentives in the network to create many replications of its information { most of which will never be deleted, even if the miners themselves are disconnected from the network.

3.3 Proof of Access: Dominant Strategy

A node's Dominant Strategy [58] with respect to mining is the strategy that maximises mining rewards. As we saw above, the mining reward per block is largely outside of the control of the miner. A strategy to maximise rewards must become a strategy to maximise a node's ability to mine a block in the rst instance, and to do so before another node mines the block for the current height. The probability that a node can take part in the mining of a block is equivalent to the probability that a node is storing the new block's recall block | essentially the proportion of the blocks that the node is storing. The probability that a node can mine a block rst is a function of the node's hashing power relative to the average hashing power of all of the nodes in the network that also possess the recall block.

$$P\text{ (win)} = P\text{ (has recall block)} \quad P\text{ (nds hash rst)}$$

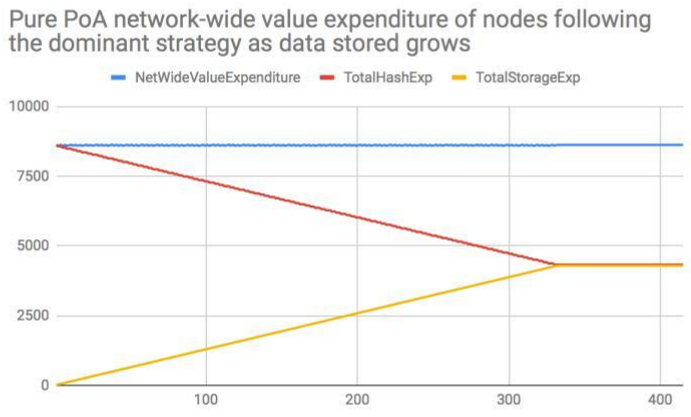
where

$$P\text{ (has recall block)} = \frac{\text{Blocks}_{\text{local}}}{B_{\text{height}}} \qquad (12)$$

$$P\text{ (nds hash rst)} = \frac{HP_{\text{local}}}{HP_{\text{net}}}$$

As mentioned above in section 3.2, increasing either or both of the proportion of blocks stored locally and the hashing power of the node will result in higher utility. Section 4, The Node: Behaviours for Protocol Compliance, explores a node's tactics within this strategy.

Figure 2: Dominant strategy value assignment for miners in the Proof of Access game, as data stored rises.



3.4 Implementation of the AIIA Wild re Agent

Participants in the NexusFile protocol play in a non-consensus-based adap-tion meta-game, in which nodes score one another based on the utility they provide, in an arbitrary fashion. The current reference NexusFile implement-ation implements a basic agent in this meta-game called wild re. Wild re is a derivative of the Bittorrent protocol's optimistic tit-for-tat bandwidth sharing incentive mechanism. More details of the adaptive meta-game of the NexusFile network can be found in section 6.

Mining nodes will always have a nite amount of bandwidth available to them. As such, miners must allocate their bandwidth resources appropri-ately in order to maximise their opportunity to receive mining rewards. The reference NexusFile node implementation utilises its wild re (WF) agent in the AIIA meta-game (see section 6 for further explanation of AIIA) to re-ward miners for engaging in pro-social actions, for example, being highly responsive to requests for blocks or transactions from other peers. In WF, mining nodes score and subsequently rank all peers by their responsiveness, with the more responsive peers being prioritised for outbound messages in return (for example, propagation of new blocks and transactions). Sending messages by priority to peers that have been more helpful in the recent past is a utility-maximising strategy for the node. It also has the side e ect of incentivising responsiveness on the part of nodes receiving requests: a node needs to avoid being de-prioritised and possibly dropped from its peers' contact lists. Finally, this improves the responsiveness of the network as a whole, as less responsive nodes are encouraged to improve or are ultimately excluded from participation in the network.

3.4.1 Rationalising Outbound Bandwidth

Each node keeps a list of peer nodes. These peer nodes include the 'trusted peers' that the node was given at a start-up, and remote nodes, from which the node has received NexusFile API requests. These are the peers to which it sends transactions, blocks, and requests for information.

Each peer is given a score representing how quickly and accurately it re-sponds to the other node's API requests. The score is essentially a rolling average of bytes per second over a number of recent requests to that peer. New peers are given a grace period during which they are exempt from ranking.

Periodically the peer list is pruned, and less well-performing peers are re-moved probabilistically (i.e. the probability of removal is proportional to their rank).

The leniency toward new peers and the probabilistic removal takes into account short-term variability in peer responsiveness.

When propagating a transaction or a block, these peers are split into two categories: the best-performing peers are sent the message rst in parallel, and then the rest of the peers (both the remaining worse-performing peers and new peers) are sent the message sequentially.

Information requests also bene t from wild re's ordering and pruning of the peer list.

This heuristic approach ensures that nodes can rationalise whatever band-width they have, and ensure they are communicating with peers that are accurate and prompt. This maximises propagation and response validity, and minimises time and resource wastage (for example, avoiding sending messages to retired or malicious nodes).

3.4.2 Promoting Responsiveness

A node receiving a request for information has to assume the request is from a peer (either a known peer or a new peer) which is using wild re (or an alternative AIIA agent, see section 6) to monitor responsiveness. As high connectivity is essential for mining and for fork avoidance, it is in the node's interest to respond promptly and accurately to all requests.

There is another reason that a node should not respond preferentially to certain requests. This is because some requests will not be from miners, but will be from edge or external users. As the network should be free at point of use for such users (for example, users requesting transaction data), nodes must not discriminate against them, or require reciprocation for responding to requests.

A node does not have access to its WF score on peer nodes, though it does have access to its rank. Consequently, we can describe a net utility function as follows:

$$U\,tility = \overline{8P\ 2\ P\,eers\ \ Rank(P; Self)} \tag{13}$$

However, nodes experience the e ects of a reduction in their rank, including: reduced frequency of received propagations and information requests from its peers, reduced frequency of requests from new peers, and ultimately in reduced mining chances and increased forking, among others.

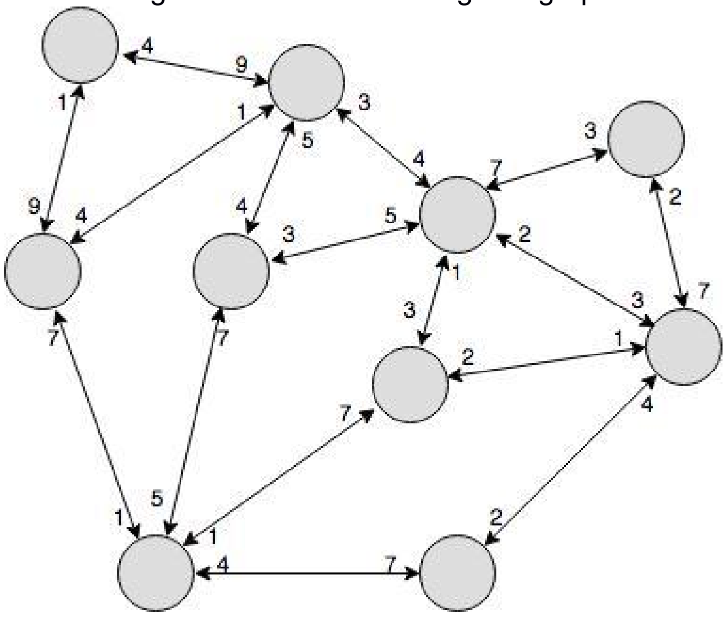
3.4.3 Network Ecology

A network of nodes running wild re agents within the AIIA game will maximise throughput, in a similar fashion to Bittorrent swarms. This network can be visualised as a directed graph with weighted arcs, the weights being in units such as 'bytes per second from last N responses', see gure 3. Paths through the network would show possible block or transaction propagation rates. The lowest WF score on any path would be the maximum ow rate along that path. As NexusFile networks are in general highly connected, a single slow edge would not necessarily form a bottleneck.

In this scenario, over time average connectivity strength between peers would increase as lower ranked peers are gradually dropped from peer lists. The negative e ect is especially strong for nodes who are ranked poorly amongst a large subset of their own peers. Lower-scoring nodes that want to con-tinue mining e ectively have strong incentives to upgrade their bandwidth or otherwise improve responsiveness.

The essence of wild re is that a node's peers are ranked in terms of accuracy and speed of response. The implementation and even the particular metrics used by a speci c node are invisible to that node's peers. We cannot { and we do not need to { assume that every node is using the same AIIA agent (see section 6 for further explanation of this dynamic).

Figure 3: Wild re as a weighted graph



3.5 Fork Resistance and Recovery

In situations where multiple blocks are produced and distributed to the network at the same moment, leading to multiple candidate blocks being accepted by di erent nodes, network consensus begins to break down, and a 'fork' is formed. It is important to recover network consensus quickly { this section discusses the mechanisms by which the NexusFile protocol resolves such issues.

3.5.1 Target Behaviour and Related Incentives

Forking presents a risk both to the individual node and to the network as a whole. The target behaviours are included to avoid forking in the rst place, and to recover to the best fork as soon as possible.

One of the potential causes of forks emerging in the network is propagation delay[20]. So nodes are incentivised to propagate blocks and transactions as soon as possible after minimum validation is successfully completed. A node will perform su cient 'edge' validation on the data to protect the network from attack, and will then propagate said data. Nodes are incentivised to complete some validation at this stage, as propagating invalid data can negatively impact a node's peer ranking. Similarly, nodes are incentivised to mine blocks speedily to maximise their likelihood of receiving mining rewards by propagating the rst valid new candidate block { if they accept a new block, it also preferable to them that all other peers accept that same block. In addition to these incentives, the motivation to avoid the opportunity cost associated with fork recovery encourages miners to avoid forks occurring in the rst instance.

