electronics

MDPI

Article

UI dApps Meet Decentralized Operating Systems

Rafał Skowroński * and Jerzy Brzeziński

Department of Computer Science, Faculty of Distributed Systems, Poznan University of Technology, Marii Skłodowskiej-Curie 5, 60-965 Poznan, Poland
* Correspondence: rafal.skowronski@put.poznan.pl

Abstract: The advent of Ethereum opened up a pandora box of decentralized possibilities. While allowing for the replicated, decentralized computation of Turing-complete instructions, platforms such as Ethereum do not offer the possibility of direct, interactive, real-time processing of users' inputs that could later affect the decentralized state machine. They cannot directly observe, replicate and authenticate users' actions performed in real-time while presenting the results of these. They lack mechanics that would incentivize full-nodes to provide low-latency-constrained services to users in-between epochs of a decentralized state machine, thus pushing dApps' developers towards hybrid architectures—ones employing centralized servers while not even considering certain applications, due to the aforementioned limitations. In this research paper, we explore our results of an attempt to create a 'decentralized operating system' user experience a reality. We propose an architecture which solves the problems of the responsiveness and finalization of multiple actions performed by users in real-time—without the need for users to pre-authenticate but after having presented a single, unitary consent to commit—through the hereby proposed Deferred Authentication mechanism. To allow for this, we employ an in-house developed #GridScript programming language, used by our decentralized state machine, along with a computer-vision-enabled and AI-aided mobile app (available for both iOS and Android). We introduce the concept of Decentralized Processing Threads (DPTs) and see how these enable fascinating possibilities. In addition, we look into how Access-Control-Lists (ACLs)-enabled, incentivized storage, incentivized Sybil-proof communication, embedded firewall apparatus, integrated off-the-chain payments, and crypto-incentivized off-the-chain storage aid such a system and thus render it as feasible. We highlight various interesting troubles we have encountered, such as state recovery after disconnects of the UI and the replication of its state across both nodes maintaining the network and web browsers. We depict 'off-the-chain' mechanics, which we use to reward for real-time services provided to users by nodes maintaining the network. We tackle crypto-incentivized WebRTC swarms not needing centralized servers for signaling. We look into a user-friendly approach to Non-Fungible Tokens (NFTs). The test-bed is readily available with multiple functional UI dApps already in place. Indeed, the paper presents UI and UX design decisions we have undertaken based on conclusions from statistical research results on a group of 50,341 volunteers over 4 years, which we have used to formulate what we codenamed as the Venice UI/UX design paradigm. We extend upon the notion of Token Pools to allow for the Sybil-proof incentivization of multiple-peers from a single data structure stored on the decentralized state machine.

Keywords: blockchain; decentralized state machines; decentralized applications; dApps

Citation: Skowroński, R.; Brzeziński, J. UI dApps Meet Decentralized Operating Systems. *Electronics* **2022**, *11*, 3004. https://doi.org/10.3390/electronics11193004

Academic Editor: Asma Khatoon

Received: 15 July 2022
Accepted: 13 September 2022
Published: 22 September 2022

1. Introduction

To the best of our knowledge, we have conceived and implemented something that both users and dApps' developers could consider as their personal, multi-purpose decentralized Operating System. Here, users receive the kind of experience they are already familiar with from centralized operating systems, while dApps' developers receive a framework with unprecedented, multi-tasking, communication, storage, presentation, and user-interaction capabilities. All decentralized. All incentivized.

1.1. Layout of this Research Paper

The layout of this research paper is as follows. First, in this section, we give some introductory overview, while discussing the 5xA Design Paradigm which we came up with after years of development of both the decentralized operating system and the UI dApps running atop of it. Section 2 provides concise definitions used throughout the rest of the paper. In Section 3, we introduce the reader to the concept of a decentralized operating system. We explore the overall rationale and explain how the very concept contributes to and builds upon the current state of decentralized systems. We explore the consensus algorithm of the available test-bed prototype; we explore the architecture, dependency relations, and use case scenarios by looking at a detailed UML diagram. In Section 4, we define our two-tier research problem. In Section 5, we look into some of the previous works. In Section 6, we highlight our contribution to the current state of the art. In Section 7, we briefly explore #GridScript, the programming language used within our decentralized state machine. Section 8 discusses the concept and mechanics of a Decentralized Bootloader. With Section 9, we proceed to the hereby introduced concepts of Deferred Authentication and State-Full Sessions. Going further, Section 10 introduces us to the fascinating concept of Decentralized Processing Threads, which became an indispensable element and paramount contribution to our test-bed environment. Specifically, we look at how actions performed by users in real time, the results of which are immediately visible—can then be replicated across the decentralized state machine at the user's discretion. Section 11 tackles the topic of incentivized data exchange and State-Channels facilitated through the unique hereby-introduced concept of Multi-Dimensional Token Pools, which allow for efficient remuneration of agents for arbitrary minuscule actions, causing minimal overhead on the decentralized state machine and requiring a minimal amount of computation when used. The only data to be remembered by a client using Multi-Dimensional Token Pools is a single seed-hash, thus yet another fascinating concept. In Section 12, we discuss our implementation of Decentralized Storage. We provide a detailed Thread Model and an algorithm depicting our Proof-of-Storage construct, with a discussion included. In Section 14 we present what we have coined as the Venice UI/UX design paradigm. It is a fruit of long years of research and development while striving to provide an appropriate Windows/macOS X-like operating system experience for decentralized applications of any kind. Further, in Section 15, we go back to discussing how the hereby presented blueprint extends upon what is available and provides a clear numerical results section to further back up possibilities provided by the hereby proposed design and development. We look into the achievable Transaction Per Second (TPS) statistics of our current test-bed environment; in particular, we analyze the performance of State-Channels facilitated through Multi-Dimensional Token Pools in the context of incentivized data exchange. Thus, on these grounds, this constitutes a continuation of our previous research [1] as we provide detailed statistical analysis of the previously introduced communication protocols and see how Multi-Dimensional Token Pools allow for what we deem as unprecedented performance in Sybil-proof incentivization of data exchange; not only within the bounds of a Decentralized State Machine but of external arbitrary data traffic as well. We explain how this statistical analysis can be seen as helpful in visualizing the performance of rewarding for arbitrary services provided in real time to users, as we focus on the case of a single-hop incentivized communication link. In addition, this research paper comes with Supplementary Material, including a spreadsheet-based simulator, where the reader can validate all of the results and probably arrive at some interesting conclusions.

1.2. Prelude to Decentralized Operating Systems

Indeed, there are reasons why non-hackers tend to prefer applications with a user interface over command-line terminals. Yet still, as of today, even the latter remains wishful thinking when it comes to current decentralized state machines. Solidity, the language backing Ethereum [2], was not designed to allow for real-time, ad hoc, on-demand, code formulation, interpretation, and execution. Still, it was designed to offer significantly

more than Bitcoin [3], which is the execution of arbitrary Turing-complete instruction sets, comprising what we know today as Decentralized Applications, or dApps for brevity. To put things into a perspective, when using a typical operating system, as the user sits at her home computer, she can execute instructions in a terminal, and these may take an immediate, tangible effect—as far as the use experience is concerned. We want the same for dApps as well. Furthermore, we demand the very that for UI dApps in particular. It is time we bridge the gap between users' needs and what nodes maintaining a decentralized state machine could directly provide to their users. Under the presumption that everything is to be incentivized, i.e., that agents participating in the system are assumed as rationale, we look into how Stateless State-Channels [1]—which we extend upon in this paper—allow for incentivization of most minuscule tasks when such are requested from nodes maintaining the decentralized state machine, rendering the latter significantly more functional.

The decentralized community needs software components—software frameworks that could provide a uniform, easily accessible functionality set for typical features required by Decentralized Graphical Applications (UI dApps), including otherwise ubiquitous mechanics such as packages' deliveries—were it for the realm of centralized environments. Indeed, these are concepts that have been available to users of centralized operating systems for over a decade.

The reader might wonder. Why minimize reliance on a hybrid approach—of mixing centralized and decentralized architectures? First, because we can, and second, because we put forward an assumption that the more, we push users towards external, centralized providers, the more we push them away from decentralization. We presume this to lower the overall security, anonymity, and privacy guarantees, which likely might have attracted users and/or developers towards decentralized applications (dApps) in the first place.

1.3. The 5xA Design Paradigm

We propose that any centralized service and/or sub-service could be implemented as decentralized, as long as the following 5xA properties and requirements are satisfied:

Accessibility—The service remains accessible. Otherwise, agents would not be able to interact with it in the first place. This property implies ease of use and promotes a pleasurable user experience.

Authentication—In order to allow for authorization, authentication is needed first. This includes authentication as 'any' user, should node(s) choose to provide free services during a grace period. Methods might range from IP address authentication to explicit strong signatures and certificates.

Authorization—Agents need to be authorized to perform activities. This includes a trivial situation in which everyone is allowed in the case of publicly available services.

Accountability—This implies verifiability. Parties of interest should be able to verify and benefit from a service delivery. Ideally, individual intermittent stages should be accountable for and verifiable as well, thus securing those who deliver as they can now be rewarded for fulfillment of smaller, identifiable sub-tasks while benefiting users, as they, on the other hand, could avoid paying up-front. Further research in the field of verifiable computations (e.g., using Zero-Knowledge Proofs [4]) could be of immense contribution here.

Actuation—Everyone needs to be incentivized. That accounts for dApps and service providers offering promotional grace periods for their services, which, thanks to authorization and accountability, could be provided in a limited, controlled way.

Notice [Figure 1] how users' and service operators' needs are located at opposite spectrums. Still, both properties, including those in between, would need to be satisfied for the system to thrive. Without accessibility, accountability, and verifiability, actuation would not be possible; thus, the two dipoles need to stay connected through their co-dependants at all times.

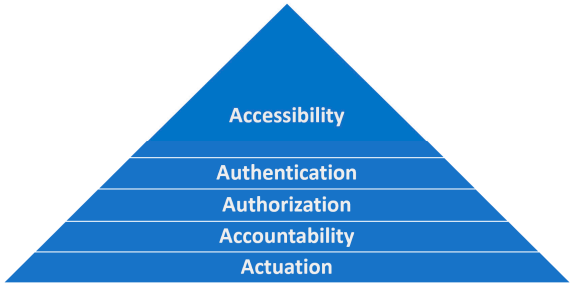

Figure 1. The concept of 5xA visualized.

2. Definitions

Log On–the moment services of the system become available to the user after being requested and possibly after the user was authenticated.

Decentralized State Machine (DSM)—as defined in [1], with the most important property that the environment is open-access, and anyone could aspire to become a leader of each round/epoch of the machine.

Core Process—a program implemented in a language operated by nodes comprising the DSM running either at full redundancy or in an IVR at one of the nodes. Notice that it may process interactive input when running in the Sandbox Mode of an IVR, otherwise not.

Client Process—a process running not directly on nodes maintaining the Decentralized State Machine. It may be running within a web browser while being delivered from and communicating with a Core Process.

UI dApp—a client process, owning UI components (e.g., a Window), communicating with the DSM, i.e., possibly with one of the Core Processes operated by the Decentralized Operating System.

Liveness—we define liveness as the property of a DSM, allowing it to react to users' inputs and or instructions provided to it in real time.

State-Full Sessions—the user session has a state that can be modified and then committed to the DSM asynchronously on demand.

System Trie—the state of a DSM is modeled through State Domains [5] implemented as nodes of a Merkle-Patricia Trie [6]. Thus, modifications of sub-leaves propagate to the root node of the Trie.

Decentralized Terminal Interface (DTI)—a virtual terminal interface, spawned by any node comprising the system, over a communication channel, e.g., SSH or a web browser. The terminal is said to be decentralized, as actions taking place within may affect the entire DSM if a user decides to '*commit*'. In the case of a UI, JavaScript code and other assets may be delivered from multiple nodes *even during a single session*. A single thread can be shared by multiple UI dApps. A thread can be read-only, write-only, or both (in terms of its ability to affect the DSM), and associated flags indicate its functionalities.

Global VM Realm—this represents the state of a DSM, as seen by the majority of nodes maintaining it. It can be altered only by code running in the *Commitment Mode of an IVR*. Thus, code capable of affecting this realm is always executed by the majority of nodes maintaining the DSM.

Private VM Realm—represents a copy of the System Trie, which was spawned for the purpose of serving a user's session, possible through the *Sandbox Mode of an IVR*.

Local Authentication—authentication performed to a single node. It has no consequences from the viewpoint of the entire DSM, as that would imply putting global trust in the latter (e.g., for the purpose of serving a user-chosen desktop wallpaper, a user's current account balance, to bring a user to their home directory in a File Browser UI dApp as they decide to launch, etc.).

Isolated VM Realm (IVR)—a software construct providing an interface through UADL for manipulation and access to either a Global or a Private VM Realm. When in sandbox mode, its main purpose is to operate on a copy of the System Trie to be made

available to instructions executed in a DPT. IVR running in *Commitment Mode* can modify the Global VM Realm and its associated System Trie. An IVR is composed of a DPT together with a Private VM Realm which can be *committed*. In addition, it is the responsibility of an IVR to decide which instructions are to be formed into a source-code package as they are executed.

Commitment Mode of an IVR—executes instructions from the associated DPT in commitment mode. Typically, that would imply the execution of byte code from a *Committable Bytecode Package*.

Deferred Authentication—authentication to the DSM takes place *after* instructions were executed. The authentication allows to commit a priori defined, i.e., actually performed (in terms of perceived results, i.e., in Private VM Realm) actions. The mechanism is facilitated through the compartmentation of the DSM's sandboxed states (Private VM Realms) that can be *'committed'* later on through bytecode packages, on a user's consent. That happens due to, in layman's terms, the 'smart recording of instructions' executed in Sandboxed IVRs, thanks *to State-Full Sessions*. In GRIDNET OS, the actual authentication (signature) is provided by the mobile app, which is onion routed to a node serving the System IVR.

System IVR—the root IVR, running in sandbox mode. There is only one System IVR for any given user session. The commitment of a System IVR implies the commitment of all the associated sub-IVRs, which implies the commitment of all the associated sub-DPTs. In layman's terms, multiple UI dApps during a single session can affect the DSM, as each can own an IVR with an associated DPT, or multiple of these.

Commitment of a DPT—when committable code inherent to a DPT makes its way to a Committable Code Package, it is compiled, propagated across the network, and expected to be executed at full redundancy, i.e., by the majority of the nodes.

Sandbox Mode of an IVR—the mode of an IVR in which code executes on an individual node but does not affect the rest of the system (storage at other nodes).

Committable Code—instructions from a DPT which are to affect the state of a DSM, implying instructions that are to execute at full redundancy on the DSM should a user choose to commit a DPT.

(Committable) Source Code Package ((C)SCP)—source code from possibly multiple DPTs associated with a single user session.

Committable Bytecode Package (CBCP)—the above, with code from each thread successfully compiled. It is authenticated, i.e., signed in the name of the user, ready to be executed by a DSM.

Execution of a CBCP—it is assumed that any CBCP that ever affected the DSM must have been executed by the majority of nodes comprising the DSM at the time the CBCP affected its persistent storage.