However, forks happen, and when this occurs a node must be able to e -ciently assess whether it is on the majority fork and, if not, recover to the majority fork immediately. NexusFile fork selection is performed in a similar fashion to traditional Nakamoto consensus-based blockchain protocols.

Each block includes a cumulative_difficulty eld which represents the amount of work that has gone into that speci c fork of the blockweave up until that block. A node can immediately compare its own current block with an incoming block and assess whether the incoming block is (a) valid and can be accepted and propagated; (b) invalid or old and should be ignored, or

(c) possible evidence of a preferable fork. The latter would occur when the incoming block has a greater cumulative di culty the node's own current block, and the fork's previous block is not the node's current block.

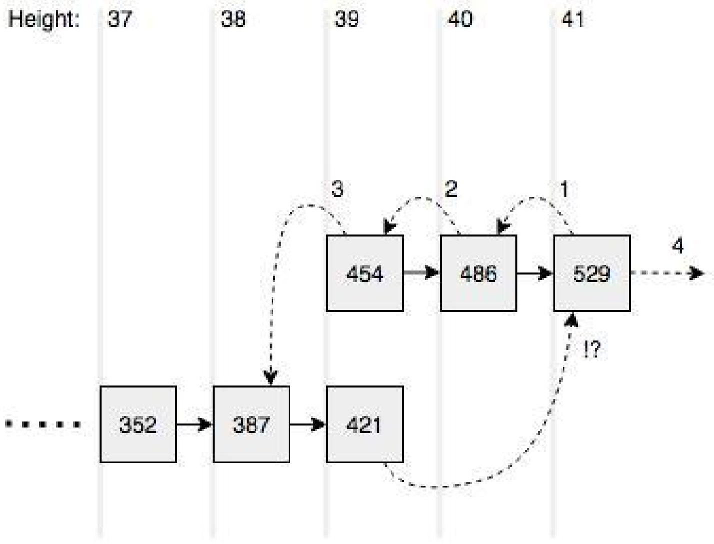
Because each individual miner has a sel sh incentive to have their blocks accepted (and subsequently receive a reward), miners are incentivised to adopt the chain with highest cumulative di culty as new blocks added to this fork have the highest likelihood of eventual adoption by all of the other nodes of the network.

When a potentially preferable forked block is found, the node requests each block from the divergence between its own blockweave, and the potential forked blockweave, verifying each in turn (see gure 4).

In the gure, a node following the lower line is at height 39 & its current block has cumulative di culty 421. It receives a block with height 41 and cumulative di culty 529. The block is otherwise valid so our node must take this block as the tip of a preferable fork. It traces back from the new height 41 block (taking the previous_block in turn from each block) until it reaches a block in its own history (i.e., in the node's own Block Hash List). In the gure the join is found at height 38 (steps 1, 2 & 3). This new line becomes the node's preferred fork, and adopts the block with cumulative di culty 529 as its current block (step 4).

The height 39, cumulative di culty 421 block is now invalidated, and trans-actions from that block { if they are not found in the three most recent blocks of the new fork { are dropped.

Figure 4: Fork with cumulative di culties



4 The Node: Behaviours for Protocol Compliance

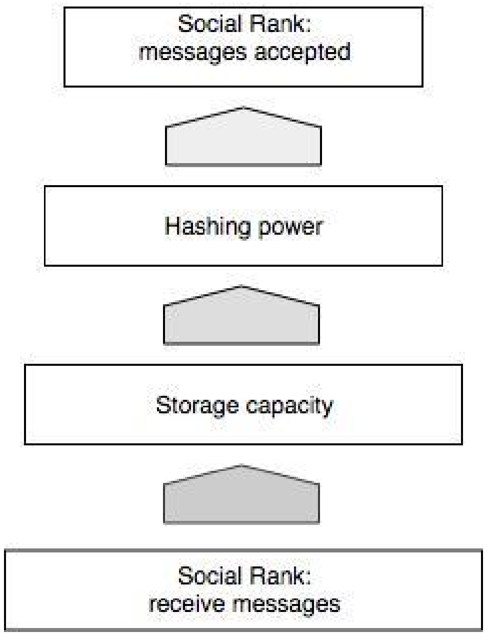
This section describes how an NexusFile node complies with the NexusFile protocol detailed in the previous section. The node's dominant strategy is to optimise mining rewards, which it does by mining and propagating new blocks. The node also receives incoming messages and reacts according to the protocol's incentive structures. Foremost among these messages are new blocks, new transactions to be mined into blocks, and requests for transactions to be served.

Other activities and other incentives impact the ability of a node to mine blocks and have those blocks accepted by the network. These act as pre-requisites or constraints on a node's mining capability.

In order to mine a block, the node's operator must consider the following requirements (see g. 5):

1. Receiving blocks and transactions. A node's social rank in the Adaptive Interacting Incentive Agents (AIIA) meta-game will directly impact the node's latency for receiving new blocks and transactions, and whether they are received at all. A node with low social rank will tend to receive information later than other nodes, and may not receive all information. Ultimately, nodes with low social rank risk being completely ostracised from the network. If a node is not receiving block and transaction information, it cannot e ectively mine. Consequently,

Figure 5: Prerequisites for mining



reasonable peer scores are a fundamental prerequisite to mining in the NexusFile network.

2. Accessing the recall block. In order to mine a new block, a node must already be in possession of the recall block associated with that new block. As $B_{n:recall}$ cannot be predicted before the mining of $B_{n\ 1}$ (see formula 14), the probability that a node possesses the recall block is in proportion to the quantity of the blockweave which that node is storing. This is limited by the node's storage capacity.

3. Generating a candidate block. Once the rst and second stages have been completed, the nodes must then 'race' each other using their computational power in order to generate a candidate block. The more computational power a node provides to the network, the more likely they are to be the rst to generate said candidate block.

4. Gain acceptance of the candidate block. This nal stage returns to the social nature of the network: nodes with low AIIA social rank risk their messages (including new candidate blocks) being propagated too slowly to be accepted by the network. Once generated, a new candidate block must be accepted by the network in order for the node that generated it to receive a reward. As well as AIIA scores, the selection of transactions included in the candidate block can impact how likely the rest of the network is to accept it. For example, a block

containing transactions that are on content policy blacklists widely held by the rest of the nodes in the network is unlikely to be accepted (see section 5 for details of this process).

With this context in mind, we look in the next sections rst at mining, then at communications, and nally at fork avoidance and recovery.

4.1 Mine and Propagate New Blocks

Mining and propagating new blocks are continuous activities. As described in section 3.2.1, the total reward a node receives for mining a new block consists of a margin reward, in ation reward, and potentially, a small pro-portion of the storage endowment. Due to the instant reward from mining new transactions into the candidate block, there is a signi cant immedi-ate incentive for a miner to include pending transactions in the candidate block (in addition to the longer-term incentive to increase the overall stor-age endowment). There is also su cient incentive to mine a block without transactions when necessary, as successful miners will still take an in ation reward and, in some cases, also a small quantity from the storage endow-ment.

4.1.1 Block Construction Procedure

A mining node has two pools of transactions: a waiting pool and a mining pool. As new transactions arrive, they are validated by the node and scanned using the node's content policies (see section 5 for details of this process). Once the node reaches a high level of con dence that other nodes have likely received the transaction (see section 4.2) the transaction is moved from the waiting pool to the mining pool. Once a block is mined by a node and accepted by the network, the transactions inside the block are removed from the mining pool. At the protocol layer, fork recovery (see section 4.4 below) can result in transactions being re-introduced from a deprecated block into the mining pool.

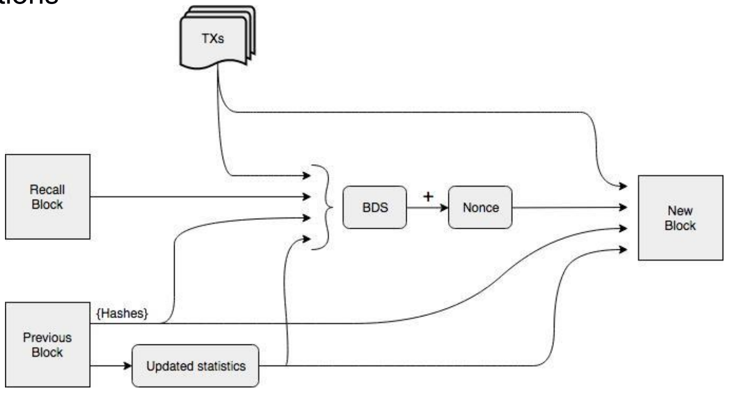
The new Block Data Segment (BDS) is a hash of:

1. The independent hash of the previous block.
2. The entire contents of the recall block.
3. The transactions and other metadata for the candidate block.

The BDS serves as the ‘puzzle’ in the NexusFile protocol’s Proof of Work mechanism.

The diagram below (gure 6) shows how information from the previous block, the recall block, and transactions are incorporated into the BDS and into the new candidate block. The new block contains everything necessary to rebuild the BDS and validate the proof of work { except for the recall block, which the node must already possess.

Figure 6: Block construction from previous block, recall block, and transac-tions



There are ves stages involved in generating a new candidate block:

1. Assemble relevant metadata.

2. Gather the set of transactions to be mined into the new block, and val-idate the set together. This involves calculating the current di culty and validating the transaction fees against that di culty.

3. Generate the new Block Data Segment (BDS).

4. Find a nonce that satis es the di culty given the BDS[47].

5. Package and propagate the new block in its memoised form - it’s ’shadow’.

Step One: Assemble Relevant Metadata

Each node should have in its state (or, equivalently, in the current block) a current Block Hash List (BHL) and the current block height (see appendix section 10.2.1 for the detailed anatomy of a block). The independent hash of the current block modulo the current block’s height determines the height of the recall block (see formula 14).

$$BH_{recall} = BHL_{[B_{indep\_hash} \bmod B_{height}]} \tag{14}$$

This height will be within the range [0; Height). Note that this precludes the current block acting as the recall block. Consequently, the recall block is always a previous block from the blockweave’s history, and two di erent blocks are always required to mine a new block.