Global Authentication—this implies authenticating to the entire DSM. This can be accomplished only through an authenticated CBCP, executed at full redundancy. An entity that the authenticated globally may perform actions on the System Trie in accordance with the associated Access Control Lists/Entries of elements that are to be affected.

QR Intent—a QR code representing a description of a processing task to be carried out by the mobile app. It may contain information required for the proper encryption and routing of the response, possibly across multiple hops.

JavaScript VM Context—part of the system running directly in the user's web browser. It makes all the user-mode APIs available to UI dApps through its objects and subroutines.

Selective Non-Malleability—while the sequence of data blocks building upon the current state of a DSM is required to be immutable as time approaches infinity, it does not hold true for higher-level representations of states inferred from data within these. I.e., we may implement a mutable file system on top of an immutable yet extendable sequence of data blocks.

Access Control List (ACL)—an ordered sequence of Access Control Entries associated with an object (file, directories, etc.) stored within a State Domain.

Access Control Entry (ACE)—a data entry associating an agent with a set of privileges (e.g., Read, Write, Execute, Ownership).

User Session—period of communication between a user and one of the nodes maintaining the DSM, during which an IVR is made available to the user, possibly with a Shell and/or a UI, with the user being capable of affecting the DSM.

State domain—as defined in [1].

Client Code—code that has been deployed by a user of the system and does not comprise part of the operating system's code-base and thus is covered by authorization checks.

Kernel Code—code executing at the highest trust level, possibly not constrained by authorization checks, typically implying part of the operating system.

User-Mode values functions—values and API functions that can be freely accessed and modified by client code comprising the Core Processes or Client Processes.

Kernel Mode Values and Functions—entities that cannot be (directly) accessed and modified by code running in user mode. Typically, Kernel Mode functions are accessed through user-mode functions that do all the authorization checks.

Execution Context of a DPT—the set of modifiable kernel-mode and user-mode properties associated with an IVR, which client processes may rely on (e.g., active State domain or directory).

Shell—an instruction interpreter. Running in the context of an IVR, it is made available to the user locally (node's operator) or through a network connection. All instructions are processed through the associated DPT.

Source code—an ordered sequence of instructions that can be executed by the instruction's interpreter.

State-Full Session—we say a session is state-full if it can enable the interactive processing of a user's requests, while preserving the results of previous computations performed during that session, with the ability of having all the results committed to the DSM on the user's consent.

User Actions' Description Language—language suitable for the representation of user actions/requests performed both in the Shell and the UI. It needs to be compiled to an efficient representation (bytecode) and executed. In our sample implementation, #GridScript is such a language.

Code Linker—fetches source code from multiple DPTs, ones marked as *committable*, and produces a single *Committable Source Code Package* (CSCP). Assuming the input source code provided to each committable DPT was correct, it is its responsibility to assure the proper formulation of the resulting Code Package's source code.

Code Compiler—an abstract software component that transforms a Source Code Package into a single Committable Bytecode Package that is to be executed by an IVR in commitment mode by the DSM.

Decentralized Processing Thread (DPT)—an entity responsible for code execution. Characterized by a tuple of: (1) a sequence of instructions to be executed in the context of an IVR (both committable and non-committable) formulated in the language nodes operation, and (2) the associated IVR. DPT can execute in the sandbox mode at particular node(s) or in the commitment mode. This concept is explained in more detail in [Section 10].

Commitment Mode of a DPT—code executed by the System Realm. A DPT may have sub-threads, which usually is the case for the System-Thread and child-threads. When a DPT is marked as committable, the Linker may fetch code from it at any moment.

System DPT—thread associated with the System IVR.

State of a DSM—described as an aggregate state of all of its associated sub-states, implemented through a Merkle Patricia Trie. In #GridScipt, the current state is identified by a single 32-byte byte vector called the PERSPECTIVE. In the case of GRIDNET OS, that means the hash-value of the root node of the Merkle Patricia Trie contains all the sub-states (leaf nodes). It can be checked from the command line by invoking the 'PERSPECTIVE' command.

Non-committable Code—code that a DPT executes but which does not make its way into a committable source code package. Thus, such a code may only run in the Sandbox Mode of IVR/DPT.

Ephemeral Services—possibly paid-for services, provided by nodes, that do not directly affect the DSM.ex. WebRTC data exchange swarms, data-proxy services, etc.

3. The Concept of a Decentralized Operating System

The nomenclature of a Decentralized Operating System is not new. It stems back to 1987 and the work of F. Bairadi et al. [4], who defined Decentralized Operating Systems as: "consisting of several instances of the same set of processes: each instance is allocated to a distinct node, and every decision about the system behavior is taken through cooperation among several (or even all) instances since none of them has authority over the others." Now, that definition perfectly fits our conceptions indeed. Recently, the term began to re-flourish within the scientific community, due to the need to better structure, visualize, and accommodate the concept of decentralized applications and the relationships between them and their surrounding environments.

3.1. Rationale

Today, decentralization and blockchain technology are the new Big Thing, right next to Artificial Intelligence (AI). In order to improve the proliferation of what is available from the scientific and engineering perspectives, we need to make the technology available to users in the way that they need and enjoy. A rational approach might be to take a look at what users of 'centralized services' already keep using and then decentralize it all. Our team managed to implement what could be considered a decentralized operating system from scratch, since current technologies, while being very usable, having in mind Ethereum [2] in particular, did not provide the liveness, Deferred Authentication, and State-Full Session [Section 9] properties required and already provided by almost each and every modern operating system. Indeed, we lay out an assumption that users' needs could be visualized by looking at centralized operating systems and the applications and services that these provide. Centralized solutions have evolved for decades, often backed by monumental development teams. Service providers no more need to rely on 'selling' users' privacy-related data to keep their business afloat. The advent of decentralized state machines changed it all.

Decentralization is not just a fancy hype word. Users can benefit from multiple inherent concepts such as the concept of a 'cryptocurrency' together with fascinating properties enabled through the system-intrinsic properties of open, decentralized state machines, of those with the property of non-malleability as time approaches infinity. Many would rather remain anonymous than share their personal data, or even metadata, with third parties, while still having a securely authenticated, authorized, and customized user experience from the moment they 'log on' assured.

Now, to put things into perspective, let us say the user needs to create a file and have it stored. They want to employ all the benefits of immutability and accountability decentralized state machines are said to offer. They want to modify the file, to delete it, or to be able to assign ownership privileges and any other access-control permissions that they would associate with ease on a 'centralized' laptop of theirs through a single invocation of the 'setfacl' command on Linux, or by clicking or tapping their way through the UI just as they would on Windows or macOS X. They want it all, just decentralized; both malleability and non-malleability. Someone has to deliver.

Have you ever looked into how many steps it takes to deploy an NFT [7] onto the Ethereum [2] state machine? The question is, does that level of complication stem from necessity or rather from an inappropriate system design? Or maybe the system at hand simply was not conceived for such things to begin with? Our results indicate that the latter just might be the case, which is not surprising, as Ethereum was conceived long before NFTs. Now, we look at NFTs as files on decentralized storage, with ownership, access-

control lists, and possibly other metadata attached. We employ the property of selective non-malleability. Consensus-driven metadata representing legal rights associated with files together with ownership privileges affect particular objects' malleability properties. In our DSM, the property of non-malleability is disabled by default for user-created data files and can be acquired by invoking 'setfacl' by the file owner.

Now, how about developers willing to create dApps allowing for over-the-network collaboration or social interactions? That requires data exchange. Since we assume agents are rational, they need to be incentivized. In [1], the authors proposed the very first data exchange protocol provably Sybil-proof in any kind of computer network (i.e., modeled as a random graph) by employing the concept of a crypto-currency. We will be extending upon these conceptions.

3.2. Consensus

Our implementation employs a modified version of the Bitcoin-NG [8] consensus protocol. The algorithm involves two types of blocks: key blocks, ones containing proof-of-work, and data blocks containing the actual, authenticated code packages, possibly affecting the DSM once executed. Block rewards are shared between the leaders of consecutive rounds. A simplified consensus algorithm using the definitions, semantics, and protocols described in [1] follows:

Threat Model: The heaviest in terms of cumulative proof-of-work, sequence of data, and key blocks is preferred. It should be unfeasible to replace blocks in an otherwise valid sequence. Only the current round leader $\mathcal{L}_i$ can propose data blocks. The agent producing the key block resulting in the heaviest in terms of the cumulative PoW chain becomes the new leader. Attackers try to disrupt the thus defined as legitimate linked (sub-)sequence(s) of blocks. The computational power needed to compute PoW is assumed to entail cost.

Epoch*$_1$*: –leader ellection

Preconditions: Λ_1 willing to become a round leader prepares a key pair $\mathbb{A}_1^{\mathrm{SK}}/\mathbb{A}_1^{\mathrm{PK}}$, B_i^K—block containing PoW (Ω), B_i^D—block containing CBCPs, B_L^K, B_L^D—the number of key and data blocks, respectively, that ever affected the DSM. B_x^D—data block at position x.

1. Λ_1 prepares B_i^K containing Proof-of-Work (Ω).
2. Data Block is propagated using incentive-compatible data exchange protocol depicted in Section 6.6.1 of [1].
3. Each $\mathrm{N_i}$ decides if $\sum_{i=0}^{B_L^K} \Omega \sim B_i^K$ is max. for any known linked sequence of key blocks. If not, but with B_i^K otherwise valid, (**3a**) it is stored on $\mathrm{N_i}$ but it does not extend the current chain. If valid and found to produce (**3b**) the heaviest chain, B_i^K is appended and Λ_1 becomes the current leader $\mathcal{L}_i$ If, there are any B_x^D where $x \geq i \sim B_i^K$ all these data blocks are discarded. Notice that it is in the intention of $\mathcal{L}_i$ to honor and extend upon data blocks by $\mathcal{L}_{i-1}$ as they receive shares from fees associated with these. $\mathcal{L}_i$ gains the right to produce data blocks, authenticated through $\mathbb{A}_1^{\mathrm{SK}}$.

Epoch*$_2$*–production of data-blocks

4. $\mathcal{L}_i$ produces B_{i+1}^D bound to $\mathbb{A}_1^{\mathrm{PK}}$, containing CBCPs it managed to collect. $\mathcal{L}_i$ prefers CBCPs yielding highest rewards.
5. B_{i+1}^D gets propagated through protocol depicted in Section 6.6.1 of [1].
6. Each node verifies B_{i+1}^D (sees if Λ_1 is the current leader, it does so if the most recent key block was bound to his $\mathbb{A}^{\mathrm{PK}}$ etc.) if it is, the block is appended, otherwise it is discarded. If the block was found to be produced by Λ_1, but is otherwise invalid, a proof-of-fraud is issued and broadcast throughout the network for processing and Λ_1 is penalized.

Incentivization: Rewards from each data block and key block are shared between $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$.

3.3. The Architecture

Below we outline some of the design relationships between the system's inertias and external agents interacting with the system through a simplified hybrid UML use-case diagram [Figure 2] below.

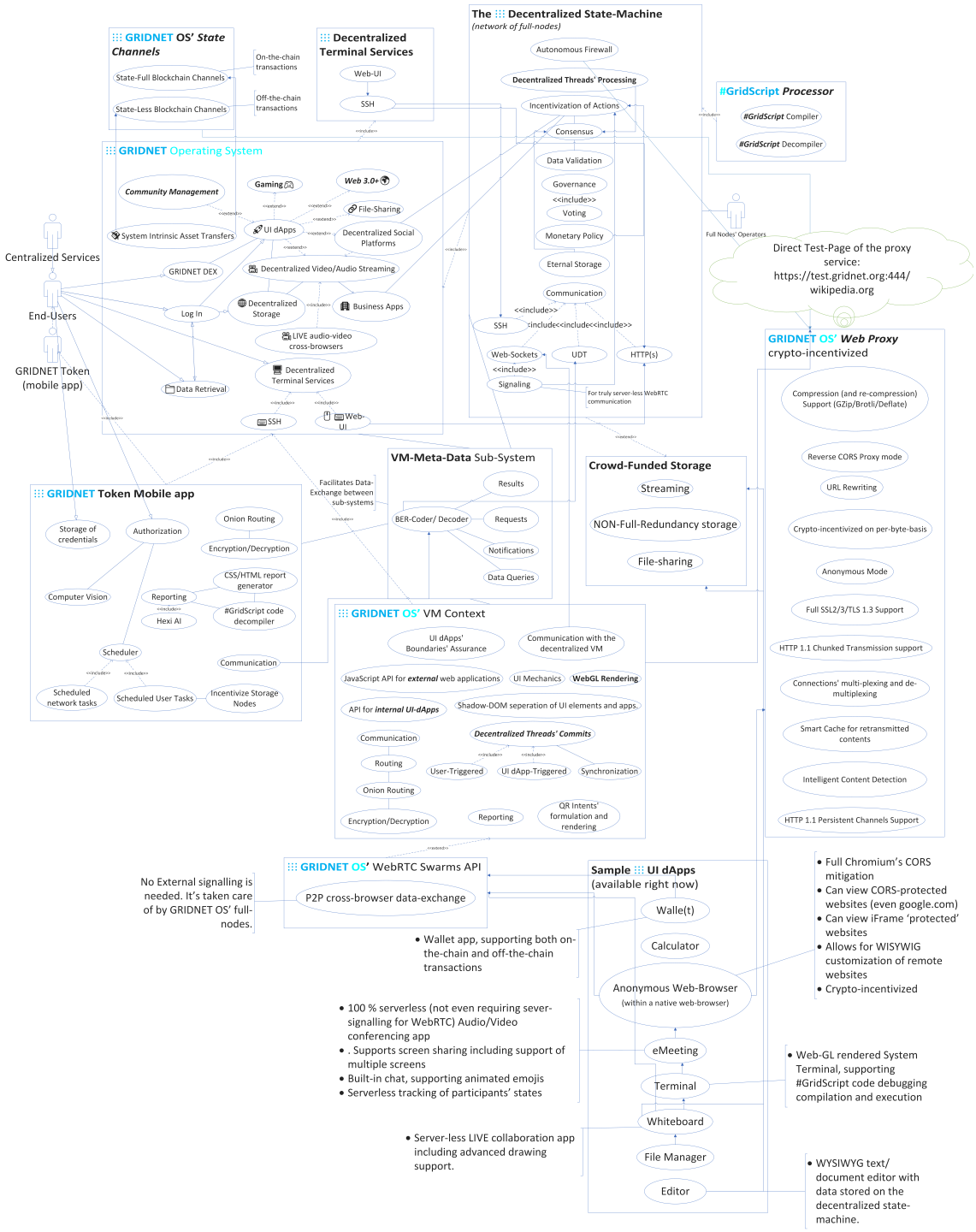

Figure 2. Simplified version of a hybrid UML use-case diagram.

UI dApps always interact with the System through a user-mode function of JavaScript VM Context, which provides client-side APIs and associates the invocation of each instruction with a particular process. dApps (core processes) without a UI comprise #GridScript instructions only. The execution of each instruction entails costs.

3.4. Relationship between the Decentralized Web-UI, UI dApps, IVRs and DPTs

In our implementation, the situation is similar to how Windows 3.11 and Windows 95 used to interact with the underlying DOS operating system. Everything the user does that involves a DSM (data reads/writes), under the hood, is translated into their User Actions'

Description Language (#GridScript) representation. Thus, UADL is the main work-horse of the entire system when it comes to executing the user's actions that involve the DSM. UADL allows to replicate these throughout the network. Here, the replication of actions means the replication of their actual result. UADL does not store mouse movements, say, within a File Manager UI dApp; instead, it contains instructions carrying out write operations along with the path and contents of a file that the user created. Logic implemented within a particular UI dApp, also within the associated IVR, aided by Linker, decide which instructions are to make their way to the resulting Source Code Package [Figure 3]

Figure 3. From user actions to byte code that can be executed by the DSM.

As seemingly we have already well begun discussing solutions, let us first put forward a formal definition of the problem at hand, shall we?

4. Problem Formulation

Note: For the comprehensiveness of presentation, we allow ourselves to tackle both architectural/algorithmic problematics as well as incentivization dilemmas, as the herein provided solution to the latter is used not only for the incentivization of data exchange but aids multiple parts of the system whenever nodes are to be incentivized for minuscule actions during user sessions. Still, we introduce these in the context of communication for on-sight compatibility with protocols and Proof of Sybil-Proofness found in [1].

Architecture/Functionality [Problem 1]

We strive for a decentralized software and algorithmic architecture which would allow for the execution of user-specified computational tasks. It should be possible to formulate such tasks remotely, i.e., either through an emulated terminal interface made available by nodes operating the DSM (i.e., over SSH) or through a user interface. The software and algorithmic apparatus should allow for the existence of decentralized applications equipped with a user interface. No assets of either the system itself (including code, graphics, etc.) or of the dApps should need to be delivered from outside of the (decentralized) system. While initializing the UI, the solution should enable for data deliveries from multiple places, if available. The computational tasks, carried out by dApps, may require affecting the storage of the decentralized state machine, which implies processing by the decentralized state machine at full redundancy. For other tasks, processing by select nodes may be sufficient. The system shall exhibit not a single point of trust. The system shall support the ad hoc, on-demand formulation of tasks/instructions and the execution of these by the decentralized state machine on the user's consent with a minimal number of authentications. For tasks affecting the decentralized state machine, the results should be computed and presented to the user in real-time before the user decides to commit the results onto the DSM and thus making them persistent. The system shall support interactive, real-time processing by nodes comprising the system, with the possibility to commit everything to the DSM on the user's consent. Every aspect of the system shall remain decentralized and incentivized. That holds true for data storage and data exchange. We strive for the solution to allow for user-perceived multi-tasking and multi-threading, with code execution spread across multiple nodes, when possible and desirable. Despite imminent parallelism, the system shall thwart concurrency issues, especially when multiple UI dApps are involved within the same user session. Still, once the user decides to commit computational results, the number of required confirmations and/or authorizations should be minimal and tend to 1. Web browsers are not trusted.

Communication [Problem 2]

The problem is, how does one allow a single peer to establish multiple, efficiently incentivized, in a Sybil-proof manner, simultaneous data streams with multiple peers, while assuring ease-of-use, for networks of arbitrary topology?

Definition of efficiency: utilization of decentralized storage is minimal, both in terms of used storage space and necessary write operations.

Definition of ease-of-use: the number of interactions required from the user should be minimal (registering data on the chain, granting authorizations, etc.).

5. Previous Works

So far, there have been a couple of attempts to market blockchain-based concepts as operating systems. 'Over ledger Operating System' [9]—as of now, the project does not provide any publicly accessible environment for UI dApps or an operating system-like management interface, let alone Linux-like decentralized terminal services. The authors claim to allow for bridging permission and permissionless DLTs, multichain and an interoperable meta identity, and zero-knowledge proofs. They can also support multi-DLT applications. One quickly gets to understand that their platform is much closer to an Ethereum-like system, possibly integrating multiple such systems under the hood, but it does not bridge the decentralized state machine $\leftrightarrow$ services $\leftrightarrow$ user gap we envision a modern Web 3.0 Operating System to fulfill.

Another contestant is 'EOS.IO' [10], and as the authors state: "Ethereum positions itself as a supercomputer while EOS positions itself as its operating system." That was back in 2017. One may judge the results by visiting their website. Unfortunately, it all boils down to deploying smart contracts and going through difficult steps of assuring integration, say, with UI interface, supposedly deployed outside of their environment. There also was a project called 'Liberty OS '; sadly, it may have been described as a Linux distribution with an 'altcoin' attached and unfortunately ended up being a scam, as the authors of the project ceased releasing new information after their fundraising ended.

In terms of incentivized communication, recently in [1], the authors proposed the first communication protocol provably Sybil-proof in computer networks of arbitrary topology. We now extend upon the concept of Token Pools as described in [1] to support multiple simultaneous data streams. Previous research did not tackle the case of multiple simultaneous streams being handled and incentivized by a single peer. In fact, it would be problematic for the previously proposed off-the-chain reward mechanics, as Transmission Tokens [1] from a single Token could not be produced for multiple peers until cashed out on the chain.

6. Contribution

We provide a working solution to the previously formulated multi-stage problem. In order to allow for ad hoc, interactive tasks' formulation, at the very foundation of everything we introduce the concept of Decentralized Processing Threads (DPTs) [Section 10]. DPTs allow for both multi-tasking and multi-threading in a decentralized operating system. When needed, the results of their computations can be committed across the entire DSM. DPTs can perform computations in real time and have state and storage attached through Isolated Virtual Machine Realms (IVRs). We look into the concept of Deferred Authentication [Section 9] and see how our implementation employs both of these for use in Decentralized Terminal Services—here, supporting both UI—accessible through a web browser from every node and Terminal—available both from UI and directly over SSH. We see how the state of an IVR/DPT can be copied across nodes when and as needed. For incentivized data exchange, but also whenever the off-the-chain rewarding of nodes is needed, we build upon [1], where the authors provided the very first Sybil-proof and incentive-compatible data exchange protocol suitable for networks of any topology, by enabling rewarding more than one agent simultaneously through a single state channel. For this, we extend upon the concept of Token Pools introduced in [1] to support multiple simultaneous beneficiaries and introduce Multi-Dimensional Token Pools.

7. #GridScript

#GridScript is a programming language derived from Forth [11]. It remains backwards compatible with it for the most of it. It was extended to better accommodate its usage as a Shell from within an Emulated Terminal Interface. for instance, with the support of in-line parameters for code words. The most notable additions include the built-in support of decentralized state machine operations, cryptographic constructs, the support of decentralized storage and Decentralized Processing Threads, and also the support of Kernel-Mode and user-mode instructions and metadata automatically associated with stack-entries for easy debugging. There are many tutorials available at https://mag.gridnet.org (accessed on 12 September 2022). For the purposes of this article, we would be delighted to highlight a couple of useful instructions:

BT (Begin Thread)—transitions an IVR into package-compilation mode. In over-simplified layman's terms, instructions are recorded and then make their way into a Code Package on 'commit'.

Commit (CT)—by default, causes Linker and Compiler to formulate a Committable Code Package from the thread the command is invoked from. If executed from the System Thread, it causes the compilation of all of the sub-threads. When invoked, the Communication, Linkage, Compilation, Authentication (including mobile app), and Networking sub-systems would be used to sign the resulting Byte Code Package and propagate it throughout the network, to be executed by the DSM.

VT (View Thread)—shows currently formulated committable source code associated with the thread the command is invoked from. A terminal-only command.

Send [Source Destination]—sample command accepting inline parameters, issuing a cryptocurrency transfer.

LogMeIn—sample command which can execute only in sandbox mode. It employs QR codes and onion routing to perform Local Authentication.

Perspective—prints the perspective of the System IVR associated with the user's session.

Chown, Setfacl, Getfacl—Linux-compatible access right and ownership look-up and management utilities.

The language allows for dynamic memory allocation, with dynamic boundaries' checking, which on these grounds is similar to Java. The allocation of each memory unit is charged with the system-intrinsic cryptocurrency. In sandbox mode, the client is allocated a grace amount of resources. Execution in commit mode (full redundancy) always entails a cost. #GridScript boards the in-terminal code compilation, formulation, and debugging constructs. On these grounds, it is interesting to note that as instructions are executed, the code-execution engine associates metadata with data put onto the stack, depicting its origin and making an attempt to include additional descriptions. One may always type 'S' to see the data stack along with the associated metadata.

As a side note, to the best of our knowledge, #GridScript is the only language operated by a decentralized state machine allowing for arbitrary precision mathematical double-floating point operations. Type conversions are performed on-the-fly; thus, on these grounds, it may be considered as a loosely typed language even though it is data-type aware. Mathematical operations employ a post-fix notation; feel free to take it for a spin at any time by accessing the Terminal Services of any node.

8. Decentralized Web Boot-Loader

Operating Systems usually come with a bootloader. The purpose of a bootloader in a Decentralized Operating System shall be to prepare any kind of resources needed to establish a user session. Thus, in the case of an SSH-based session, that would mean the preparation of a Shell. In the case of a UI session, that would involve (1) the preparation of resources on the accessed node(s)' side—the spawning of the command's processing engine, etc. (2) the delivery of a JavaScript bootstrap payload code package (the JavaScript VM Context) to a web browser, which in turn would attempt to fetch additional resources, possibly from multiple nodes, and take care of further initializations. The main rationale

here is to allow for an uninterrupted user experience once the control over the UI is handed over to the user since all the major components are to be pre-fetched. Below in [Figure 4] are actions performed by the decentralized Bootloader, while bootstrapping is what we call a decentralized user interface. Here, it is worthwhile to notice that assets contributing to a single user session indeed might be coming from multiple peers simultaneously.

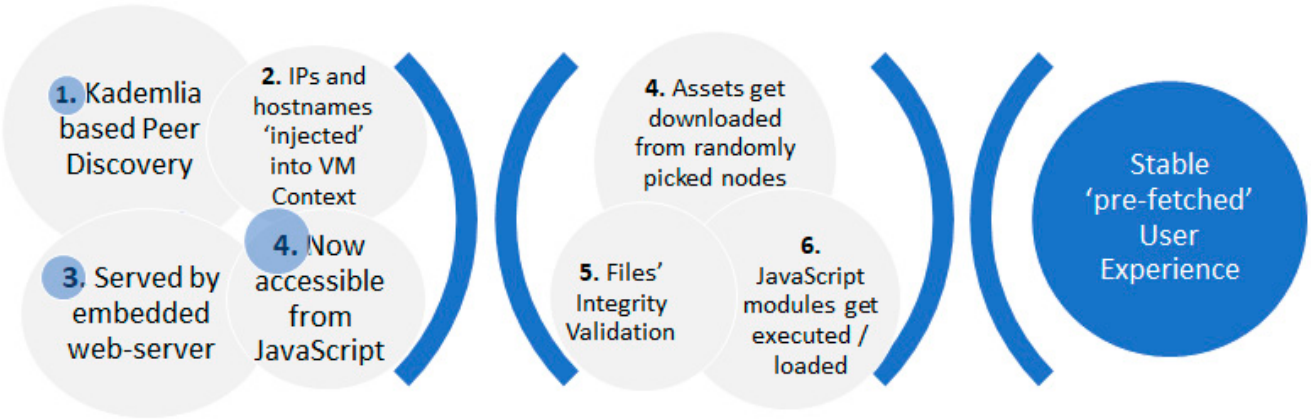

Figure 4. Steps to be taken by the decentralized bootloader (UI).

The bootloader shall support the loading of assets from multiple nodes simultaneously, the validation of assets' integrity, and the resumption of downloads, should a need arise. In the case of our system, one may see all the events flying by as it boots by going to the Chromium developer pane. Here, the process of peers' discovery happens at full nodes through a modified version of Kademlia [12]. As peers are discovered, these are made available to JavaScript components [Figure 5] to be then served through the embedded web server, once the user connects. As assets are delivered, both their integrity and semantics are verified. Should there be any unrecoverable error within any of the fetched ECMA6 modules, the system would refuse to boot. The bootloader supports assets being divided into multiple groups, each of which is to be processed in a separate stage [Figure 6].

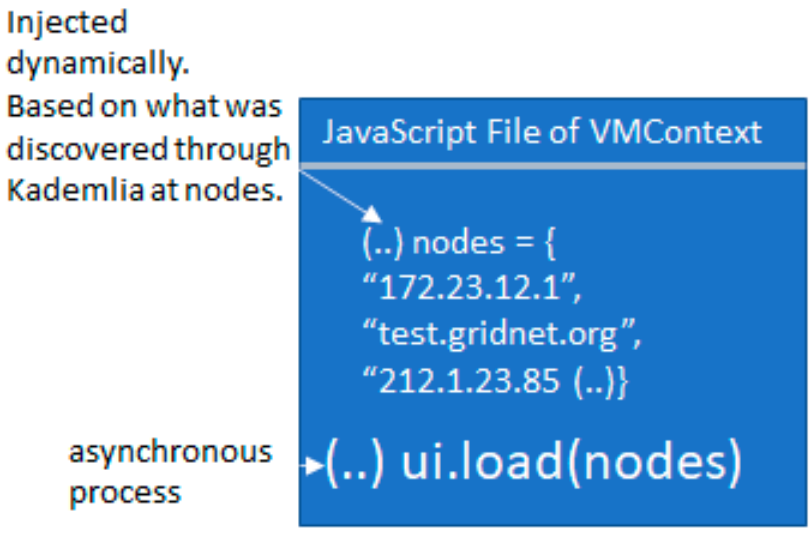

Figure 5. Discovered peers injected into JavaScript files.

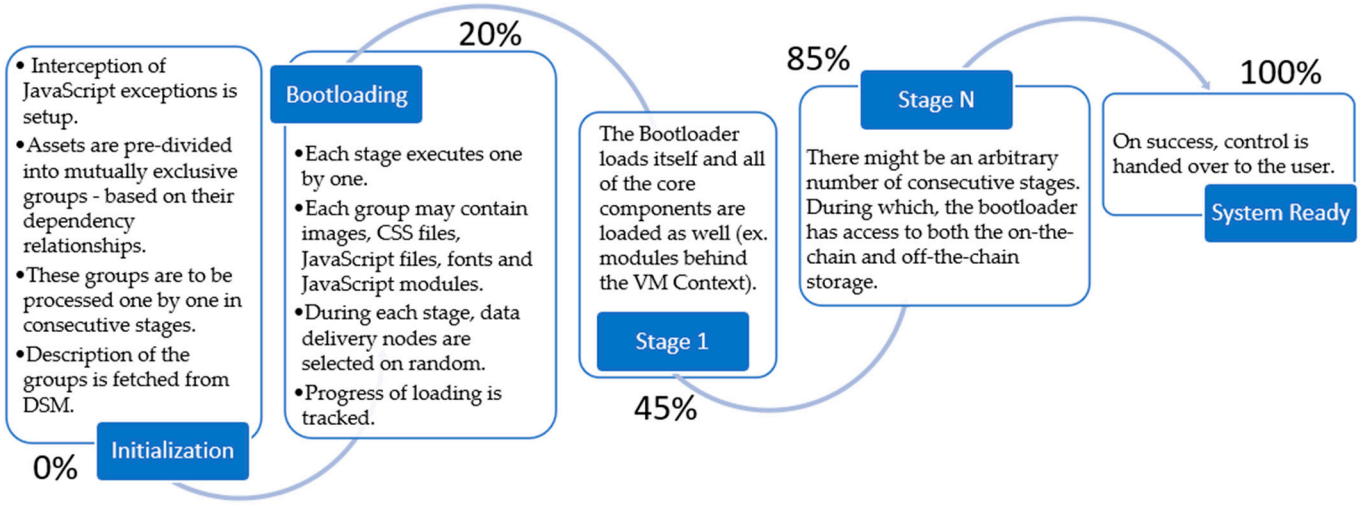

Figure 6. Assets are processed in separate stages.