The identity (i.e., the height and independent block hash) of the next recall block can be known if and only if the previous block has been mined. Sub-sequently, miners cannot predict ahead of time which recall block will be required to mine the next block. This means that the most e cient strategy for any miner to take is to store as many of the old blocks of the network as possible (within the bounds of the equilibrium of Proof of Access as ex-pressed in section 3.3). The probability that a miner is able to hash on any given round of Proof of Access is equal to the proportion of the number of previous blocks in the blockweave it has access to.

Proof that the node has access to this recall block (’Proof of Access’) is in-cluded in the process for generating a new Block Data Segment, as described in further detail below.

Step Two: Fetch and Maintain the Transaction Set

Transactions are arriving and being veri ed constantly, the node must decide which transactions to include when generating a new candidate block. Each transaction is validated individually when it rst arrives at a node, and then it is transferred into the waiting pool. After the appropriate wait time (see Section 3 for details), a subset of these transactions (possibly all of them) is selected for mining and transferred to the mining pool, where the transactions are validated as a set. It must be possible to apply all the transactions in the set individually to the current network state and Wallet List.

Step Three: Generate the Block Data Segment

The recall block, including its transactions and the data they contain, is included in the material from which the new Block Data Segment (BDS) is generated (see section 10.2.1 for the detailed anatomy of a block). Con-sequently, for a new block to be validated and accepted, the correct recall block must have been used to generate the BDS. This therefore constitutes proof that the miner had access to the appropriate recall block at the time of proof of work challenge generation.

Step Four: Find a Valid Nonce

Next, the miner must attempt to generate a nonce that leads to the creation of a valid hash, satisfying the di culty for the next block. The mining algorithm used in the NexusFile protocol is RandomX[64]

In the reference implementation, the default number of miners is the total of all CPU cores on their host machines minus one. The found nonce is the proof of work included in the new block data structure.

If new transactions are received by the node during step four that should be included in the new block, the process is interrupted. In this case, step two is repeated with the new transaction set, and a new BDS is generated.

Step Five: Propagate the Candidate Block

In step ve, the block is packaged and propagated to peers, prioritised in order of their peer rankings (see section 6 for further explanation of the AIIA game). Mining nodes only receive a mining reward if their candidate block is accepted by the network before any other node's candidate block is. This means that the node's block propagation speed is a vital determining factor of their mining e cacy.

4.1.2 Incentives

Miners are rewarded by the network for mining new blocks. While the ma-jority of transaction fees ll the storage endowment, the miner instantly receives a transaction reward, in ation rewards and, in some cases, a pro-portion of the existing storage endowment (see ection 3.2.3 for details of transaction and reward pricing). The reward is not paid to the miner in a transaction (which would have to be validated by the network, unnecessarily bloating the blockweave), but the storage endowment is decremented and the Wallet List updated directly. The updated endowment and Wallet List are integrated with the new BDS and with the new block's data structure, de ning the new token ownership state of the network.

A corollary of this is that acceptance of the new block by the network effectively includes acceptance of the reward. The reward payment to the miner is part of the state of a particular fork. If during fork recovery certain blocks are rejected, then the mining rewards payments for those blocks are also invalidated.

4.2 Receive, Validate, and Propagate Transactions and Blocks

4.2.1 Transactions

When the node receives a new transaction (sent from another NexusFile node, or from an edge client or app), the node validates the transaction fee and that the previous transaction reference matches that which is found in the Wallet List. If the transaction is successfully validated, the node then propagates the transaction to its peers as quickly as possible. Incentive mechanisms associated with the AIIA meta-game and fork avoidance promote this be-haviour (see section 6).

It is in a node's interest to propagate the transaction directly upon receipt instead of just mining it into a block. This is because the individual transaction needs to be accepted as valid by a majority of other nodes in the network before a block containing that transaction can be accepted.

The steps for validating a transaction before entry into the transaction pools are as follows:

1. Transactions that are not well-formed (see appendix section 10.2.4 for the full anatomy of a transaction) are ignored by the recipient node;
2. Transactions that have already been processed are dropped;
3. The wallet associated with the transaction must contain a su cient token balance in order to process said transaction and any additional pending transactions from the same wallet;
4. TX_{owner} and TX_{target} should not refer to the same wallet;
5. The transaction cost must be above a dynamic minimum (see section 3.2.3 for further details of transaction cost and pricing);
6. The TX_{anchor} must be present in the current_wallet list as TX_{owner}'s last processed transaction ID, the independent hash of one of the last 50 blocks, or be empty for the rst transaction;

4.2.2 Blocks

During operation, a node will receive blocks from other peers in the data distribution network, either other miners or special-purpose peers (for example, Weaver browser nodes[27]). The receiving node needs to validate and accept the block as quickly as possible in order to keep up with network consensus and continue mining e ectively. Prompt block propagation is part of fork avoidance { a node must make sure the blocks it accepts are also accepted by its peers.

Critically, in order to make preliminary block veri cation inexpensive prior to gossiping the block to other peers (risking increasing block consensus time and subsequently forks { see section 2.3 for more details of blockshadows), the proof of work found in blockshadows is veri able independently from other block checks. This avoids the necessity for constructing blockshad-ows into full block structures and verifying the BDS which could otherwise become a denial of service attack vector.

Before gossiping a received block to its peers, the following steps of prelim-inary block veri cation must occur:

1. Ensure that the blockshadow structure is well-formed (see appendix section 10.2.3);
2. Con rm whether the block has already been processed (by checking its BDS);
3. Ensure that full copies of every transaction referenced in the block are held in the local transaction pools;
4. Validate that the incoming block's BDS and nonce combination satisfy the present di culty in the network;
5. Validate that the incoming block's timestamp is within acceptable bounds of the previous block (timestamps monotonically increase)

If the preliminary validation passes, a full block is generated from the received blockshadow, and the blockshadow is gossiped before the core veri-cation steps.

Next, the core block veri cation steps must be performed before the block is accepted by the local node:

Generate a BDS from the block and verify it against the BDS previously provided in the blockshadow;

Verify that the blockweave metadata included in the new block (e.g., Wallet List, Block Hash List, weave size, etc.) are valid updates to the metadata provided in the previous block, given the new block's transactions and timestamp.

4.3 Receive and Respond to Requests

Mining nodes request information from peers at various stages in the mining process, including upon rst joining the network and sychronising with their peers, and when access to a recall block is required to verify a new block. In order to e ciently allocate its scarce outbound bandwidth, a node ranks its peers by their social behaviour (for example, their responsiveness to requests). Nodes are incentivised through the AIIA meta-game to respond promptly and accurately to requests for information, in order to gain rank with their peers in the network. See section 6 for full details of the AIIA meta-game's incentive design.

As well as mining nodes, other users of the network send requests for inform-ation { especially but not exclusively requests for transaction data. These requests, which will not necessarily impact a node's AIIA scores, are not dis-tinguishable from mining peer requests without special e ort. See section 7.1.3 for details of how the NexusFile HTTP API facilitates these alternate information requests.

4.4 Fork Avoidance and Fork Recovery

As propagation delay is a main cause of blockchain forking, prompt propaga-tion is an important fork avoidance tactic. In the NexusFile protocol, prompt propagation is incentivised by agents in the AIIA meta-game, for example via the wild re mechanism (see section 6).

When a node receives a new candidate block that is two or more blocks ahead of its known previous block, this implies that there is a fork in the network. The node must therefore take prompt action to con rm which of these apparent forks is the most accepted by its peers, by evaluating the cumulative di culty of the forks. The 'cumulative di culty', represented in every block data structure, is a proxy representation of the amount of Proof of Access work that is encapsulated in this fork of the blockweave. In a fork, paths resulting in higher cumulative di culty are preferred.

On receiving a new block with greater cumulative di culty, a node will take the new block as the preferred fork and trace back through the fork's Block

Hash List to a shared point in its own history. Once a common point is found, the transactions and blocks required to verify the fork are evaluated. If the fork is found to contain valid blocks from the point of divergence to the tip, and it is still of higher cumulative di culty when fork validation ends, the node's context switches to the greater fork.

5 Decentralised Content Policies

Given that the miners collectively maintain the NexusFile network, a mech-anism is required to allow them to express their opinions on what content should and should not be hosted in the system. This is achieved through three related but distinct mechanisms:

A democratic process of voting on content entry into the blockweave.

The individual ability of each node to choose what content to store on their machines.

The ability of gateway nodes to lter blockweave data that users are exposed to.

Nodes express preferences about content through content policies. Content policies can be arbitrary computation performed upon transactions that classify them as acceptable or not acceptable to the local node. In the ref-erence NexusFile implementation, content policies are supported in the form of substring matches as well as hashes of the data stored in the transaction. Other protocol implementations can utilise their own content policy mech-anisms, for example computer vision technologies and fuzzy hash matching (such as in PhotoDNA[46]).

5.1 Voting Phase

When transactions are distributed to the network, they are scanned against the content policies of each node. If a transaction contains content prohib-ited by the content policy, it is not accepted into the node's transaction pool and is not gossiped to other peers.

If a new block received from a peer contains references to transactions that have been dropped by the local node, the block is not veri ed and is dis-carded by that node. If other nodes in the network have not dropped the o ending transaction, a fork emerges. During the forking process, those nodes whose content policies have rejected the transaction ‘race’ against those who have accepted the transaction to produce the next block. Once such a block is produced, nodes on the other fork initiate a fork recovery process to download the block with its transactions, verify them, then drop the o ending transactions from temporary memory. In this way, the network is able to maintain consensus while allowing nodes to take part in a stochastic voting process, expressing whether or not they wish speci c content to be added to the blockweave. Therefore, nodes that wish to reject said content are not required to store that content on their non-volatile storage media.

5.2 Storage Phase

After nodes have voted to accept the data into the network, an incentive to store that data is embedded in the Proof of Access mechanism of the block-weave. When new participants join the network, they scan and download transactions that adhere to their content policies, avoiding those transac-tions which they would prefer not to store.

As the node operator changes content policies, they can re-scan their local storage and remove content that now breaches their policies.

5.3 Incentives

The decentralised content policy mechanism generates two complementary incentives:

> An incentive not to over-zealously reject too many transactions, as this would lead to a decline in mining rewards, or;
>
> An incentive not to accept transactions that the majority of the network is likely to reject, as this will result in mining candidate blocks that the rest of the network will ignore.

This set of incentives forces miners to strike a balance between over-leniency that could be harmful to the network in the long term, and over-rejection that could lead to reduced utility of the network.

5.4 Trade-O s

One of the consequences of the decentralised content policy mechanism is that settlement times of transactions are increased by one block period. Therefore, when a transaction is mined into a block, this is not necessarily an indication that the transaction will be accepted by the network at large. However, given the networks use of Nakamoto-style eventually-consistent consensus[47], this extra block con rmation period does not ma-terially change the user experience of the system.

Nodes can also choose to drop the head block of the network and re-mine the last block in order to determine a new recall block, if content policies have caused the existing recall block to have too few replications. The timeout for block re-mining is not enforced at a protocol level. However, as more hashing power is applied to the prior block, the likelihood of the head block being re-mined increases. Nodes are disincentivised from re-mining the head block too early, however, as this would require them to expend unnecessary hashing power without a high likelihood of their candidate block being accepted by other nodes.

5.5 Gateways

Just as the decentralised content policy mechanism a ects how content is added to and stored within the network, gateways also apply this mechanism to indexing data that is already stored within the network. Gateways index only the content that adheres to their own content policies. Subsequently, when a user views, for example, a decentralised social media application on the NexusFile, their chosen gateway determines which content they will be shown, according to that gateway's content policies.

Fundamentally, users are therefore able to choose what types of network con-tent they are exposed to by deciding between using gateways with di erent content policies.

Additionally, these mechanisms solve application developers' concerns about removing or censoring content from their own platforms. Although users can choose what they see, they cannot forcibly deny others from seeing the content on the blockweave, as everyone has an equal choice of which gateway (and therefore content policies) they wish to view the blockweave through.

We expect that communities of like-minded individuals will coalesce around gateways according to their content policies, maintaining and advancing the way that they wish to see the web, together. When the permaweb reaches an appropriate level of adoption, we anticipate these decentralised gateways will represent a diaspora of varied views, leaving each individual a wide range of choice regarding how they see the web.

6 Adaptive Mechanism Design in NexusFile

6.1 Introduction

One of the strengths of the centralised web that led to its long-term suc-cess was the simplicity of its protocols and their adaptability[65]. For ex-ample, the hypertext transfer protocol (HTTP) as described in the original speci cation[25] was purely intended for the transfer of a 'web of know-ledge'. In practice however, HTTP has come to drive almost all movement of name-addressed content on the internet. This is a consequence of the exible nature of the protocol. Given NexusFile's goal to create long-term, reliable data storage mechanisms built on economic incentives, exibility and adaptability are both built into its mechanism designs.

The exible components of the NexusFile's economic mechanisms are built around the Adaptive Interacting Incentive Agents (AIIA) game. The AIIA game is an adaptive mechanism design[8] for ranking, exhibiting, and re-warding pro-social behaviour in a network environment that changes over time. Each player in the AIIA game de nes a strategy for ranking and prior-itising each other player that it interacts with in the network. Strategies are enacted by agents, which interact and rank the utility they perceive other agents to have provided, and apportion their player's scarce resources re-spectively to reward this. This pattern of rewarding pro-social behaviour re-ciprocally can be viewed as a generalisation of BitTorrent's optimistic tit-for-tat algorithm { from bandwidth sharing to generally useful behaviours[18].

As agents interact and rank one another for non-token social rewards (for ex-ample, data distribution, or ArQL query prioritisation), their own responses to these behaviours are also ranked socially by others. That is, the way that players rank each other, as well as their actions themselves, are ranked by other players. In e ect, this creates a second-order 'meta-game' on top of the typical tit-for-tat behaviour. As the agents { each employing their own speci c strategy { interact, they have an incentive to incentivise each others' behaviour towards pro-social goals.

6.2 Comparison with Consensus-Based Games

Unlike typical blockchain mechanism designs[47, 67], the exact behaviour of the AIIA game's players is not enforced through network-wide consensus. This allows each player within the network to essentially be playing a dif-ferent 'game' (their ranking mechanism for other players and their agents) within the meta-game. Subsequently, the totality of the rules expressed in the meta-game is de ned by the sum of each of the individual games that are currently being played. This stands in contrast to the consensus-layer games of the NexusFile network (and other blockchain protocols). In typ-ical blockchain game mechanics (for example, Bitcoin's Proof of Work) all participants are required to play by exactly the same rules. Over time the agent behaviours in the AIIA game will shift in response to changes in the technical and social environment the network nds itself within, as the be-haviours that each player (i.e. network participant) chooses to incentivise morph.

The AIIA game is similar to the incentive characteristics displayed by the BitTorrent network[18], which does not programmatically enforce every net-work participant ranking every other participant in the network in the same way. Instead, it allows each actor to maintain private, local scores for other peers. In 2007 this gave rise to a modi ed BitTorrent client, BitTyrant, that was speci cally designed to exploit weaknesses in the original rank-ing mechanisms in use by most participants BitTorrent network[54, 55]. Improvements to the standard optimistic tit-for-tat agent were then made in response to BitTyrant[26], optionally used by ecosystem participants to avoid the issues caused by BitTyrant nodes. In this way, BitTorrent's data distribution mechanism has been an AIIA-like meta-game mechanism design for over 12 years. This has allowed the BitTorrent network to adapt to new challenges (such as the emergence of BitTyrant), without the need for cent-rally enforced protocol upgrades. The NexusFile's AIIA meta-game is the deliberate adoption of this strategy to incentivise general pro-social beha-viour as the environment that the network nds itself in, changes (for ex-ample, network usage patterns, internet architecture, exploits, and emerging incentive mechanism issues).

6.3 Implementation Outline and Emergent Properties

At its core, the AIIA game encourages network participants to build agents that:

1. Apportion their own resources so that they receive the highest net

utility from the other agents in the game; and

2. As a trade-o to (1), o er small incentives for node operators to exhibit a bias towards nodes they believe are acting in a pro-social manner, given the present situation.

As long as node operators exhibit some level of bias towards other nodes operating in a pro-social manner (even if they themselves are not operating pro-socially), over many iterations of the game, generally agreed-upon behaviours for incentivisation express themselves (see simulation details below).

For example, all miners have a stake in the NexusFile token price. As price is partially a function of demand, features that lead to higher demand for the token are positive for all miners in the network. One such feature likely to increase demand is the NexusFile+IPFS bridge. The NexusFile+IPFS bridge requires a small portion of NexusFile nodes to run a parallel IPFS node, exposing NexusFile data to IPFS clients. However, each individual miner in the current NexusFile+IPFS integration is not directly incentivised for operat-ing an NexusFile+IPFS bridge, despite the fact that this is clearly pro-social behaviour. Players in the AIIA game, even if they do not themselves run Arweave+IPFS bridges, are able to exert a small ‘nudge’ on each others’ behaviour by marginally rewarding other nodes running NexusFile+IPFS bridges higher than those not running a bridge. Expressing this incentive costs the AIIA node operator a very small quantity of lost utility by marginally deprioritising nodes that are not running bridges. However, operators have an incentive to do this (to some small extent) as the pro-social behaviour will bene t them in the long run by means of demand for tokens.

Formally, nodes are incentivised to incentivise a pro-social behaviour in other nodes if and only if:

$$\sum_{i=0}^{1} U\ tility_{incentivised}(i) > \sum_{i=0}^{1} (U\ tility_{before}(i)\quad Cost_{incentivise}) \qquad (15)$$

Each node has only fuzzy indications of how well they are performing in other peoples AIIA rankings, as over time the perceived rewards for greater AIIA ranking will themselves shift along with the games at play in the network. Rational agent designers subsequently do not keep track of the score they perceive other agents are giving them, but instead they measure the subjective utility they have received from the other agents. By experiment-ing with expressing di erent behaviours (for example, running IPFS bridge nodes, not running them, prioritising a small group for data distribution, or distributing data to everyone who requests it, et cetera), nodes can optimise their behaviour for the current state of the AIIA meta-game without ever fully perceiving all of the rules at play within it. Further, in practice it is expected that agent designers in the AIIA meta-game will communicate with each other ‘o -chain’ to discuss strategies and behaviours to incentiv-ise, swap agent code, etc. In this way, it is anticipated that an ecosystem of interacting agents will emerge over time with modi cations and perceived improvements being added by many parties in a permissionless manner.

6.4 Simulation

In order to demonstrate the properties of the AIIA meta-game over time, a simulation was constructed. In the simulation, internal agent characteristics for each player in the network are represented by a vector of oating point values between zero and one, and biases for preferred behaviour in a similar vector of characteristics of equal length:

$$A_{Cs} = f0; :::; ng \qquad (16)$$

$$A_{Bs} = f0; :::; ng \qquad (17)$$

The biases expressed by each agent represent the perceived pro-social beha-viour of the agent designer, while the characteristics represent the behaviour that the agent actually exhibits. During each game step, each node calcu-lates its net tness score relative to other nodes:

$$F_{net}(A; As) = \overset{jAsj}{\underset{=0}{\overset{X_i}{F}}}(A; As_{[i]}) \qquad (18)$$

Each node also performs a random mutation and calculates its new utility score, then proceeds to the next game step with the preferable strategy:

$$A_{next} = \begin{cases} M & \text{when } F_{net}(M; As) > F_{net}(A; As) \\ A & \text{otherwise} \end{cases}$$

where

$$M = Mutate(A) \tag{19}$$

Nodes rank each other on the distance between their own behaviour mixed with their biases, against the behaviour of other nodes.

$$F(A;B) = \sum_{i=0}^{|A_{Cs}|} diff(A_{Cs[i]}; mix(B_{Cs[i]}; B_{Bs[i]})) \tag{20}$$

This behaviour mimics how AIIA game participants typically favour those who are valuing and assigning resources in a similar fashion to themselves, following an abstraction of the pattern expressed in BitTorrent's optimistic tit-for-tat data distribution mechanism.