That is to allow maintainers of the system to ensure that dynamically loaded ECMA6 modules are processed and loaded in the expected order, preventing issues with dependency relationships, had the assets been loaded all at once, or as dictated by the web browser and uncertainties caused by the state of the network.

9. Deferred Authentication State-Full Sessions

Thanks to State-Full Sessions, made available both through the SSH and the UI, the user has a perception of interacting with the DSM in real time. Ephemeral services are being offered to the user throughout the optional grace period while being fully accountable [Figure 7].

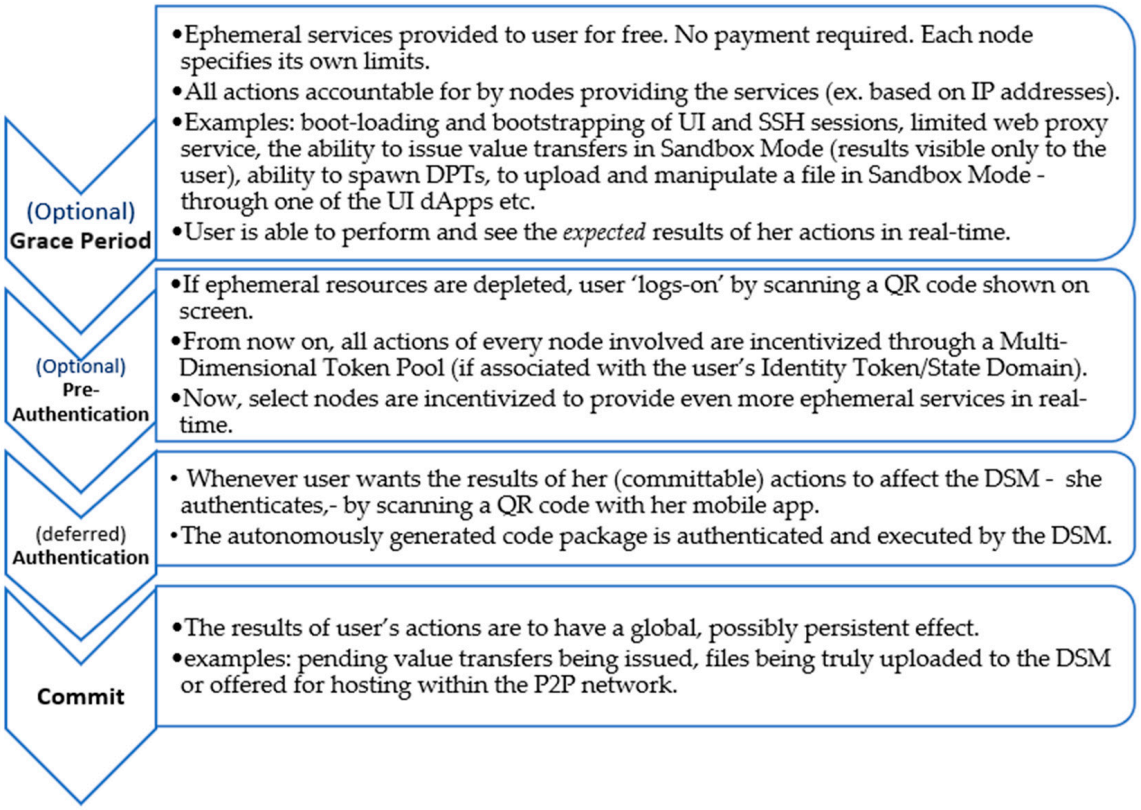

Figure 7. Higher level overview of the deferred authentication functionality.

In fact, what is happening under the hood, as far as the state of the DSM goes, is that the user is implicitly (through UI dApps, terminal commands, etc.) reading from and altering a sandboxed copy of the System Trie, made available to their session through an IVR. In order to allow for parallelism and multi-threading, we came up with the concept of Decentralized Processing Threads. Obviously, the code affecting the DSM needs to be authenticated. Deferred Authentication allows for (1) the tracking of instructions executed by the user once a DPT is caused to enter the 'Compilation State' through the BT command (2) the cumulative collection of the resulting sequences of instructions from possibly multiple DPTs (3) the linkage and optimization of the resulting code bundle, producing a byte code (4) the final authentication of the resulting bytecode package through the user's private key (5) after being propagated throughout the network, the bytecode is then processed at full redundancy, possibly affecting the DSM. The UML diagram below [Figure 8] showcases how the mechanics of Deferred Authentication are implemented through the interactions of a variety of components. Do notice that the mechanics of authentication to the entire DSM—needed once code packages execute at full redundancy—is decoupled from the incentivization of particular nodes, serving the user ephemeral services, possibly leading to the process of the commitment of actions performed within the UI. Nonetheless, the user may be fine with primarily services alone throughout the grace period and may pre-authenticate to particular nodes when out of free ephemeral resources (e.g., while playing games or using web-proxy anonymization services). It is then the task of the operating system to ask the user for either authentication or pre-authentication when and as needed, calculating the amount of needed assets autonomously all the same. Further, we lay out some of the details related to DPTs, which have already been incorporated within the diagram above.

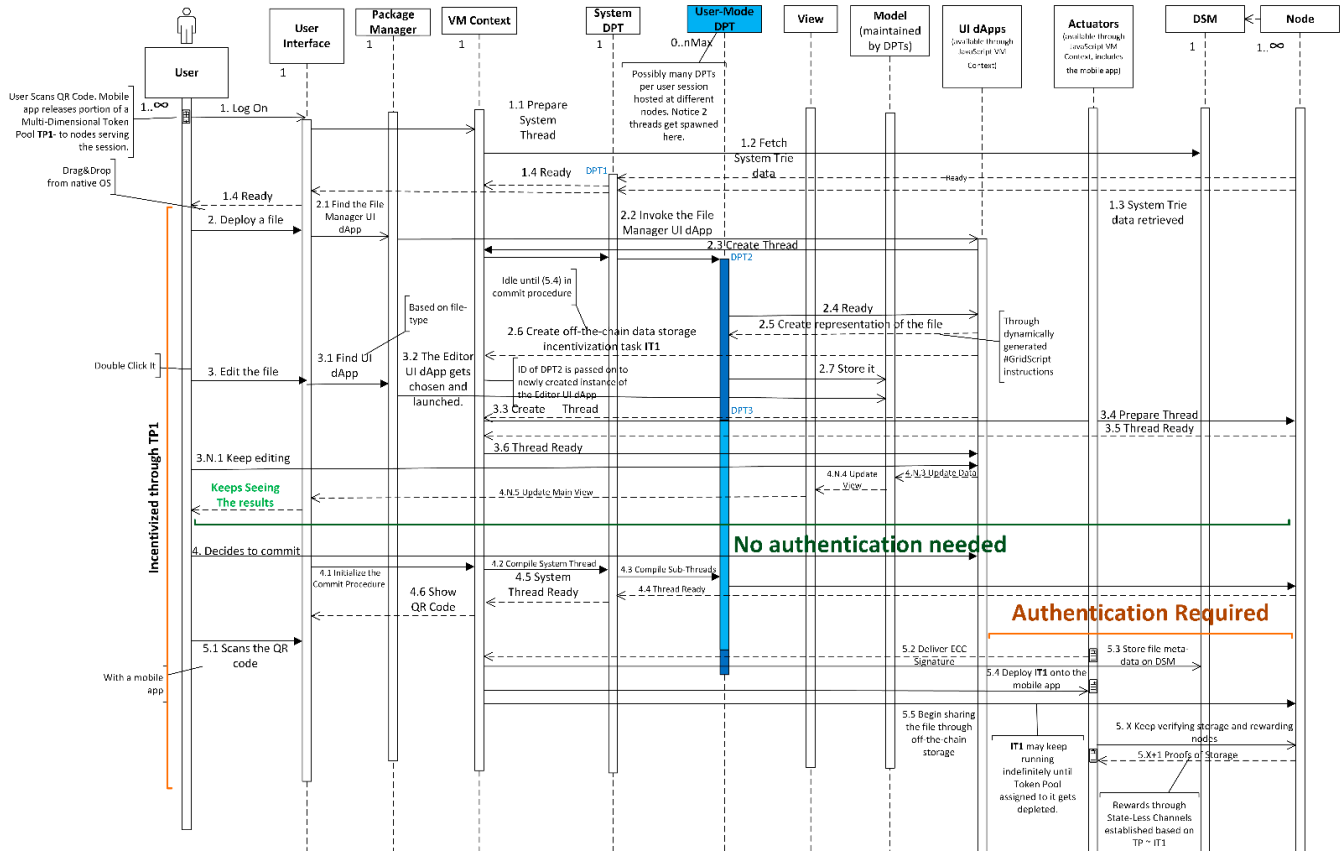

Figure 8. UML diagram showcasing Deferred Authentication.

10. Decentralized Processing Threads

At some point, our development team had a good-looking user interface and a couple of dApps, but something was utterly not right. Lots of synchronization was needed between applications running in the UI. It was difficult to allow for multiple UI dApps to interact with a single code-formulation construct—at least, without them getting in a way of each other. Some applications needed only to perform ad hoc code execution and computation in sandbox mode, say, to read data from the DSM or to see if a file existed, while others wanted to have their associated IVRs enter compilation mode and formulate instructions that resulted in a file on the decentralized storage being deployed or a cryptocurrency transaction being issued. As users kept interacting with an early version of the system, some UI dApps had their code ready to be committed while others were in the midst of achieving such a state. Some wanted to be checking up on data from the DSM at high frequency, while others wanted to do single but persistent data writes once in a blue moon. Then, as soon as the data write instruction was compiled, it quickly turned out that another app would inject data access instructions that (1) needlessly progressed into the committable bytecode and (2) could even have messed up code formulated by the former UI dApp. It quickly became imminent that for some scenarios, we had to allow DPTs to be shared among UI dApps (e.g., the user creates a file in a File Manager and wants to edit it before it is committed to the DSM, but for now, the file exists only in a Private Realm associated with the File Manager UI dApp).

Indeed, as our UI dApps became more and more sophisticated, and as we set out to test more and more of them in parallel, it quickly turned out that better and better isolation among processes was needed (think of 'processes' and/or UI dApps running within a single tab of a web browser). We had to improve the multi-threading and multi-tasking in terms of the code executed in sandbox mode. In particular, we had to support some form of multi-threading, especially in terms of instructions executed on Sandboxed IVR, in parallel. Indeed, it was no longer sufficient for a node to host a single IVR. We had to allow for the better parallelization and isolation of UI dApps and to allow for the uninterrupted formulation of code which would then affect the DSM by each of these. To address these dilemmas, we conceived Decentralized Processing Threads. From the current perspective, the result turned out to be an extremely powerful apparatus of unprecedented possibilities and agility. Had we picked a single concept that enabled a multi-threaded, decentralized operating system pleasurable user experience, it definitely would be for the DPTs, alongside Deferred Authentication.

Threads are one of the most ubiquitous concepts in modern computer science. They allow for the perceived parallelization of instructions' execution and isolation through separate code-execution stacks.

Decentralized Processing Threads serve similar purposes. Each DPT has stacks of its own. In ⋮⋮⋮ Gridnet Os, DPTs execute code written in #GridScript. Thus, the running code has access to all the DSM's opcodes, codewords, and APIs while being underlying native-architecture agnostic. It may define and compile new codewords and opcodes as needed, thus being on these grounds compatible with Forth. Each DPT is associated with an IVR. The latter provides resources to the former, such as a copy of the current System Trie—the instructions are executed by a DPT effect. DPTs can execute in 'compilation mode', which can only be enabled on an IVR running in Sandbox mode. Enabling compilation mode (BT) kicks off the background code-formulation process. From now on, all operations the user performs, which affect the System Trie, facilitated through the very DPT either explicitly from the Terminal or through UI dApps, would be described through UADL and included in the resulting CBCP once the user decides to commit. It is only when DSM runs code at full redundancy, affecting its state that the corresponding (system) IVR is not Sandboxed. A DPT can be marked as committable (#GridScript 'RT' command) at any moment either by user or the associated UI dApp, which is when code from within it may be fetched by Linker (and provided to Compiler) at any moment. In our implementation, VM Context running in the web browser automates the process. If there are processes or threads awaiting a

data-commit, the fact is indicated through the Magic Button, with details shown on mouse hover. The user can then see pending actions and decide whether they want to commit or abort. The system takes care of translating actions from UADL to their human-readable representations. One may thus visualize a DPT running in compilation mode as recording all the instructions executed. These transitions are depicted in a simplified UML Use-Case diagram below [Figure 9]:

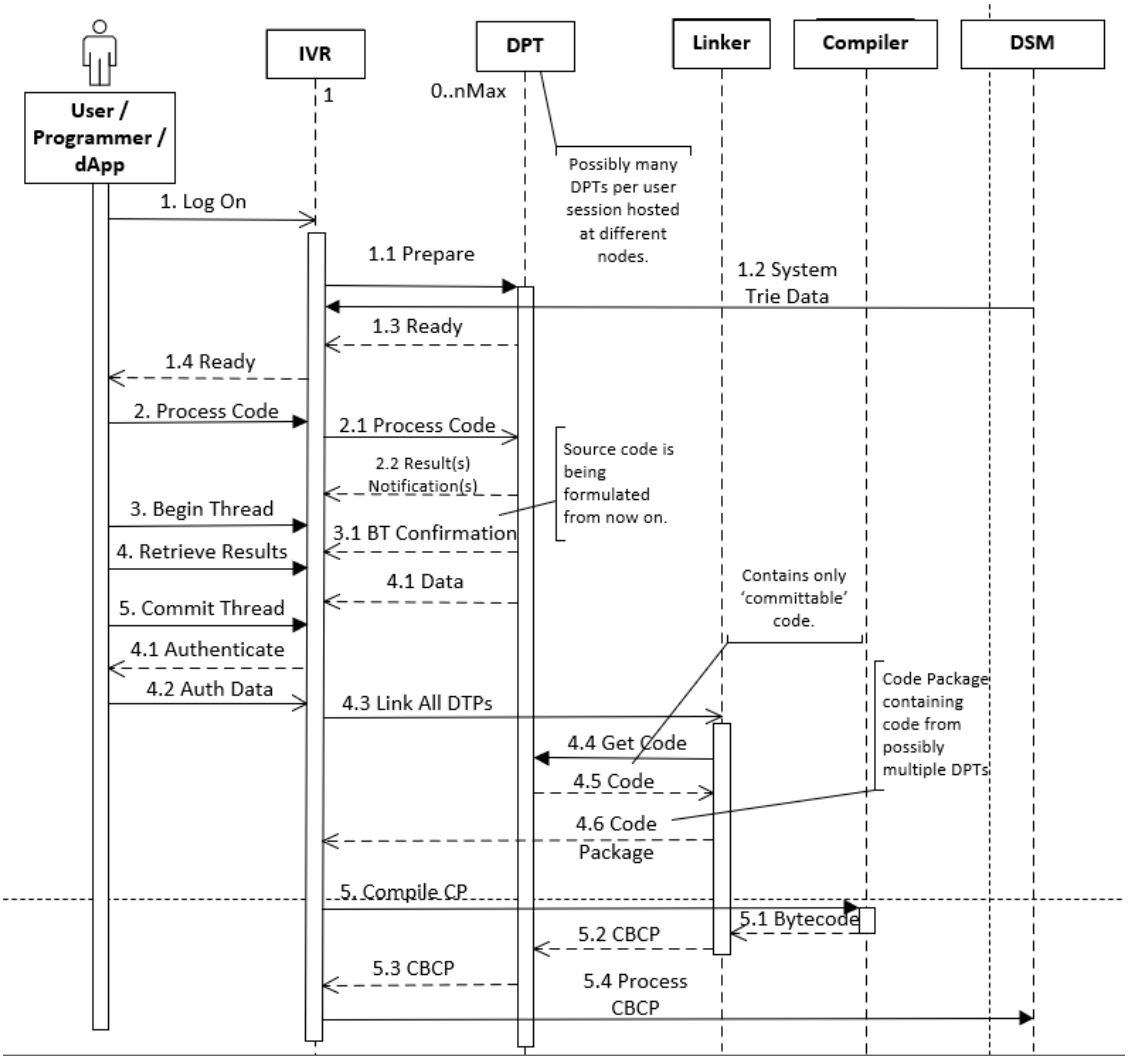

Figure 9. From 'log-on' to a broadcasted bytecode package.