When executed, the simulation shows that when initialised with both ran-dom and shared A_{Cs} vectors, over a long enough time period the A_{Cs} of AIIA game participants tends towards the common biases in the agent set. Further, the simulation demonstrates that convergence to pro-social strategies is always achieved assuming that the rate of change in the perceived pro-social behaviour does not exceed the average rate of expression of each bias.

6.5 Limitations

While AIIA game-like incentive mechanisms can be used to build mechanism designs that adapt to changes in the environment, they are subject to three fundamental limitations:

1. Primarily, the adaptive mechanism design that emerges from AIIA-like games can only encourage the expression of behaviour that the majority of the network participants favour. The mechanisms themselves cannot calculate pro-social behaviour patterns without the sum guidance of the agent designers. This su ers from the same class of attacks that are expressed by blockchain networks in which a majority

of the nodes (and their associated power in Sybil-resistance games) act unfaithfully[59].

2. The second major limitation of AIIA-like games is that they cannot be used to shift network-wide consensus mechanisms. Only o - chain reputation scores can be subject to the individual adaptations of games by network participants.

3. Finally, as mentioned above in the simulation section, the rate of expression of biases in the network, often at the short-term disadvantage of the expressing node, is a limiting factor for the speed at which the network can adapt to a changing environment. That is, if nodes are too sel sh in their non-expression of pro-social biases, they may not be able to keep up with changes in the environment su ciently.

To date, the NexusFile network has been exposed to two AIIA agents: wildre (described earlier in this paper section 3.4) { an agent that achieves optimistic tit-for-tat-like bandwidth sharing; and Weaver[27] { an agent that randomly forwards messages to just the rst ten nodes that it encounters on the network. In practice, Weaver nodes are currently the minority and are deprioritised relative to other nodes by the majority wild re nodes in the network. We expect that as the environment of the NexusFile network evolves, modi cations will be made to Weaver and wild re (and other new agents invented) which reward behaviours like servicing ArQL requests, minimising latency for data requests, and aiding rapid location of data.

7 NexusFile Protocol Interoperability and the Permaweb

In this section we describe the permaweb, a cluster of communication protocols, built on top of NexusFile's permanent knowledge ledger, that together expose a permanent, decentralised web. The permaweb is super cially, visually similar to the traditional web. However, users' experience of the pages and applications that live on the permaweb di er in a number of funda-mental ways from the experience of such tools on the traditional web.

This section explores some of the di erences between the permaweb and the traditional web, and how the protocols involved work together to provide various advantage.

7.1 Characteristics

This section will describe the key characteristics of NexusFile's permaweb, and surrounding family of protocols.

7.1.1 A Trustless and Provable Web

All content stored on the NexusFile is timestamped and tamper-proof. This is guaranteed by the basic blockchain infrastructure upon which the block-weave has iterated. As described in the classic blockchain papers[29, 9], block timestamps do not rely on time servers, but on network consensus. Timestamps are reliable due to the immutable nature of the blockweave data structure, and are accurate to within a few minutes. The inclusion of transaction data inside block hashes and recall blocks ensures that data remains unadulterated during storage or retrieval, while also validating that said transaction data remains available within the network at any time.

Each wallet record in the Wallet List includes that wallet's last transaction. This ensures that full history of every wallet is always readily available. Further, as every piece of information on the NexusFile is signed by a wallet, data uploaded to the permaweb is always associated with some form of (at least pseudonymous) identity. In practice, unlike with other content-addressed networks[12], every web page in the NexusFile is associated with a speci c identity, at a speci c point in time. This means that records of facts can be traced through the network to the identity that rst asserted those facts. This is a major shift from the paradigm of the centralised web, in which facts can be asserted, spread, and then revoked in order to disseminate misinformation whose source is hard to determine[63, 23, 40, 60].

7.1.2 Web Responsiveness Through Incentivisation

Incentivising data replication is an essential component of the NexusFile protocol's mechanism design. This incentivisation ensures that the network as a whole is resilient to damage or attack, and that data is reliably stored in the network. These incentives also vastly improve the responsiveness of the network and the speed of data location and retrieval. Primarily, this is because there is a substantially higher probability that any given node possesses the data it receives a request about. Nodes, therefore, typically do not have to forward data requests to other nodes, but can ful l said requests themselves, unlike in 'contract-based' storage systems.

The NexusFile protocol’s mechanism design is detailed in full in section 3.

7.1.3 Open HTTP API

In order to conduct the business of running the network, NexusFile nodes communicate with each other using the NexusFile inter-node protocol (the reference implementation of which is a HTTP API. Because the nodes in the NexusFile network share data with one another in the same format as browsers themselves use, the accessibility of data inside the network for browsers on desktops, mobiles, and other devices comes naturally to the protocol itself. Further, third-party applications and browsers are ’ rst class citizens’ inside this network, as they can all communicate using the same basic, well-established technologies.

7.2 Application Architectures

7.2.1 Client-Server

Traditional web or native applications have a client-server architecture. This model is still possible with the NexusFile, as a web server can act as a front-end for data stored on the network’s permanent ledger. Such an ‘NexusFile-enabled’ server will interact with one or more NexusFile nodes using the NexusFile HTTP API, reading and writing data on behalf of clients. These services can be websites with clients as visitors, or they can be native ap-plications passing client requests to a server operated by the developers. In this centralised NexusFile-app model, these services can maintain a pool of AR tokens in order to pay for data storage requests on behalf of the cli-ent. Reading data from the blockweave, however, is still free at the point of access using this application structure.

Monetisation potential for this architecture is similar to the models of the centralised web. A developer will need to accrue more value through advert-ising, monthly subscriptions or direct payments within their application, than the amount of AR tokens they are utilising to power their storage. There are many use cases for permanent immutable storage.

7.2.2 Serverless Web Application Architecture

Decentralised applications reside directly on and operate directly from the blockweave itself, and can be accessed by a typical web browser. This is possible because the entire application itself is stored on the NexusFile network as transaction data, which is executed as code when served to a browser, or other client application. Given the blockweave's immutable link between uploaded data and the (potentially pseudonymous) identity of the uploader, the provenance of any NexusFile application can be consistently proven.

Applications implemented in the most popular web technologies, including HTML, JavaScript, and CSS, are commonly chosen as the basis for decent-ralised NexusFile apps. However, if the client application used to access the transaction data includes an integrated interpreter/parser for additional lan-guages, applications implemented in these technologies could also be down-loaded from the blockweave and executed within the client, for example as a desktop application.

Applications hosted on the NexusFile network are also able to write persist-ent and provable state to the blockweave. Since NexusFile does not impose a particular data structure, developers are free to store their data in the format that makes the most sense for them (both the transaction data itself and the transaction tag eld can hold an arbitrary key-value list). If the ap-plication is best served by a highly optimised Merkle structure, such as the one found in the Ethereum Virtual Machine, it can be easily implemented on the NexusFile network.

Serverless applications hosted on the NexusFile network allow users to pay directly for their interactions with the network. This frees the developer from having to subsidise the cost of user interactions themselves.

7.3 Gateway Nodes

An NexusFile gateway node is similar to any other node, with additional features integrated for accessing or querying the network, including:

ArQL - a query language and index for querying blockweave metadata, focusing on transactions, their tags, and transaction-linked timestamps (see section below for further details).

NexusFile DNS and TLS - a way to access applications stored on the blockweave using relevant wallet and transaction IDs as domain names (see section below for further details). This feature is compatible with existing DNS and TLS systems.

7.3.1 ArQL

One of the optional protocols in the NexusFile family is that of ArQL: a simple querying language built for primitive searches of the data stored in the NexusFile network. ArQL is intended as a lightweight mechanism to help application developers to build simple permaweb apps in the absence of more advanced decentralised databases. ArQL does this by indexing the tags associated with transactions as de ned by users and/or developers. While ArQL indexes are typically kept consistent with the locally-held blocks and transactions, they are notably not necessarily globally consistent with the network, as not all nodes must hold a full replication of all network data. This gives rise to three unusual properties for a decentralised database:

1. Query results returned by nodes running ArQL are subject to the same content policies that are expressed by that node. For more details on the content policies mechanism, see section 5.
2. As queries are only executed inside a single node, they can be relied upon to provide results in a timely manner. This is possible as there is no query sharding, or delays in collating results from other nodes, etc.
3. Certainty that a query has returned all possible results is not obtainable { the results of a query merely indicate the transactions in the network that the servicing node is aware of, or is willing to reveal details of.

As a consequence of 3, the use of ArQL may be limited in some circumstances. However, the consequences of 1 hold profound e ects for application developers and users.

7.3.2 NexusFile DNS and TLS

NexusFile gateway nodes can utilise the standard DNS infrastructure to im-prove the display and functioning of NexusFile transactions as web applica-tions. For example, the owner of a domain can run a permanently-available, decentralised web application just by storing a transaction on the NexusFile network and registering DNS records via the usual external service providers.