Thus, the product of a DPT is usually two-fold. First, when in sandbox mode, the code running within its scope reacts to the user's and/or UI dApp's input in real time. Second, the instruction-sequences formulated in compilation mode can later affect the Global VM Realm. For this to happen, the thread needs to be marked as ready/committable (#GridScript 'RT' command). Then, the CT command needs to be invoked from the System Thread for the latter to initialize Linker, which aggregates code packages from all the sub-threads (those that were marked as ready—i.e., executed RT). At the end, the System IVR, after having everything compiled and authenticated, broadcasts the signed bytecode throughout the network.

System IVR boarding a single initial System Thread is spawned for each new user-session. The relationship between the DPTs and IVRs is depicted in the infographics below [Figure 10]:

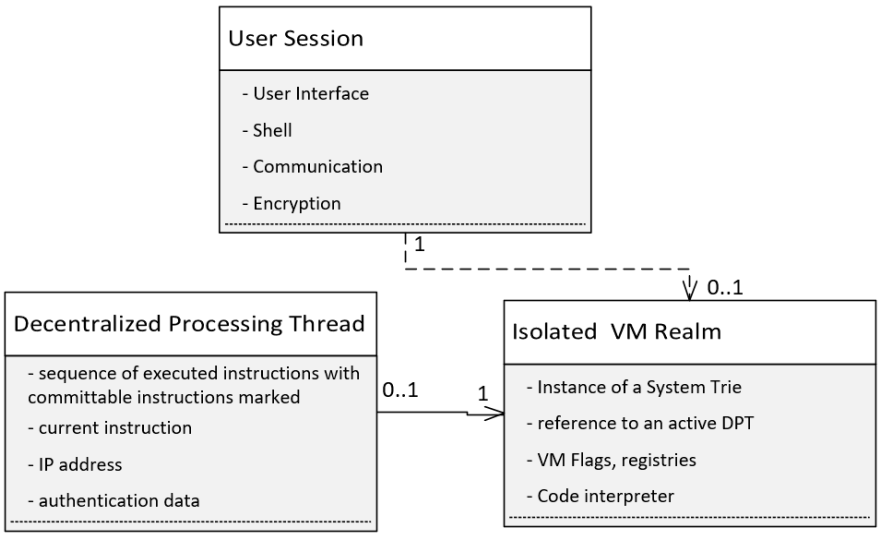

Figure 10. Relationship between User Session, Decentralized Processing Threads, and Isolated VM Realms.

10.1. Interactive Code

A DPT can process #GridScript instructions interactively. The user may keep providing new instructions, and these would be executed right away in a Sandboxed IVR. Oversimplifying a little bit, as the code in between invocations of BT (Begin Thread) and CT (Commit Thread) is executed, the result of those instructions would be replicated across the entire DSM. Some #GridScript instructions allow for multi-stage data inputs also from over the network or mobile app.

10.2. In Terminal (SSH)

Whenever a session for a user is prepared, the node the user is connected to prepares an IVR with a single DPT to begin with, ready to start processing instructions. New threads may be spawned anytime with ST (Start Thread).

10.3. In Browser (Decentralized UI Interface)

Whenever a session for a user is prepared, the node the user is connected to prepares an IVR with a single DPT to begin with, ready to start processing instructions. New threads may be spawned anytime with ST (Start Thread).

10.4. Mobile App and QR Codes

The mobile app provides authentication to particular full nodes and to the entire DSM once CBCPs signed through it are executed. In accordance with the Venice Design Paradigm, it was designed to be simplistic on the surface with all the technicalities hidden away. The app boards the #GridScript code (de)compiler and task scheduler. It is capable of maintaining communication with nodes maintaining the network through a custom UDT-based protocol. Whenever a CBCP is to be authenticated, i.e., signed through the user's private key, a QR code is presented on the user's screen. It is rendered through Unicode 1.1 half-blocks in the Terminal for increased density of data and through WebGL when viewed from the graphical UI. The QR code embeds all the information needed to process the current task, which may be about providing a signature and routing it through multiple hops, end-to-end encrypted over an onion-routed connection to the node which is currently serving the user's session (System IVR). Once received, the node embeds the signature into a CBCP and broadcasts it throughout the network for processing.

10.5. Commitment

Armed with Decentralized Processing threads, it soon became imminent that we needed proper synchronization within web browsers. While $dApp_1$ had to commit executing on the System Thread, we did not want $dApp_2$ to be getting in the way of the former. Thus, we have introduced mechanics similar to those known from database systems, at the heart of which are the tryLockCommit() and Commit() user-mode API functions. Whenever a UI dApp wants to initialize a Commit Procedure, it invokes tryLockCommit() of VM Context, passing the identifier of its process. Any other UI dApp that has signed up for an appropriate event would be notified. Additionally, any other invocation of tryLockCommit() would have failed until the previous commit failed, succeeded, or timed out. The process would be protected and guarded at both the client's and node's sides. Appropriate signaling between these is facilitated at all times through appropriate binary encoded (BER) sub-protocols.

10.6. Cross Node-Browser State Replication

When the user has a UI dApp running, it may happen that the node which is hosting DPT(s) for this very application becomes unavailable. The user might have already performed some work and desires an undisturbed experience. The VM Context would detect such a situation, reconnect to another node, attempt to spawn DPT(s) over there, and replicate the state by executing all the committable instructions all over again, resulting in the very same state.

11. Communication

11.1. Previous Construct of Token Pools

Notice that once the Hash N is known, all the consecutive values up to the Final Hash can be generated from it. This is the main limitation in a multi-peer environment [Figure 11]. In [1], while rewarding peers, the incentivizing node dispatched rewards based on consecutive entries in a hash chain. Each newly released value uncovered additional rewards. The final value could be used to redeem the value of all the previously uncovered entries, once cashed out on the chain. Thus, there was no way to employ a single hash chain for multiple data streams simultaneously. Thus, we could say that the token pool was 'one-dimensional'. In a previous proposal, a peer had either to use a single Token Pool or register multiple Token Pools. That is not 'storage efficiency' ~[Problem 2]. Had they wanted to reward multiple peers simultaneously (during the off-the-chain stage), then they could manage multiple pools and remember about the state of each. That is not 'ease-of-use' ~[Problem 2].

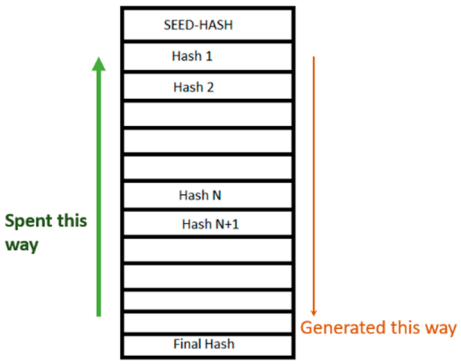

Figure 11. Releasing tokens from a Token Pool.

Rewarding multiple peers with Stateless blockchain channels through the employment of (one-dimensional) Token Pools is troublesome in the presence of asynchronicity caused by parallel peers willing to be rewarded. That is because revealing a single token from the Token Pool required us to wait for the peer to cash it out (on-the-chain) before another

peer could be rewarded through the same Token Pool. Had we not waited and provided a second peer with a further token, the recipient could have cashed out both sub-chains.

11.2. Solution-Multi-Dimensional Token Pool

The Multi-Dimensional Token Pool contains multiple dimensions (banks) [Figures 12 and 13] represented by multiple hash chains. These are generated from the *MasterSeedHash*, which is enough to generate all hashes (tokens) within these. No matter how many dimensions, it suffices for its owner to store the secret *MasterSeedHash* only [*Storage Efficiency~Problem 1*]. In DSM, we only store dimension-boundary points, i.e., ceiling $Hash_{ND_i}$ for each dimension, $Hash_{ND_1}..Hash_{ND_{k-1}}$ $..Hash_{ND_k}$. The values of tokens in each are the same and the dimensions are equally sized. When issuing rewards, in our implementation, the mobile app in possession of *MasterSeedHash* [Figure 13] releases hashes in a reverse order [Figure 14], with each hash representing part of previously frozen/sacrificed assets used to generate the Pool in the first place when requested through QR Intent [*Ease Of Use ~Problem 1*].

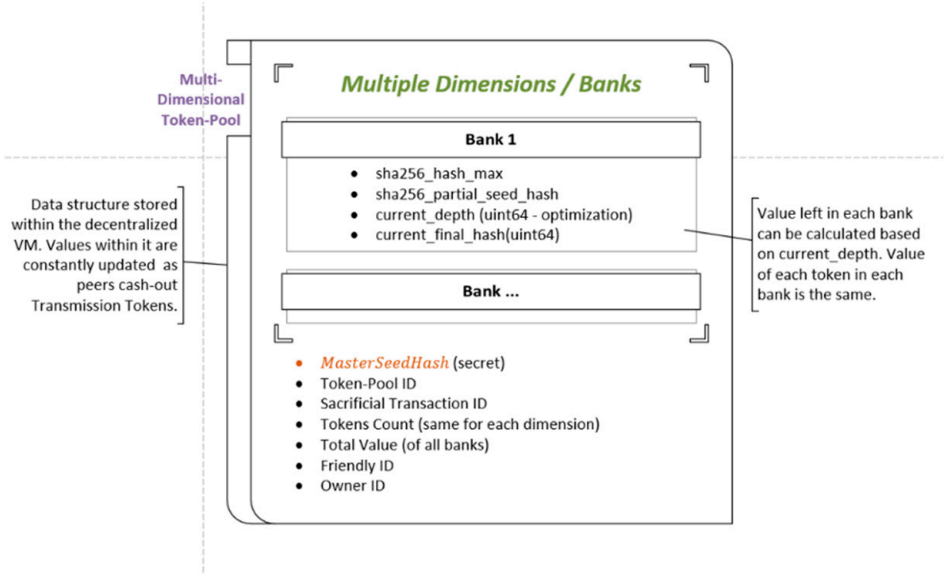

Figure 12. Multi-Dimensional Token Pools.

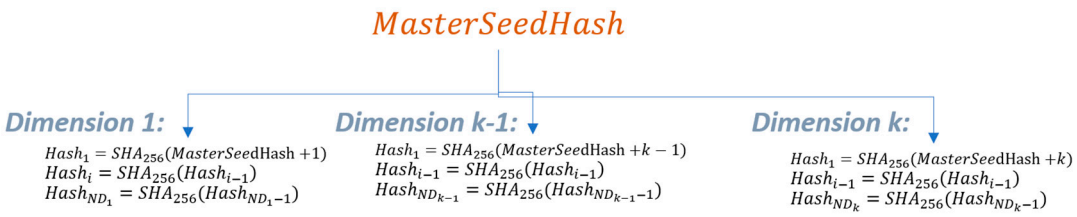

Figure 13. Each dimensions comprises a hash-chain of its own.

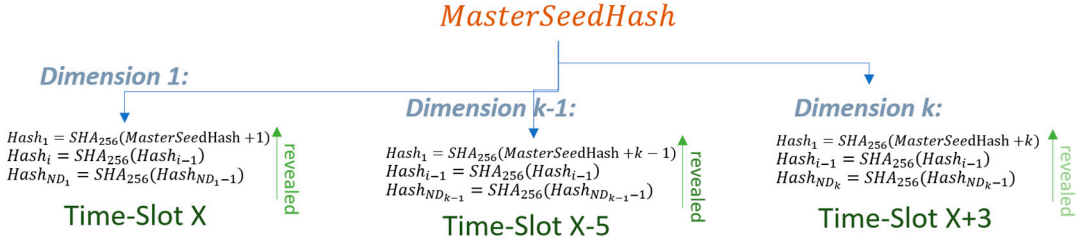

Figure 14. Particular dimensions are independent from each other.

11.3. A-Synchronicity Supported

$Hash_i$—*iTh* hash (token) in a dimension; ND_i—number of hashes(tokens) in the iTh dimension. Indeed, tokens from dimensions can be released independently without affecting each other. To handle N simultaneous content providers, it is enough for the Token Pool to contain N dimensions. The value of a single *dimension* is thus $VD_i = \sum_{i=1}^{N} Value(Hash_i)$ with a total Multi-Dimensional Token Pool's value equal to $\sum_{i=1}^{k} VD_i$.

11.4. A Compatible Transmission Token

So as to make incentivization protocols introduced in [1] compatible with Multi-Dimensional Token Pools, we need to make Transmissions Tokens compatible as well [Figure 15]:

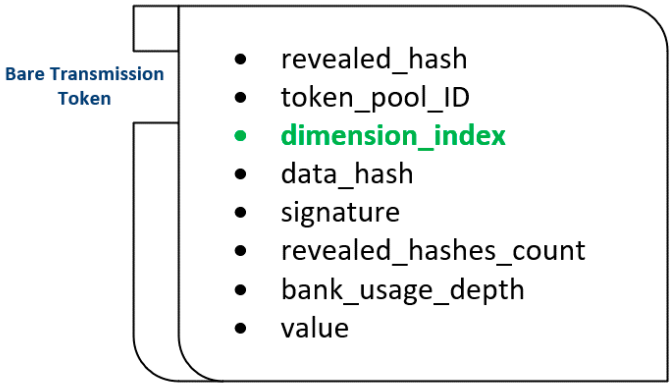

Figure 15. Very low overhead compared to original Transmission Token.

Each Transmission Token now contains an identifier of a dimension from which assets are being released. All the Sybil-proofness guarantees defined in [1] thus remain intact.