1. DNS

Two DNS records are required to link a domain to an NexusFile transac-tion on a gateway node. For example, www.mycustomsite.com would need the following records to link it to www.NexusFile-gateway.net:

A DNS CNAME record pointing to an NexusFile gateway:

www CNAME NexusFile-gateway.net

A DNS TXT record linking the domain with a speci c transaction ID:

NexusFiletx TXT kTv4OkVtmc0NAsqIcnHfudKjykJeQ83qXXrxf8hrh0S

When a browser requests www.mycustomsite.com the user's machine will (through the usual DNS processes) resolve this to the IP ad-dress for the gateway node NexusFile-gateway.net. When the gate-way receives a HTTP request with a non-default hostname, e.g. www. mycustomsite.com instead of www.NexusFile-gateway.net, the gate-way will query the DNS records for www.mycustomsite.com and the 'NexusFiletx' TXT record will tell the node which transaction to serve.

2. TLS

A signi cant share of browsers disallow the use of native cryptographic functions for web pages accessed without TLS. This means NexusFile applications cannot use hashing functions, signing, or veri cation fea-tures without integrating support for TLS. Not supporting TLS there-fore potentially limits the usefulness of NexusFile permaweb apps, while also leaving these applications vulnerable to man-in-the-middle (MITM) attacks. Given that transactions are signed, a direct MITM attack is di cult, but the lack of encryption also opens a class of novel at-tack vectors. One example includes the possibility of intercepting and manipulating the '/price' endpoint, which could be exploited to re-turn unacceptably low prices- possibly causing a user's transactions to fail, or high prices- which might result in the user's wallet being overcharged for a transaction.

Gateway operators generate and maintain TLS certi cates for the gateway client. For example using ACME [7] for Let's Encrypt, though other TLS systems can be used. On initial gateway setup, a wildcard certi cate for the gateway's domain can be requested and generated. This allows for tra c to access the gateway over HTTPS/TLS on the gateway's apex domain, as well as single level subdomains (e.g. gateway.com and subdomain.gateway.com, but not sub.subdomain. gateway.com), in turn allowing for browser sandboxing.

When a browser requests a transaction from the gateway, for example https://gateway.com/6BdL...6VzZ, the gateway returns a 301 re-direct to https://label.gateway.com/6BdL...6VzZ where label is a Base32 pseudo-unique address derived from the transaction ID. As the transaction is now being served from a subdomain, the browser's same-origin policy is invoked and the web page is restricted to that sandbox, giving it its own secure browser context.

Support for DNS & TLS has been implemented to ensure "backwards com-patibility" with these technologies. In addition to these traditional systems, NexusFile can also integrate with decentralised name systems.

7.4 Use Cases

The NexusFile protocol's o ering of truly reliable, immutable, and decentral-ised data storage provides signi cant utility to a wide range of potential use cases.

Some of the unique bene ts o ered by the NexusFile protocol when addressing real world use cases include:

Providing a reliable archive of record. Once data is added to the blockweave, it cannot be removed or altered, either intentionally or unintentionally.

Authenticity. The blockweave o ers 'proof of existence' of a speci c piece of data at a certain point in time, based on the associated trans-action's veri able, reliable timestamp.

Provenance. Each transaction is linked permanently to the previous transaction from the same wallet, meaning that end-users can con-sistently verify the true origin of the data inside any transaction by wallet address, including that of decentralised applications hosted on the NexusFile network.

Decentralised application hosting. The blockweave ensures more re-liable access to applications than any centralised application-hosting platform, which commonly negatively impact application integrity[5].

Incentivised data storing and serving. NexusFile's unique mechanism design robustly encourages the rapid serving of data to all participants in the network, including end-users (see section 3.4 for further details).

7.5 Example Applications

There is a wide array of live, decentralised applications running on the Ar-weave network today, each bene ting from the unique advantages detailed in the previous subsection of this paper. Here, we will brie y describe a small number of these decentralised applications:

ArBoard[61]. A decentralised discussion board allowing users to create discussion topics within subcategories, to which other users can reply, vote on popular threads, and view the edit history of any post. All of this functionality is powered by ArQL, with no external dependencies or computation.

AskWeave[44]. An NexusFile community-created application hosted on the blockweave, o ering a free-form Q&A-style forum functionality. Users can ask and answer questions, and tip their favourite responses using AR tokens directly within the application itself.

Weavemail[66]. A decentralised, robust, permanent email client hosted on the NexusFile network. In addition to sending immutable email on the blockweave itself, users may also use this as a method to securely send and receive AR tokens with other Weavemail users.

NexusFileID[43]. Another NexusFile community-created application, this enables users to claim a unique username linked to their NexusFile wal-let address. Most interestingly, a number of other community de-velopers have integrated NexusFileID into their own permaweb applica-tions, including the aforementioned AskWeave, meaning that users can utilise the 'unsiloed' feature of permaweb data (i.e. that the block-weave can act as a single, universal data source for any permaweb application). In this instance, this means that users can have one universal username for all permaweb applications, without having to register for individual user accounts on every individual platform.

8 Future Work

8.1 Succinct Proofs of Access

The primary scalability challenge presented by the current NexusFile archi-tecture is the necessity for pushing all transaction data to all nodes during the writing of the data into the network. While the current architecture of the system allows for approximately 310 million typically-sized pieces of NexusFile data to be stored per year (as of block 223,280), substantial im-provements are possible in this area. The core NexusFile development team plans to o er these improvements to the community by introducing a new optional data access proof mechanism: Succinct Proofs of Access.

Transactions secured by the Succinct Proof of Access mechanism replace the direct embedding of data within the blockweave with storing only Merkle roots of said data. This introduces a number of trade-o s. While it allows for consensus about the entry of data into the blockweave to be acquired without all participating nodes receiving a full copy of the data (if, for example, they do not have excess storage capacity with which to store said data), it also removes the proof of distribution guarantees presented by traditional blockweave data structures. This allows transactions to contain massive quantities of data, but loses the certainty that said data was ever fully seeded to the network.

Succinct Proofs of Access to transaction data can be provided by foreign nodes during block con rmation with a high degree of con dence, by simply then transferring a series of Merkle paths through the Merkle tree generated for the dataset. This has the advantage of allowing vast amounts of data to be added to the network quickly, but does not provide the same data avail-ability guarantees as traditional Proof of Access. Subsequently, Succinct Proofs of Access will be provided as an optional feature for those with large datasets that require long-term storage with high levels of redundancy, but do not require that certainty be achieved about the data ever being fully publicly available.

8.2 Wallet Logs

Further improvements are also planned for the system of Wallet Lists in the NexusFile network. Wallet Lists (a mechanism for maintaining accounts, balances, and avoiding replay attacks) currently represent a high proportion of the data storage overhead in the NexusFile network. Each wallet ever used in the block weave is represented in the Wallet List in each block. It is planned that these wallet lists will be replaced with append-only wallet logs. By appending a new entry to the wallet storage data structure each time a balance is changed, the storage complexity drops from Oblocks wallets to Otxs + blocks. Further, the wallet log mechanism does not materially a ect wallet data look-up times, as the log structure can be e ectively 'compacted' once every given number of blocks. This compacting mechanism keeps the size of the structure manageable relative to the number of wallets in the network. Further, this mechanism also allows for O1 veri cation of new block contents.

8.3 Fast Find

As the NexusFile network incentivises all network participants to store as much of the blockweave as possible, the issue of rapid data location in a de-centralised web is quickly diminished when compared to typical distributed hash table-based approaches. By ensuring that there are large numbers of replications of each piece of data in the network (approximately 750 at the time of writing) the game of nding data turns from ‘ nding a needle in a haystack’ to ’ nding some hay in a barn’, since replications are so abundant in the NexusFile network. Nonetheless, at some point optimisation of data location times may be desirable. We expect that this is likely to take the form of modi ed AIIA agents that reward players not just for providing fast access to data they store locally, but also for re-directing participants to nodes that are able to provide the data.

As in typical AIIA meta-games, each node in the network is sel shly in-centivised to incentivise the expression of this pro-social behaviour in other nodes. The implementation of each independent AIIA agent will de ne ex-actly how participants maintain tables for routing requests of other nodes towards likely holders of the data they are requesting. Nonetheless, it is expected that eventually these agents will closely resemble next-hop routing layers, similar to the structure of the existing internet.

9 Conclusion

In conclusion, we have presented the NexusFile family of protocols: a modular system for achieving permanent replication and recall of knowledge and his-tory, backed by sustainable economic mechanisms. As well as promoting the perpetual storage of information through a novel storage endowment, Ar-weave also promotes the distribution of this information through adaptable and robust o -chain incentive mechanisms.

On top of the primary NexusFile protocol layer, we have also presented an out-line of the ‘permaweb’: a permanent, decentralised, and resilient web. Unlike the traditional web, all content in the permaweb is immutable, timestamped, and cryptographically signed (ensuring strong authorship properties).

The permaweb is currently in its infancy. It has been live for 500 days, and is growing at an approximate rate of 2,500 web applications and pages per day. Further to the presented description of the current protocol implementation, we have also outlined areas of potential future work, as well as adaptive mechanism designs to encour these developments.

References

[1] Mustafa Akgul and Melih Kirlidog. Internet censorship in turkey. Internet Policy Review, 4(2):1{22, 2015.

[2] Eitan Altman, Rachid El-Azouzi, Yezekael Hayel, and Hamidou Tembine. The evolution of transport protocols: An evolutionary game perspective. Computer Networks, 53(10):1751{1759, 2009.

[3] Anderson, Moore, Nagaraja, and Ozment. Incentives and information security. In Nisan et al. [50].

[4] Farhad Anklesaria, Mark McCahill, Paul Lindner, David Johnson, Daniel Torrey, and B Albert. The internet gopher protocol (a distributed document search and retrieval protocol). University of Minesota Microcomputer and Workstation Networks Center, Spring, 1993.