11.5. Inter-Process Communication-Signaling and Event-Driven architecture

In our implementation, UI dApps may invoke the user-mode functions of the JavaScript VM Context at will. It is assumed that it is on the shoulders of the operating system (the VM Context is part of) to assure the proper synchronization of the actions performed through these. Each user-mode API call takes the identifier of its caller. UI dApps may sign up for a multitude of events grouped into various categories. These include the global DSM's state, the state of particular DPTs, the state of the connection with the nodes serving the particular user session, etc.

11.6. Integrated Web-Server and WebSockets

When designing the system, we had two options—(1) to rely on the loose coupling between its elements. That road could have theoretically drastically lowered the overall development time and costs. For instance, it would have involved third-party components such as nginx or Apache for serving the 'decentralized UI'; the native operating system's firewall for coping with abuse (e.g., iptables); WebKit's authentication API for authenticating users over the UI; or we would have used TOR for onion routing, or we would have employed ready-made cryptographic constructs for encryption and authentication, or used IPFS for low-redundancy storage, etc. To enable for a Shell or Access Control List, we could have forked one of the Linux distributions and proceeded from there. Had we gone down this road, we probably would have needed to distribute the 'OS' as a Virtual Machine or a Docker container for at least some ease of deployment. The (2) option was to introduce extremely tight coupling and to implement everything from scratch. We set out on the second journey. Looking back, we regret nothing. We now know it was the only way to go.

11.7. Protocol Invariant Routing and Path Discovery (Including Onion Routing)

Each node maintains a protocol invariant routing table. When accessing the Terminal of a node, one may switch to the Events' View [Ctrl+W] to see such routing table entries being constructed.

11.8. Web-RTC Swarms

Each node maintains a protocol invariant routing table. When accessing the Terminal of a node, one may switch to the Events' View [Ctrl+W] to see such routing table entries being constructed.

12. Storage

Without the ability to execute instructions, a state machine would remain in a single initial state. For the machine's state to survive power surges, it demands persistent storage. For full redundancy of storage, we employ Merkle Patricia Tries (used also by Ethereum [2]). Instances of State domains are serialized to nodes within a System Trie. These nodes, in turn, have sub-tries representing folders, files, sub-folders, etc. Whenever anything changes in any of these elements, the hash of its elements changes, and the change propagates to the root of the Trie. In contrast to Ethereum, we allow for the modification and data-pruning of the data stored within the Trie. All the user-mode data is malleable by default by the user who owns the owning State Domain.

12.1. On-the-Chain

On-the-chain storage (Eternal Storage—ES) implies storage within the System Trie. The user pays for each and every byte and, of course, the storage cost may be high.

12.2. Off-the-Chain (Including Rewards for Content Deliveries)

Off the chain storage employs a modified incentivized version of Torrent/WebTorrent peer-to-peer data exchange protocols. In layman's terms, there is a 'link' on the ES to a file hosted off the chain, with ACLs on the ES. In a State Domain, we put a reference R to a possibly large, user-provided file F maintained by the former. As soon as the user attempts to access F through R, F is attempted to be fetched. The entity fetching might be the full-node (if the file is accessed over SSH) or the web browser. The mobile app constantly accounts for and releases assets through a Stateless Channel on the node's request, facilitated through QR Intents as nodes use their bandwidth, acting as a proxy. For the incentivization of storage, the mobile app steadily releases remuneration to those hosting the file by means of off-the-chain transactions facilitated through Stateless Channels [1] implemented through the just-introduced Multi-Dimensional Token Pools. The algorithm follows:

Threat Model: Client Λ_1 wants to have his file stored by *a set of identities* $\mathbb{A}_R$; β is the storage redundancy threshold; R_{min} is an aggregated reward in the system-intrinsic cryptocurrency which Λ_1 undertakes to pay to members of $\mathbb{A}_R$ during a number of epochs of the decentralized state machine S_T, defined as the number of consecutive key blocks appended to the DSM. Assume $R_{min} \geq \mathbb{A}_R \neq 0$. For each storage epoch, $\mathbb{A}_i \in \mathbb{A}_R$ expects to receive remuneration of at least $\left\lfloor \frac{R_{min}}{\mathbb{A}_R} \right\rfloor$. Λ_1 is not necessarily expected to verify storage or issue rewards after each epoch. It is up to $\mathbb{A}_R$ to decide whether they want to keep maintaining F. The array of verification vectors V_V allowing for the verification of storage during the entire, possibly undefined storage period is known only to Λ_1. S_T thus defines the maximum storage time defined in the number of epochs should Λ_1 choose to verify and issue rewards each and every epoch. Λ_1 issues rewards only to those who manage to deliver a valid Proof-of-Storage P_S. Attackers attempt to trick Λ_1 into believing they store the file, while Λ_1 attempts to trick $\mathbb{A}_R$ into thinking that a reward was issued, when it was not.

Proof-of-Storage: There is a file F which we model as an ordered sequence of bytes, issued by Λ_1 to be stored by $\mathbb{A}_R$. $F_{V_S}^{V_E}$ is a sub-vector of F, where V_S and V_E are the starting and ending offsets, in bytes, from the beginning of F, respectively.

For members of $\mathbb{A}_R$ to prove storage of F to Λ_1, the latter first generates an ordered sequence of vectors $V_V = \sum_0^{S_T} \left[Sha256\left(F_{V_S}^{V_E}\right), V_E, V_S \right]$ with random values of V_S and V_E, such that $V_E > V_S \forall V_i \in V_V$. Thus, $|V_V|$=S_T; V^I —the index of the recently used verification vector $V_i \in V_V$. To verify the storage of F, Λ_1 broadcasts a verification request $V_R = [RequestID, F_{ID}, V_S, V_E] \sim V_i \in V_V$ where F_{ID} is a unique identifier of F and increments V^I so to never use it again. As V_R is read by $\mathbb{A}_i \in \mathbb{A}_R$, the latter responds with $P_S = Sig_{\mathbb{A}_i^{SK}} \left[Sha256\left(F_{V_S}^{V_E}\right), RequestID \right]$, where $\mathbb{A}_i^{SK} \in \mathbb{A}_i \in \mathbb{A}_R$. Once Λ_1 receives $V_i^{'} \sim V_i \sim P_S$, it checks if $V_i^{'} = V_i$. If true, Λ_1 issues rewards through a Stateless State Channel [1] $\forall \mathbb{A}_i \in \mathbb{A}_R$. Should any $\mathbb{A}_i \in \mathbb{A}_R$ not be rewarded as expected, $\mathbb{A}_i$ stops storing the data and responding to verification requests. Notice that members of $\mathbb{A}_R$ may freely change at any time. Those who choose to store download the file. If more than β nodes respond to V_R, Λ_1 may choose any heuristics it desires to pick members of $\mathbb{A}_R$ (download speeds, etc.)

Discussion: In our implementation, V_V is stored on a mobile security app, which acts in the name of Λ_1, steadily realizing rewards to $\mathbb{A}_R$. The benefit of this approach is that Λ_1 does not need to store F at all after they generate V_V. The algorithm covers the case when the integrity of file was needed to be verified (e.g., private large, encrypted files) without any other agents besides Λ_1 having an incentive to have access to the F. Should incentives be structured differently, though (i.e., in case of 'public-goods' say movie clips, whose integrity needs not be verified), other parties may continue issuing rewards to $\mathbb{A}_R$ as they are having $F_{V_S}^{V_E}$ delivered. Thus, the protocol may effectively constitute a content-delivery incentivization mechanism, with $\mathbb{A}_R$ expecting to be rewarded after each and every $F_{V_S}^{V_E}$ they deliver to those to whom they stream F. In our implementation, all the user needs to do, from time to time, is to scan a QR code displayed on their computer (on which they are accessing decentralized videos/files/streams). That causes parts of a Multi-Dimensional Token Pool being uncovered to the web browser, which is assumed to be semi-trusted. It would receive portions of multiple Token Pools' dimensions but not the entire thing. Notice that uncovering just a single, previously unknown hash suffices to release many tokens to the web browser. These are then used (by the web browser) to unveil single tokens to peers delivering data. Should Λ_1 wish to have F stored for more epochs than defined by S_T, he could download the file and regenerate a new verification set V_V. In order to improve protection against colluding attackers in $\mathbb{A}_R$, Λ_1 could use $|V_V| = |\mathbb{A}_R| * \mathbb{A}_R$ and issue a separate $V_i \forall \mathbb{A}_i^{SK} \in \mathbb{A}_R$.

12.3. Access Control Lists and Ownership

Access Control Lists (ACLs) are widely employed mechanics on UNIX/Linux and recently the Windows family's operating systems. When enabled, each object on the file-system may have an associated list specifying privileges to agents through Access Control Entries. Our implementation fully supports access control lists. It is possible to define read and write the execution and ownership of any object within any State Domain. That opens a pandora box of entirely new possibilities, not seen before in any other decentralized state machine.

12.4. Non-Fungible Tokens Reinvented

Had there been a version of Linux running on some sort of a decentralized state machine, would there be anything special about NFTs [7]? Would the term ever be coined? Probably not. GRIDNET OS in a way comprises such a decentralized Linux. It allows to assign ownership to files and folders. In fact, it allows to alter these properties through the Linux-compatible setfacl, getfacl, and chown commands, thanks to the aforementioned ACLs associated with each object in the System Trie. Each bytecode package is authenticated to a single identity. Once the code is executed, the security sub-system at each node compares the identity with security descriptors associated with the accessed object. As for NFTs, one may execute the setmeta and getmeta commands to see their documentations.

13. Abuse Mitigation

Our implementation boards autonomous embedded anti Denial-of-Service measures. These account for all communication sub-systems (embedded web-server, web-sockets, web-RTC swarms, UDT communication and SSH sessions). Should the system detect suspicious activity, the suspect is banned autonomously for a random period of time. One may access firewall utility from over the Terminal, though. Its functionality is limited from remote sessions.

14. Venice UI/UX Design Paradigm

Venice is a beautiful city 'It is a city of mirrors, the city of mirages, at once solid and liquid, at once air and stone. '—Erica Jong once said. It is pretty much the same with decentralized systems. Blocks come and go, and with them, the current state. Nodes one may be connected with at time slot T_i might be no longer T_{i+1} Still, the user is after a pleasurable user experience. We depict the following essential elements of the Venice UI Design Paradigm for UI dApps:

Remain Responsive—it is so much more difficult to look good and behave well with all the uncertainties around but do your best. The user interface should remain responsive at all times and user should be notified whenever she needs to wait. This may be achieved through techniques such as pre-fetching and notifying the user about the current state.

Offset The Uncertainties—the uncertainties may stem directly from the nature of a decentralized state machine or from the technology used to render the user interface. The solution here is the hiding away of the uncertainties through automation, aggregation, and hiding away.

Be scarce—on resources, especially with full-redundancy storage. It is expensive. Each full-node operator pays for it (literally). Use lower redundancy storage when possible.

Isolate—assure boundaries between processes. That goes for both code and UI elements. Since as of now, web browsers do not provide isolation mechanics for Web Elements of the same quality as those available in native operating system systems, it is in a large portion the responsibility of each and every UI dApp not to misbehave. This would change in the future, as web browser frameworks provide better isolation constructs to decentralized operating systems, and isolation boundaries would rest firmly on the components of the latter.

Want to be Incentivized—there is nothing wrong with being willing to be paid for your work. It holds true for your services.

Incentivize Others—if other services or agents do work for you, remunerate these as well. Even if your UI dApp doesn't take advantage of third-party services, it still employs core nodes to load itself. Do not be reluctant to reward the Core nodes implicitly (system-wide taxes) or explicitly.

React—both to user inquiries and notifications made available to you by the Decentralized Operating System.

Client Side (Shadow DOM, windows)

In order to improve the isolation of CSS definitions defining the looks of the UI dApp's elements, we have employed a Shadow DOM. Each window in the user interface thus wields a separate Shadow DOM's root, thus not affecting other windows. MAGIC BUTTON—is at the heart of GRIDNET OS' UI. When hovered over, it shows all the pending actions to be committed to the DSM. All the ready DPTs would be committed on a single tap/click. If there are no pending actions, it would synchronize the state of all IVRs attached to all UI dApps with the current Global VM State. THE CURTAIN—one of the more interesting aspects of the user interface. It employs Webkit's Monitor with Animation APIs to hide away contents of a window as these are loaded and/or rearranged. GRIDNET OS makes Curtain APIs available to the user, should the user need to customize its default barely transparent looks. Start-Menu—the elements are rearrangeable and grouped into categories. Taskbar—the elements are rearrangeable, with the support of drag and drop with changes persistent across sessions. Elements on hovering present current windows'

previews, just like on Windows or macOS—we have borrowed from both Windows and Mac OS X when it goes for these. Once you move a window close to any edge of the screen, it would resize itself accordingly. Maximizing, resizing, and minimizing to the taskbar—it is all there. We employ the concept of dApp Packages. A package is comprised of both code and other assets (images, sound files, etc.). We have borrowed from Android with the concept of namespaces, so here, each UI dApp wields a namespace of its Integrated UI and code elements within one package, deployed at full nodes with *possibly* external assets.

15. Overcoming Limitations of Previous Architectures and Evaluation of Results

15.1. The Architecture

Bitcoin [1]—the very first decentralized state machine, executing instructions in a programming language—Script. Ethereum [2]—with the major improvements of allowing for more developer-accessible storage constructs, having introduced signaling between code entities—'smart-contracts', and above all, having allowed for authenticated Turing complete instruction sets. Many other projects were brought to life, such as EOS.IO [13]—improving both upon the scalability and aiming to aid developers through Web Assembly smart contracts.

One inherent dilemma remained. Our work has focused not so much on improving the scalability of the underlying state machine, but on researching and paving the ground for unprecedented, integrated possibilities. With that said, as it is to be seen, we've arrived at some very sound performance results. Previous architectures, such as [2,13], were concerned with the intrinsic language and the dilemma of scalability. Everything else was to be deemed as 'client code'. One could thus imagine previous architectures as relatively simple robots, concerned with how to communicate and how come to an agreement upon matters discussed in the language they spoke. Now, dApps' developers were to be writing code letters in that language for processing, possibly relating to (calling a smart contract) what had been previously agreed upon.

The blueprint of the hereby proposed decentralized operating system is concerned with more. We plug into the robot of ours lots and lots of additional sensors and actuators to be used for the construction of decentralized applications of any kind and we make these available at the fingertips of developers. Attaching prior architecture to an external web server would never be enough once one paid attention to the meticulous details and fathoms that full nodes need to be aware of particular windows' dimensions and account for these in real-time. Ethereum or EOS.IO, in their current form, would not be able to accommodate these 'peripherals' due to the aforementioned lack of deferred authentication mechanics and thus of the ability to react to users' inputs in real time. We make sure all the elements integrate with each other, not only through APIs but through algorithmically model-ed threat models and proofs (Sybil-proof crypto incentivized data exchange, the construct of Multi-Dimensional Pools) and that the usage of these remains incentive compatible, all with the aim of fulfilling all the end-users' and developers' needs, in-house, through a unified environment. We propose that such a system should take it from consensus, through the storage of data through arbitrary redundancy guarantees, through the hosting of model/view logic, incentivized threat-model-validated data exchange and storage, up to making sure that all components are properly delivered to the user. The test-bed of the hereby proposed operating system is readily available for testing from the main access node at https://test.gridnet.org (accessed on 19 September 2022).

One of the interesting aspects of decentralized operating systems, and the implementation of ours, is its ability to take care of incentivizing data exchange. Undeniably, the majority of developers' concepts boil down to data. Everything is data. Thus, it shall be of a paramount importance to ensure that data can move freely. Below, we present evaluation results of the incentivized data exchange sub-system, as per its current implementation. The evaluations to follow include references to algorithms and protocols previously introduced in [1]. The following system parameters were assumed: the interval between data blocks: 3 s; maximum data block size: 11 MB. For the below calculations, we assume an average

bandwidth utilization of 11 MB (block size) at an average speed of 4 MB/s. To make our scenario more challenging, we assume the user forms a new path every 15 min.

From the algorithmic perspective, our analysis covers three cases: the modeling of the performance of the protocol for the incentivization of the established data paths, the protocol for data deliveries where the sender sends out data packages, knowing only their destination and not knowing whomever the intermediaries might come to be. Finally, we evaluate the third—which takes the use of the prior for the full incentivization of data transmissions. More details are found in [1]. It is worthwhile to note that while we evaluate performance in the context of data exchange, the performance of a data path of 1 hop in length can be seen as the overall achievable transaction per second throughout when there is a single recipient. Further, the numerosity of possible 'connections' can be treated as the possible number of open State Channels, with intermittent on-the-chain payouts happening as per the simulation properties.

The article comes with a simulator in Excel, where one can manipulate all the variables, inspect the parameters, and probably notice a lot of fascinating properties as the simulator allows for specifying how likely data paths are to change, the portions of data blocks that are to be used for the incentivization of data exchange, etc., and see how all these parameters affect the achievable maximum amount of possible connections and/or State Channels.

Below, we are to evaluate performance both at the systems' current max throughput parameters under the most demanding scenarios and in a more relaxed scenario, which we call a 'realistic one'. The most demanding case assumes that intermediaries of a data exchange are to be rewarded as soon as either the data path changes or a connection ends, thus dealing with a scenario which is supposed to yield the most overheads on both the DSM and generate the most network overheads as well. We also tackle a more relaxed, 'realistic' case, which takes into account the current layout of the Internet and the users' behavior, which we describe as follows. According to the empirical study of Wang et al. [14], the average path length between autonomous systems on the Internet is about 3.9 and the longest path is about 20. Now, we are to simulate real-life usage, with users accessing remote endpoints at random distributions over access providers, which we assume are less likely to change than the target endpoints. We thus stem for the assumption that the ending 20% of any given data path is 90% more likely to change than the prefixing portion of it, as we assume that should visualize the typical user browsing the Web. The average TCP/IP packet size was calculated as an average of all of the months of Internet traffic in Chicago as of 2016, reported by CAIDA [15] and resulted in an average packet size of 907 bytes.

Intermediaries are rewarded through Stateless Channels [1] with assets allocated from Multi-Dimensional Token Pools [1]. Yet again, while we simulate incentivized data exchange, it is worthwhile to note that the case of a single hop can be considered as a case in which the recipient receives direct payments for any kind of a work, performed at the discretion of the payer. We believe data exchange with billions of data packets is one of the best showcase scenarios.

Let us consider a test network under default performance cap settings (data block-size set to 11 MB, a new data block occurring every 3 s and analyze the achievable throughput and the point at which a bottleneck occurs based on the link properties (path variance, number of hops). Additional simulation parameters are in [Table 1].

Table 1. Some of the significant simulation parameters.

Payout Confirmation Data-Structure Size (Bytes)	Transmission Token Data Constant Overhead PA_2 (Bytes)	Average Payload Size per Datagram(Bytes)	Single TP Dimension Overhead (hash+ depth) (Bytes)	Number of Packets to be Exchanged during Transmission
132	73	907	36	12128

Table 1. *Cont.*

Average Payload Size Requested by User	Token Pool Data Structure Overhead(bytes) (no Dimensions)	Single-Packet Network Overhead (bytes) without AES	Single-packet Network Overhead (bytes) with AES	Smart Onion Co-Efficient
11 MB	60	179	195	0.1
Blockchain Block Interval (Minutes)	**Sender's Lock-Release (PA_1) Size(Bytes)**	**ChaCha20 AEAD Overhead Bytes**	**Pub. Key. Field Size (Accounts for Sign on Elliptic Curve) (Bytes)**	**PA_2 Additional Per TT Overhead (Bytes)**
3 sec	96	32	33	106

First, is evaluation of results for a scenario in which data exchange happens solely by means of algorithm PA_1 [1], which is when nodes to be encountered during path traversal are unknown; thus, it is used during path discoveries. Protocol PA_2 [1] is used for the incentivization of data-transmissions during already established data paths. Both PA_1 and PA_2 are employed by PA_3 [1] which we use to simulate a typical, real-life scenario. During 'strict' analysis, we assume that transmission tokens traverse alongside each and every datagram. In the more realistic 'relaxed' scenario, nodes fall back to a certain threshold, which in our case, sent a transmission token in 1% of datagrams. As seen in the attached simulator, even that yields 121 'checkpoints' during the transmission of a payload, which is 11 MB large. During these checkpoints, the intermediaries of a data exchange receive payments, thus at any point in this example, risking the retransmission of upmost 100 KB without being paid. This amount, surely any router would be willing to risk.

For the data field, AES encryption overhead has been taken into account, characterized by the following equation: cipherLength = (clearTextLength/(16 + 1)) × 16;

As for the 'onion' data structure within the data-packets used, it is encrypted using the Salsa20 stream cypher. The symmetric key is derived by performing Diffie–Hellman with the recipient's public key. The secret persists for the duration of a connection link, i.e., for as long as a Transit Pool lasts. When the path changes, the new session secret key is generated. The size of a Transmission Token as of writing is 73 bytes. For transmitting public keys, we use a 'compressed' form, i.e., we transmit only one coordinate on the elliptic curve.

15.2. Influence of Path Instability on the Amount of Data Stored inside of the DSM

Path instability has a direct effect on the amounts of data needed to be stored by the DSM and thus on the number of simultaneous data streams that can be incentivized [Figure 16]. That stems from the assumption that intermediaries want to be rewarded, on-the-chain, by the end of each data transmission, they happen to be participating in.

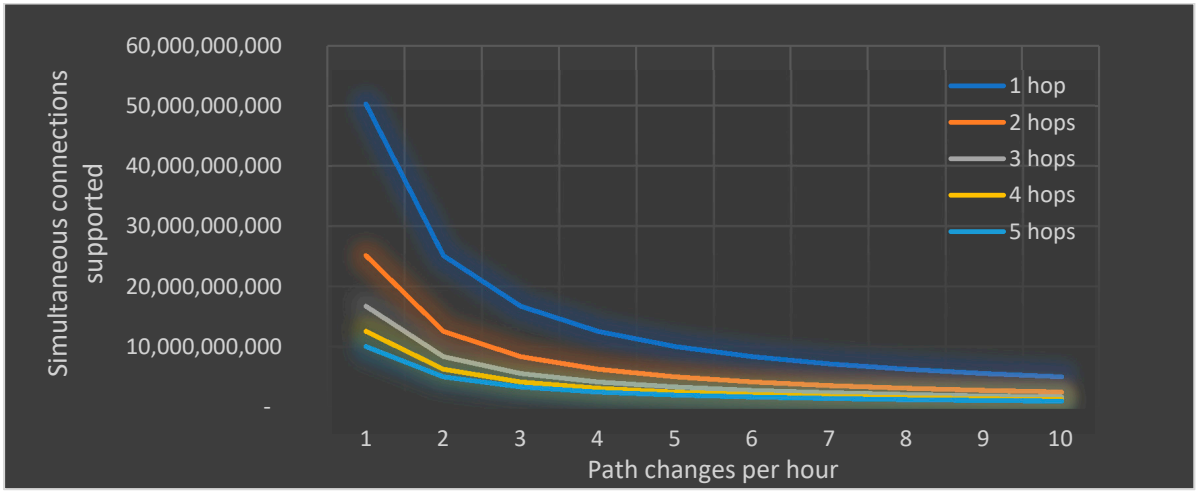

Figure 16. Effect of path instability on the number of simultaneous connections (active State Channels).

As seen in [Figure 17], we can offset the diverse effects of path instability by either (1) increasing the number of dimensions within Multi-Dimensional Token Pools, so that clients do not need to close Stateless Channels after each data transmission. With Token Pools of multiple dimensions, we are able to proceed with another dimension, even though the State Channel, facilitated through the previous dimension, was not closed yet. (2) Or, by falling back to a lower frequency of rewards issuance. Both these procedures contribute to the maximization of the numerosity of achievable simultaneous and incentivized data streams by the DSM.

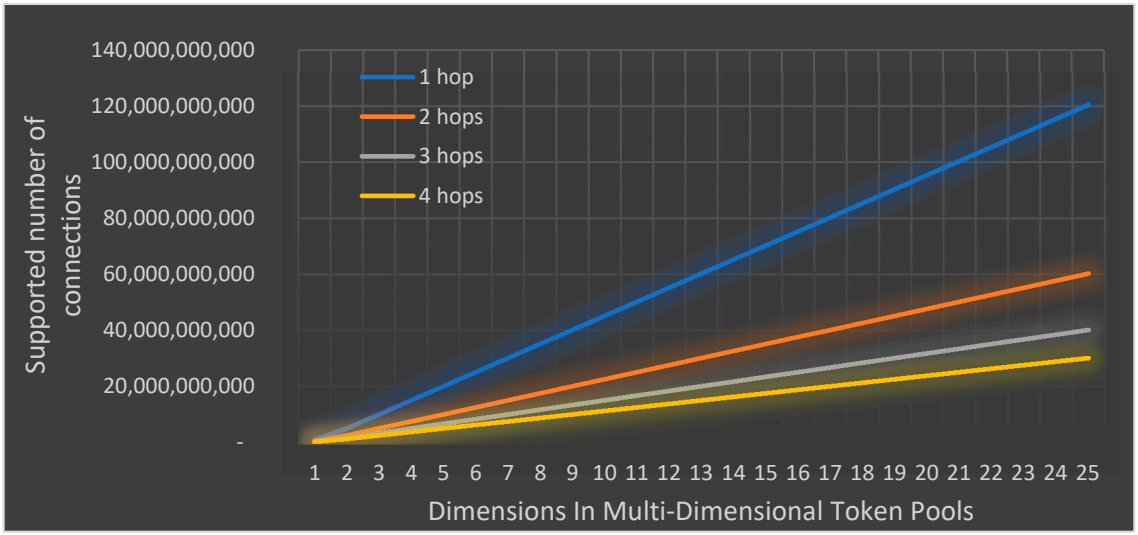

Figure 17. The effect of token pools' dimensionality on the supported number of achievable simultaneous connections and/or state-channels.

In the case of a single payout clearance and just a single hop, the size of a data structure needed to be stored within the DSM is equal to 408 bytes. That is the size needed to confirm payouts to one intermediary and one recipient, with an arbitrary [2] amount of data flowing in between. In the case of no interruptions, i.e., no path changes during data transmission, that would be the only data stored on the DSM. The AES encryption of the data part accounts for less than 2% of bandwidth overhead. In our DSM, it is used mostly during path discoveries, after which the connection falls back to a state which is more bandwidth-efficient, as in PA_2 we are exchanging hashes from Multi-Dimensional Token pools for most of it, which comprise payments, of course. That, broadly speaking comprises how it operates. Looking closer, the size of a single 'on-the-chain' Transit Pool payout clearance is calculated as below:

$$payoutClearanceOverhead = PayoutRequest + PayoutConfirmation;$$

$$PayoutRequest = \varepsilon + (\rho * 33) + \epsilon + \tau$$

where ε stands for a secret from inside an onion, ρ for the number of intermediaries, τ size of the Transmission Token and ϵ represents the ECC public session key used to encrypt a symmetric key.

The number of intermediaries has a direct impact on the size of metadata attached to a data packet, as the data structures depicting payouts (Transmission Tokens) need to account for more agents involved. Payout clearance might be initiated at any time by any agent involved [1]. The relationship between path length and achievable TPS, for both PA_1 and PA_2 is visualized in [Figure 18].

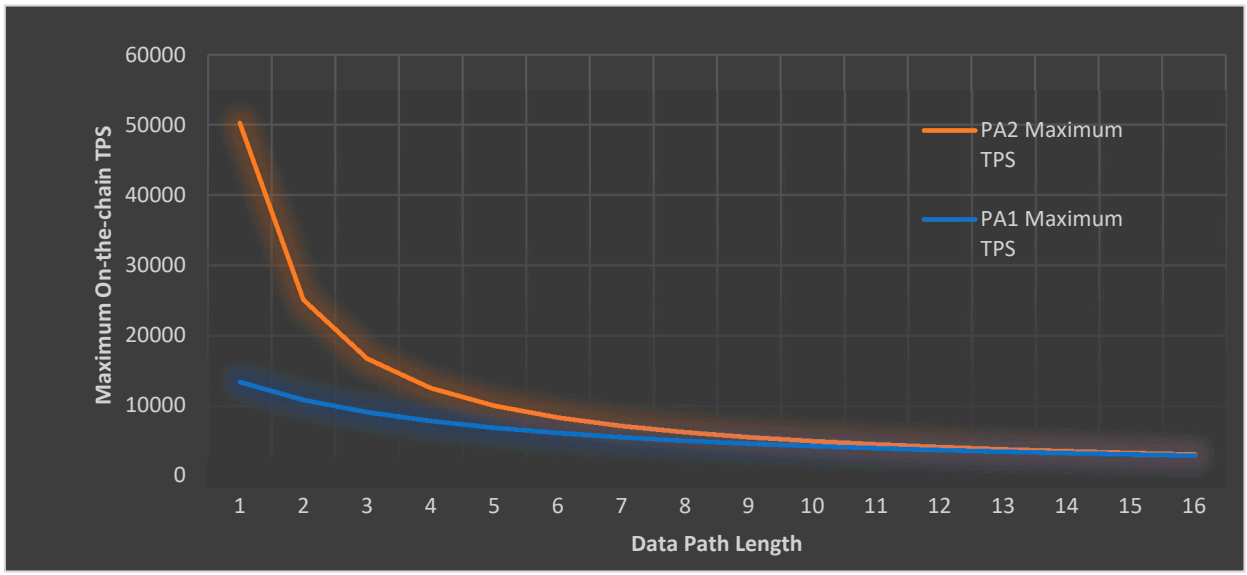

Figure 18. The effect of path length on the achievable TPS in a 'strict' case.

Each compartment on the x-axis represents the achievable Transactions Per Second (TPS) statistic achievable at a particular path length. If the user is in possession of a Token Pool with multiple dimensions (Multi-Dimensional Token Pools), they wouldn't need to initiate payout clearance each time a new data path was to be formed, as they would simply pick a new dimension from the Token Pool to incentivize the newly formed data-path and continue from a previous one once and if the previous data path turns out viable again.