[5] NexusFile.Avoiding consumer lock-ins with the decentralised web. https://hackernoon.com/ avoiding-consumer-lock-in-with-the-decentralised-web-c618f28241ab, 2019.

[6] Fernando Baez and Alfred J MacAdam. A universal history of the destruction of books: From ancient Sumer to modern Iraq. Atlas New York, 2008.

[7] R. Barnes, J. Ho man-Andrews, D. McCarney, and J. Kasten. Automatic certi cate management environment (acme). https: //ietf-wg-acme.github.io/acme/draft-ietf-acme-acme.html, 2019.

[8] Tobias Baumann, Thore Graepel, and John Shawe-Taylor. Adaptive mechanism design: Learning to promote cooperation. arXiv preprint arXiv:1806.04067, 2018.

[9] D. Bayer, S. Haber, and W. S. Stornetta. Improving the e ciency and reliability of digital time-stamping. In R. Capocelli, A. De Santis, and U. Vaccaro, editors, Sequences II, pages 329{334. Springer, 1992.

[10] Barry L Bayus. An analysis of product lifetimes in a technologically dynamic industry. Management Science, 44(6):763{775, 1998.

[11] Jacob D Bekenstein. Universal upper bound on the entropy-to-energy ratio for bounded systems. Physical Review D, 23(2):287, 1981.

[12] J Benet. Ipfs - content addressed, versioned, p2p le system (draft 3). https://ipfs.io/ipfs/ QmV9tSDx9UiPeWExXEeH6aoDvmihvx6jD5eLb4jbTaKGps.

[13] A Berman. Bitmex research nds potential bug in syncing of ethereum parity full node. https://cointelegraph.com/news/ bitmex-research-finds-potential-bug-in-syncing-of-ethereum-parity-full-node, 2019.

[14] Bitcoin genesis block. https://www.blockchain.com/btc/tx/ 4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b, 2009.

[15] Ren Carmona and Francois Delarue. Probabilistic Theory of Mean Field Games with Applications I-II. Springer, 2018.

[16] Anton-Hermann Chroust. Socrates{a source problem. The New Schol-asticism, 19(1):48{72, 1945.

[17] D Clark. Intel rechisels the tablet on moores law. https://blogs.wsj.com/digits/2015/07/16/ intel-rechisels-the-tablet-on-moores-law/, 2015.

[18] Bram Cohen. Incentives build robustness in bittorrent. In Workshop on Economics of Peer-to-Peer systems, volume 6, pages 68{72, 2003.

[19] Matt Corallo. Compact block relay. bip 152, 2017.

[20] C. Decker and R. Wattnehofer. Information propagation in the bitcoin network. In IEEE P2P 2013 Proceedings, 2013.

[21] Robert P Dellavalle, Eric J Hester, Lauren F Heilig, Amanda L Drake, Je W Kuntzman, Marla Graber, and Lisa M Schilling. Going, going, gone: Lost internet references, 2003.

[22] Bianca M Federico. The patent o ce re of 1836. J. Pat. O . Soc'y, 19:804, 1937.

[23] Steve Forbes. Web of deception: Misinformation on the Internet. In-formation Today, Inc., 2002.

[24] Anne Frank and Storm Jameson. Anne Frank's diary. Vallentine, Mitchell, 1958.

[25] Henrik Frystyk, Tim Berners-Lee, and Roy T Fielding. Hypertext trans-fer protocol { http/1.0. RFC 1945, RFC Editor, 1996.

[26] Pawel Garbacki, Dick HJ Epema, and Maarten Van Steen. An amort-ized tit-for-tat protocol for exchanging bandwidth instead of content in p2p networks. In First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007), pages 119{128. IEEE, 2007.

[27] GoldZeus. Weaver. https://github.com/GoldZeus/weaver, 2019.

[28] Olivier Gueant, Jean-Michel Lasry, and Pierre-Louis Lions. Mean eld games and applications. In Paris-Princeton lectures on mathematical nance 2010, pages 205{266. Springer, 2011.

[29] S. Haber and W. S. Stornetta. How to time-stamp a digital document. Journal of Cryptology, 3:99{111, 1991.

[30] Zhu Han, Dusit Niyato, Walid Saad, Tamar Baar, and Are Hjrungnes. Game Theory in Wireless and Communication Networks: Theory, Mod-els, and Applications. Cambridge University Press, 2011.

[31] Adam Hayes. A cost of production model for bitcoin. Available at SSRN 2580904, 2015.

[32] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoins peer-to-peer network. In 24th fUSENIXg Security Symposium (fUSENIXg Security 15), pages 129{144, 2015.

[33] Lambertus Hesselink, Sergei S Orlov, and Matthew C Bashaw. Holo-graphic data storage systems. Proceedings of the IEEE, 92(8):1231{ 1280, 2004.

[34] D Johnston. North's ex-secretary tells of altering memos. https://www.nytimes.com/1989/03/23/us/ north-s-ex-secretary-tells-of-altering-memos.html, 1989.

[35] B Kahle. Fire update: Lost many cameras, 20 boxes. no one hurt. https://blog.archive.org/2013/11/06/ scanning-center-fire-please-help-rebuild/, 2013.

[36] N Kuznetsov. As russian censorship increases, is a decentralized web the answer? https://thenextweb.com/podium/2019/05/30/ as-russian-censorship-increases-is-a-decentralized-web-the-answer/, 2019.

[37] A Lafrance. The internet's dark ages. https://www.theatlantic. com/technology/archive/2015/10/raiders-of-the-lost-web/ 409210/, 2015.

[38] J Langston. Digital collections. https://www.loc.gov/, 2019.

[39] J Langston. With a hello, microsoft and uw demonstrate rst fully automated dna data storage. https://news.microsoft.com/ innovation-stories/hello-data-dna-storage/, 2019.

[40] Sangho Lee and Jong Kim. Early ltering of ephemeral malicious ac-counts on twitter. Computer Communications, 54:48{57, 2014.

[41] Birmingham Public Libraries. Notes on the History of the Birmingham Public Libraries: 1861-1961. Birmingham Public Libraries, 1962.

[42] Ziyao Liu, Nguyen Cong Luong, Wenbo Wang, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. A survey on applications of game theory in blockchain. arXiv preprint arXiv:1902.10865, 2019.

[43] Lyner. NexusFileid. https://NexusFile.net/ fGUdNmXFmflBMGI2f9vD7KzsrAc1s1USQgQLgAVT0W0, 2019.

[44] Lyner. Askweave. https://NexusFile.net/ HhIjOjxgHYXJU5RVjRYfAR017vbZdujbCSIaA8NQ20U, 2019.

[45] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. IACR Cryptology ePrint Archive, 2018:236, 2018.

[46] Microsoft. Photodna. https://www.microsoft.com/en-us/photodna, 2019.

[47] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http: //www.bitcoin.org/bitcoin.pdf, 2008.

[48] The National Archives. Investigation into forged documents discovered amongst authentic public records: Documents purporting to have been created by members of the british government and members of the brit-ish armed services relating to leading nazis gures and axis power gov-ernments. https://discovery.nationalarchives.gov.uk/details/ r/C16525, 2007.

[49] Mehdi Nikkhah, Aman Mangal, Constantine Dovrolis, and Roch Guerin. A statistical exploration of protocol adoption. IEEE/ACM Transactions on Networking, 25(5):2858{2871, 2017.

[50] Nisan, Roughgarden, Tardos, and Vazirani, editors. Algorithmic game theory. Cambridge University Press, 2007.

[51] G Orwell. Nineteen Eighty-Four. Secker & Warburg, 1949.

[52] A Pinar Ozisik, Gavin Andresen, George Bissias, Amir Houmansadr, and Brian Levine. Graphene: A new protocol for block propagation using set reconciliation. In Data Privacy Management, Cryptocurrencies and Blockchain Technology, pages 420{428. Springer, 2017.

[53] Steve Phelps, Peter McBurney, and Simon Parsons. Evolutionary mech-anism design: a review. Autonomous agents and multi-agent systems, 21(2):237{264, 2010.

[54] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in bittorrent. In Proc. of NSDI, volume 7, 2007.

[55] Michael Piatek, Tomas Isdal, Tom Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Building bit-tyrant, a (more) strategic bittorrent client. login, 32(4):8{13, 2007.

[56] J; Rydning J Reinsel, D; Gantz. Data age 2025: The digitization of the world from edge to core. https://www.seagate.com/files/www-content/our-story/trends/files/ idc-seagate-dataage-whitepaper.pdf, 2018.

[57] Jonathan Rose. The holocaust and the book: destruction and preserva-tion. Univ of Massachusetts Press, 2008.

[58] T. Roughgarden. Twenty lectures on algorithmic game theory. Cambridge University Press, 2016.

[59] Muhammad Saad, Je rey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen. Exploring the attack surface of blockchain: A systematic overview. arXiv preprint arXiv:1904.03487, 2019.

[60] M Sa . Whatsapp 'deleting 2m accounts a month' to stop fake news. https://www.theguardian.com/technology/2019/feb/06/ whatsapp-deleting-two-million-accounts-per-month-to-stop-fake-news, 2019.

[61] sergejmueller. Arboard. https://bkxqaor2dlei.NexusFile.net/ pvmiu4SZKQGWAYjrLWzE_ml70u1-v8zIzQ8WaxIYURk, 2019.

[62] A Souppouris. The internet archive is now home to 10 peta-bytes of data. https://www.theverge.com/2012/10/27/3563082/ internet-archive-total-data-2012-ten-petabytes, 2012.

[63] Margaret Sullivan. Were changes to sanders article stealth editing. The New York Times, 2016.

[64] tevador. Randomx. https://github.com/tevador/RandomX, 2019.

[65] Dave Thaler and B Aboba. Rfc 5218: What makes for a successful protocol?, 2008.

[66] S Williams. Weavemail. https://NexusFile.net/ oEhzHOE2o9uZbi6O9cQatzhiHtc2EdFBGCQdqHsK-o4, 2019.