The number of simultaneous connections supported by PA_3 (taking use of both PA_1 for path establishments and PA_2 for actual data exchange) in the 'realistic' case is visible below [Figure 19].

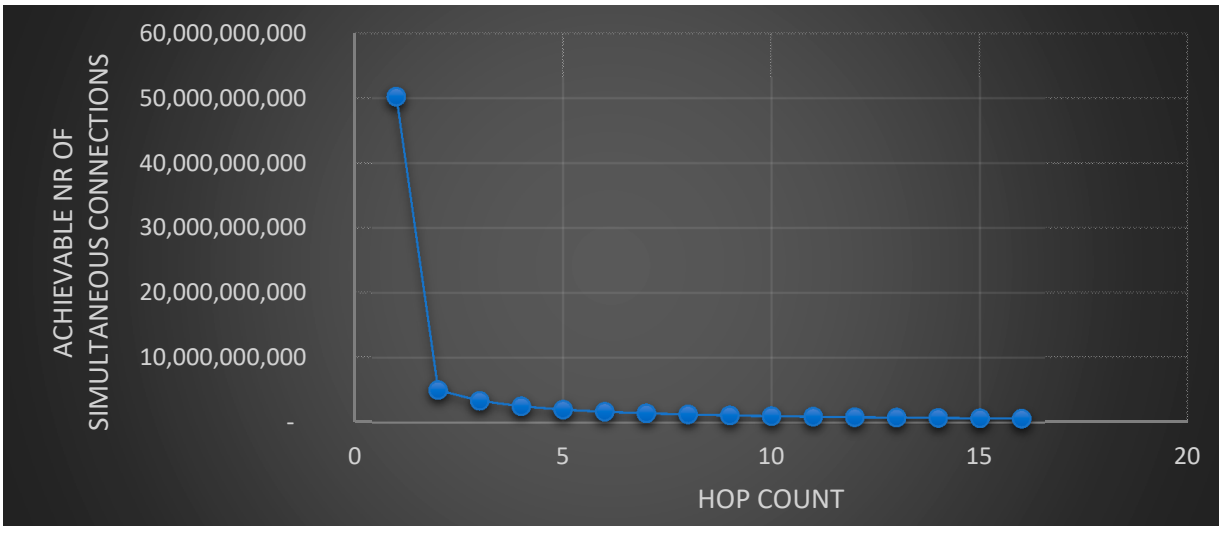

Figure 19. Achievable simultaneous connections based on hop-count. Realistic scenario in which nodes perform a fallback to having transmission tokens attached alongside 1% of data packets.

Achievable theoretical number of simultaneous connections τ can be calculated as follows:

$$\tau = \left(\frac{\left(\frac{bs}{pcs} \right)}{\frac{\mu}{\sigma}} \right)$$

where, *bs* stands for the blockchain blocks size, *pcs* for the size of a single payout clearance, μ is the blockchain block interval, and σ represents the path variance.

15.3. Evaluation of Multi-Dimensional Token Pools for Rewarding for Arbitrary Services

As has been discussed, the number of dimensions within Multi-Dimensional Token Pools contributes directly to achievable performance in terms of simultaneous, active State Channels. The below [Figure 20] visualizes the relationship between the number of dimensions of a single Token Pool and the amount of DMS storage needed to accommodate it.

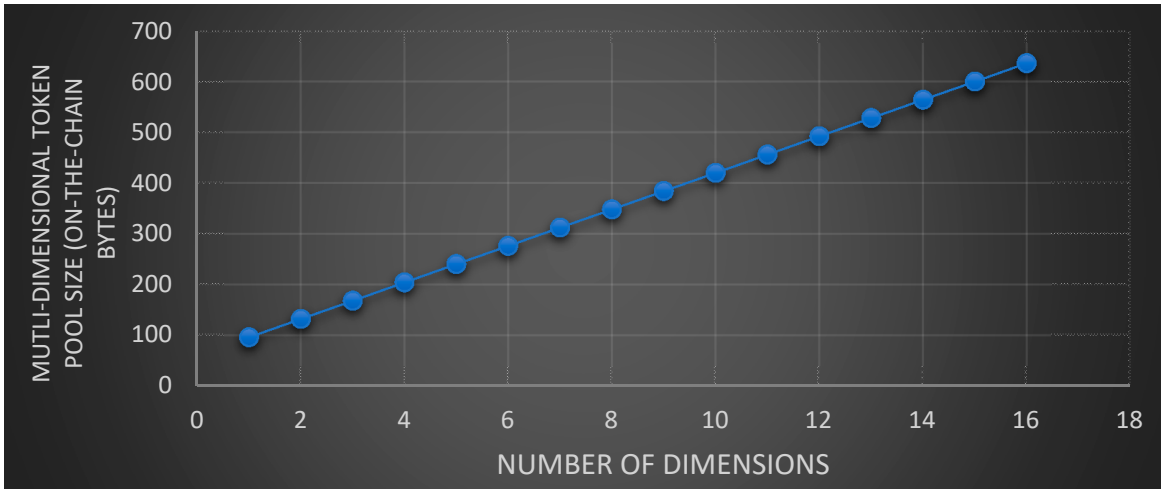

Figure 20. Dependency of the resulting Token Pool's data structure size on the numerosity of dimensions.

Broadly speaking, each dimension involves the storage of a yet another seed hash. The number of tokens available from a single dimension does not affect the size of a Token Pool, though.

While a dedicated research paper up taking various analysis of the data exchange incentivization sub-system is pending, we invite the reader to already play around with the attached simulator, where details with regard to the formulas and constants describing the overheads of particular low-level data structures can be found seen in [Figure 21] below.

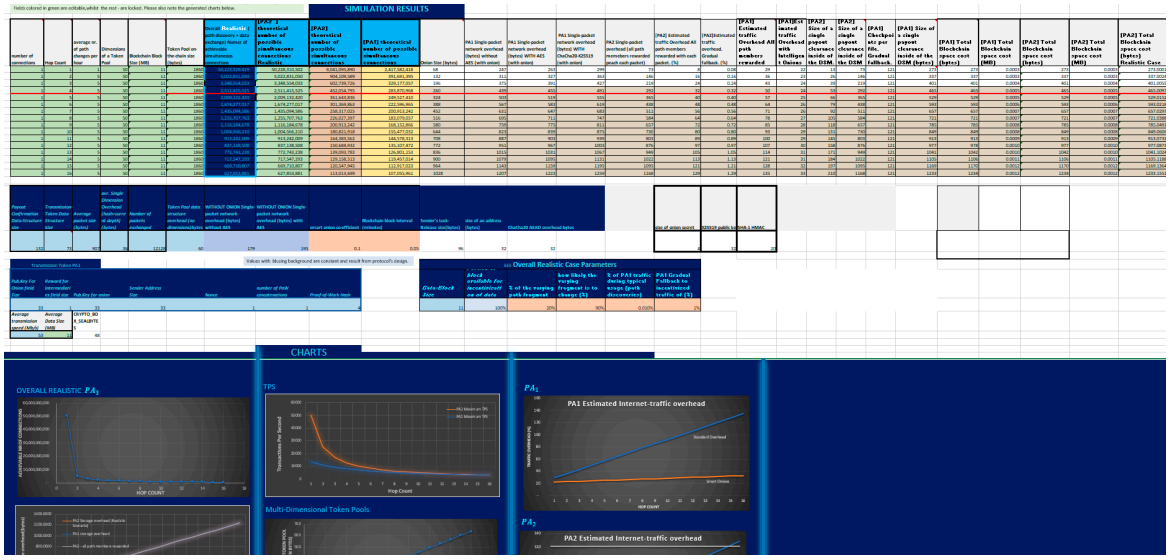

Figure 21. Fragment of a dedicated, Excel-based simulator of State-Channels and data incentivization performance under various parameters and conditions.

These are some fascinating results that could benefit not only the rigidness of data exchange within decentralized state machines but also extrinsic applications such as IoT protocols and services [16–18].

16. Future Work

We are to continue opening up our implementation to the outside world. As the reader is reading this, both the UI and Terminal Services have been long made available and are accessible through a public DNS-based gateway at test.gridnet.org. Should one choose to omit the public DNS system entirely, the list of IP nodes comprising the system is available over the Torrent protocol through a Magnet URL [19] (accessed 12 September 2022). In order for the described conceives to flourish even further, we demand better isolation constructs from web browsers' developers. The Shadow-DOM and the current implementation of web workers are great, but we shall continue extending upon and employing similar security and compartmentation constructs so as to aid the security of decentralized UI dApps. Web browsers are web browsers no more. The integration of verifiable off-the-chain computations on the side of web browsers could surely further improve upon the overall system's capabilities. It is worth noting that sound research results in the field of verifiable computation could enormously benefit the eco-system, further improving incentivization for computational tasks happening off the chain. We have been paying special attention to the problematics of incentivizing anonymous web browsing through crypto-incentivized reverse CORS proxies. Respected readers may already test these mechanics by launching the Browser UI dApp available on our test-bed, or by pointing their web browsers to https://test.gridnet.org:444/https://wikipedia.org or https://test.gridnet.org:444/https://google.com (accessed 12 September 2022). The result is, not a single cookie from a target web-server ends up on the client computer, all the privacy-related HTTP headers are stripped, and every kind of current CORS policy implemented in the current web browsers is effectively mitigated. Yet again, we strive to make these mechanics available at the fingertips of UI dApps' developers, all in the name of the freedom of information [20,21].

17. Summary

In this research paper, we have tackled the concept of a decentralized operating system, which stems back to the year 1987. The results we have arrived at with our implementation are very encouraging indeed. We have tackled lots of specificalities and demonstrated how we have approached various dilemmas as we went along. Notable contributions include the quantification of Decentralized Processing Threads, Multi-Dimensional Token Pools, and our implementation's specific Proof-of-Storge. The test-bed environment is available at test.gridnet.org, many tutorials are available at mag.gridnet.org, and we invite you to have a conversation at talk.gridnet.org. Above everything, we consider that it shall be our responsibility to allow for the information to be free. We believe that the concept of decentralized operating systems builds upon the paramount contributions of the research community and countless hours spent by developers, which are, as we believe, driven by passion, for the most of it.

Supplementary Materials: The following are available online at https://www.mdpi.com/article/10.3390/electronics11193004/s1, Excel S1: A Spreadsheet-based Simulator.

Author Contributions: Conceptualization, R.S.; methodology, R.S.; software, R.S.; validation, R.S., J.B.; formal analysis, R.S.; investigation, R.S.; resources, R.S.; data curation, R.S.; writing—original draft preparation, R.S.; writing—review and editing, R.S.; visualization, R.S.; supervision, J.B.; project administration, R.S.; funding acquisition, R.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data available within the attached spreadsheet-based simulator, other external data as per the references.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Skowroński, R.; Brzeziński, J. SPIDE: Sybil-proof, incentivized data exchange. *Clust. Comput.* **2022**, *25*, 2241–2270. [CrossRef]
2. Dannen, C. *Introducing Ethereum and Solidity*; Apress: Berkeley, CA, USA, 2017; Volume 1.
3. Nakamoto, S. Bitcoin Whitepaper. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 17 July 2019).
4. Goldreich, O.; Yair, O. Definitions and properties of zero-knowledge proof systems. *J. Cryptol.* **1994**, *7*, 1–32. [CrossRef]
5. Skowroński, R. The open blockchain-aided multi-agent symbiotic cyber–physical systems. *Future Gener. Comput. Syst.* **2019**, *94*, 430–443. [CrossRef]
6. de Ocáriz Borde, H.S. An Overview of Trees in Blockchain Technology: Merkle Trees and Merkle Patricia Tries. 2022. Available online: https://www.researchgate.net/publication/358740207_An_Overview_of_Trees_in_Blockchain_Technology_Merkle_Trees_and_Merkle_Patricia_Tries (accessed on 12 September 2022).
7. Wang, Q.; Li, R.; Wang, Q.; Chen, S. Non-fungible token (NFT): Overview, evaluation, opportunities and challenges. *arXiv* **2021**, arXiv:2105.07447.
8. Eyal, I.; Gencer, A.E.; Sirer, E.G.; Van Renesse, R. {Bitcoin-NG}: A Scalable Blockchain Protocol. In Proceedings of the 13th USENIX symposium on networked systems design and implementation (NSDI 16), Santa Clara, CA, USA, 16–18 March 2016.
9. Verdian, G.; Tasca, P.; Paterson, C.; Mondelli, G. Quant overledger whitepaper. *Release V0. 1 (Alpha)*, 31 January 2018.
10. Lee, S.; Kim, D.; Kim, D.; Son, S.; Kim, Y. Who Spent My {EOS}? On the ({In) Security} of Resource Management of {EOS. IO}. In Proceedings of the 13th USENIX Workshop on Offensive Technologies (WOOT 19), Santa Clara, CA, USA, 12–13 August 2019.
11. Moore, C.H.; Leach, G.C. *Forth–A Language for Interactive Computing*; Mohasco Industries Inc.: Amsterdam, The Netherlands, 1970.
12. Maymounkov, P.; Mazieres, D. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*; Springer: Berlin/Heidelberg, Germany, 2002.
13. Huang, Y.; Wang, H.; Wu, L.; Tyson, G.; Luo, X.; Zhang, R.; Liu, X.; Huang, G.; Jiang, X. Characterizing eosio blockchain. *arXiv* **2020**, arXiv:2002.05369.
14. Wang, C.; Li, Z.; Huang, X.; Zhang, P. Inferring the average as path length of the internet. In Proceedings of the 2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC), Beijing, China, 23–25 September 2016.
15. Center for Applied Internet Data Analysis. *Trace Statistics for CAIDA Passive OC48 and OC192 Traces*; CAIDA: La Jolla, CA, USA, 2018.
16. Buttyan, L.; Hubaux, J. Enforcing service availability in mobile ad-hoc WANs. In Proceedings of the IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHoc), Boston, MA, USA, 11 August 2000.
17. He, Q.; Wu, D.; Khosla, P. SORI: A Secure and Objective Reputation-based Incentive Scheme for Ad-hoc Networks. In Proceedings of the 2004 IEEE Wireless Communications and Networking Conference, Atlanta, GA, USA, 21–25 March 2004.
18. Buchegger, S.; Le Boudec, J.Y. Self-Policing Mobile Ad-Hoc Networks by Reputation Systems. *IEEE Commun. Mag.* **2005**, *43*, 101–107. [CrossRef]
19. Torrent Protocol Manget. Available online: https://www.magnet:?xt=urn:btih:62665C5C61976A3D7C30ED5B6B8486CFE285F9DB&dn=GRIDNET%20Endpoints.txt&tr=udp%3a%2f%2ftracker.openbittorrent.com%3a80%2fannounce&tr=udp%3a%2f%2ftracker.opentrackr.org%3a1337%2fannounce (accessed on 12 September 2022).
20. Braun, S.; Flaherty, A.; Gillum, J.; Apuzzo, M. *Secret to PRISM Program: Even Bigger Data Seizures*; Associated Press: New York, NY, USA, 2013; Retrieved June 18.
21. Gellman, B.; Poitras, L. US Intelligence Mining Data from Nine U.S. Internet Companies in Broad Secret Program. *The Washington Post*, 6 June 2013; Retrieved 15 June 2013.