[67] Gavin Wood et al. Ethereum: A secure decentralised generalised trans-action ledger. Ethereum project yellow paper, 151:1{32, 2014.

[68] Xueyang Xu, Z Morley Mao, and J Alex Halderman. Internet censorship in china: Where does the ltering occur? In International Conference on Passive and Active Network Measurement, pages 133{142. Springer, 2011.

[69] Jingyu Zhang, Mindaugas Gecevicius, Martynas Beresna, and Peter G Kazansky. 5d data storage by ultrafast laser nanostructuring in glass. In CLEO: Science and Innovations, pages CTh5D{9. Optical Society of America, 2013.

10 Appendices

10.1 Hard Drive Capacities and MTBF Over Time

Please see the following NexusFile transaction:

https://NexusFile.net/wufZ10dIzwfPFTNKr3uRAyeMRfMdkNx1iG9yjolRbv8

10.2 Anatomies

This section will describe a range of data structures and their contents.

10.2.1 Block Data Structure

Transactions: A list of zero to many transactions (see 10.2.4 Transaction Data Structure).

Block Size: The sum of the number of bytes of the data in each transaction in Transactions.

Timestamp: The decimal representation of the Unix Timestamp when mining the block started.

Reward Address: Up to 32 bytes. If this matches a Wallet Address in the previous block's Wallet List, then this Wallet Address will get the reward.

Previous Block: The Independent Hash of the previous block.

Height: The zero-based sequential number of this block in the Block Weave.

Weave Size: The sum of the Block Size of this block and the Weave Size of the previous block.

Storage Endowment: The number of Winston in the endowment (see 3.2.3).

Hash List: A list of all the Independent Hashes starting from the previous block down to the Genesis Block

Hash List Merkle Root: The SHA-384 hash of the concatenation of the previous block's Hash List Merkle Root and the previous block's

Independent Hash. For the Genesis Block, the value is empty. Basically, this is the root of a completely unbalanced Merkle tree for all Independent Hashes from the Genesis Block up to the previous block.

Wallet List: A list of all wallets ever received any AR, so called Active Wallets, after applying the transactions in this block. The list is ordered by Wallet Address.

{ Wallet Address: The SHA-256 hash of the wallet's public key.

{ Last Transaction: The ID of the last mined transaction created by this wallet, i.e. the last mined transaction where the SHA-256 hash of Owner for the transaction is the Wallet Address.

{ Balance: The current balance of AR for the wallet in Winston. It's adjusted by these rules:

The block contains a transaction where the SHA-256 hash of Owner is matching the Wallet Address. The balance is then reduced by the Reward of the transaction. Negative balance is not allowed.

The block contains one or more transactions where the Tar-get is matching the Wallet Address. The balance is then increased by the sum of the Quantity values of these trans-actions. If this is the rst time this Wallet Address receives any AR, its entry in the Wallet List will be created according to these rules:

The balance will be set to the transaction's Quantity minus 25 AR. This represents the Wallet Creation Fee which is spam lter for active wallets.

The wallet will only be created if the quantity is 25 AR or more.

The transactions are applied in the order they are speci ed in the block.

The Reward Address is matching the Wallet Address. The balance is then increased by the mining reward for this block (see 3.2.3).

Block Data Segment: A SHA-384 hash of the concatenation of the following:

{ Independent Hash of the previous block.

{ Dependent Hash of the previous block.

{ The decimal representation of Timestamp.

{ The decimal representation of Last Retarget.

{ The decimal representation of Height.

{ Concatenation of every value in Hash List.

{ The Serialised Representation of Wallet List.

{ Reward Address.

{ The decimal representation of Storage Endowment.

{ The Serialised Representation of this block's Recall Block (see 2.1 The Recall Block, Proof of Access, and the Blockweave).

{ Concatenation of the Serialised Representation of each transaction in Transactions.

{ Hash List Merkle Root

Dependent Hash: The SHA-384 hash of the concatenation of Block Data Segment and Nonce.

Nonce: A ects the Dependent Hash and must be chosen so that the Dependent Hash ful ls the Di culty. Up to 512 bytes.

Di culty: The number the PoW hash must be greater than. The value is inherited from the previous block except if this is a retarget block where the value may increase or decrease proportionally to the ratio between the target time and the actual time.

Last Retarget: The Timestamp of the last block where Retarget happened. If this is a retarget block, then the value should be this block's Timestamp.

Cumulative Di culty: The expected number of hashes tried by the network, summed up over previous blocks of this fork.

Independent Hash: The Deep Hash (see 10.2.2 Deep Hash) of a list of the following:

{ Nonce.

{ Previous Block.

{ The decimal representation of Timestamp.

{ The decimal representation of Last Retarget.

{ The decimal representation of Di culty.

{ The decimal representation of Cumulative Di culty.

{ The decimal representation of Height.

{ Dependent Hash.

{ Hash List Merkle Root.

{ The concatenation of all transaction IDs of the transactions in Transactions.

{ The Serialised Representation of Wallet List.

{ Reward Address or "unclaimed" if empty.

{ Empty list.

{ The decimal representation of the Storage Endowment.

{ The decimal representation of Weave Size.

{ The decimal representation of Block Size.

Serialised Representation: concatenation of the following:

{ Nonce.

{ Previous Block.

{ The decimal representation of Timestamp.

{ The decimal representation of Last Retarget.

{ The decimal representation of Di culty.

{ The decimal representation of Height.

{ Hash.

{ Indep Hash.

{ The concatenation of the Serialised Representation of the transactions in Transactions sorted by Transaction ID.

{ The concatenation of each element of Hash List.

{ The Serialised Representation of Wallet List.

{ Reward Address.

{ Decimal representation of Weave Size.

10.2.2 Deep Hash

Deep Hash is a hash algorithm which takes a nested list of values as input and produces a 384 bit hash, where a change of any value or the structure will a ect the hash.

```
de ep _h as h ( List ) when is_list ( List ) -> h a s h _ b i n _ o r _ l i s t ( List ) .


%%%          INTERNAL

h a s h _ b i n _ o r _ l i s t ( Bin )  when          is _b in ar y ( Bin )  ->
      Tag = << " blob " , ( i n t e g e r _ t o _ b i n a r y ( by te _s iz e ( Bin ) ) ) /
    binary > > ,
      hash_bin ( < <( hash_bin ( Tag ) ) / binary , ( hash_bin ( Bin ) ) / binary
    > >) ;
```

```
h a s h _ b i n _ o r _ l i s t ( List )  when           is_list ( List )  ->
        Tag = << " list " , ( i n t e g e r _ t o _ b i n a r y ( length ( List ) ) ) /
      binary > > ,
        h as h_ li st ( List ,  hash_bin ( Tag ) ) .

ha sh _l is t ( [ ] ,  Acc )  ->
        Acc ;
ha sh _l is t ( [ Head  |  List ] ,  Acc )  ->
        HashPair = << Acc / binary , ( h a s h _ b i n _ o r _ l i s t ( Head ) ) /
      binary > > ,
        NewAcc  =  hash_bin ( HashPair ) ,
        h as h_ li st ( List ,  NewAcc ) .

hash_bin ( Bin )  when       i s_ bi na ry ( Bin )  ->
        h a s h _ s h a 3 8 4 ( Bin ) .
```

Listing 1: Deep Hash reference implementation in Erlang

10.2.3 Blockshadow Data Structure

A Blockshadow is a slimmed-down version of the full Block where the removed data can be reconstructed from other data. This makes the blockshadow tiny compared to the full block, which makes it fast and cheap to propagate between nodes in the network. See 2.3 Blockshadows.

A blockshadow is the same as the block with these di erences:

Transactions: Instead of the list of transactions, the blockshadow has a list of the transaction IDs. When the blockshadow of a new block is propagated though the network, the receiving node is likely to have the transactions in its mempool, making it possible to reconstruct the full list of transactions from the transaction IDs.

Hash List: The Hash List is trimmed down to only include the rst 50 entries. The full Hash List can be reconstructed by taking the Hash List of the previous full block and pre-pending the previous block's independent hash.

Wallet List: The Wallet List is not included. It can be reconstructed by the Wallet List of the previous full block and the transactions of this block.

10.2.4 Transaction Data Structure

Data: Between 0 and 10,485,760 bytes of arbitrary data.

Owner: The public key of the RSA key-pair signing this transaction.

Quantity: Amount of Winston to send to another wallet.

Target: The Wallet Address of the recipient of the Winston speci ed by quantity. No validation may be performed of the validity. Up to 32 bytes.

Reward: The amount of Winston payed by Owner which will go to the Storage Endowment (see section 3.2.3.

Tags: A list of any number of tags where its Serialised Representation is up to 2048 bytes.

{ Tag: A key-value pair for arbitrary metadata.

Name: The key. Unlimited number of whole bytes of data.

Value: The value. Unlimited number of whole bytes of data.

Serialised Representation: A concatenation of the Name and Value.

{ Serialised Representation: A concatenation of each tag's Serialised Representation.

$T X_{anchor}$: $T X_{owner}$'s last processed transaction ID or the independent hash of one of the last 50 blocks. Empty for the rst transaction.

Signature Data Segment: A concatenation of the following elds in the following order:

{ Owner.

{ Target.

{ Data.

{ The decimal representation of Quantity.

{ The decimal representation of Reward.

{ Last TX.

{ The Serialised Representation of Tags.

Signature: The RSA-SHA256 signature of Signature Data Segment for the RSA key-pair where the public key is Owner.

ID: SHA-256 hash of the Signature.

Serialised Representation: The concatenation of the following:

{ ID.

{ Last TX.

{ Owner.

{ The Serialised Representation of Tags.

{ Target.

{ The decimal representation of Quantity.

{ Data.

{ Signature.

{ The decimal representation of Reward.