IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 14 August 2022, accepted 13 October 2022, date of publication 25 October 2022, date of current version 31 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3216569

RESEARCH ARTICLE

Distributed Repository for Software Packages Using Blockchain

FELIPE Z. DA N. COSTA, RUY J. G. B. DE QUEIROZ, GUSTAVO P. BITTENCOURT, AND LEOPOLDO TEIXEIRA
Centro de Informática, Universidade Federal de Pernambuco, Recife 50740-560, Brazil
Corresponding author: Felipe Z. da N. Costa (felipe@zimmerle.org)

ABSTRACT A package repository is an essential piece of a software ecosystem. In FOSS, the software repositories are oftentimes hosted using limited donations, given the technical solutions adopted in the implementation. This work proposes a package repository using Blockchains with experiments and statistics based on a real-world scenario. The Blockchain described has its consensus algorithm crafted to befit the purpose of a package repository without financial appeal; Also, the proposed Blockchain keeps a compatible layer with the traditional repositories, easing its adoption. Furthermore, this work also presents a package search over peer-to-peer, computed on untrusted nodes, yet guaranteeing that the results are trusted. Finally, we present a functional Blockchain that cohesively exposes the PyPi catalog.

INDEX TERMS Blockchain, distributed consensus, repository.

I. INTRODUCTION

Modern software development is commonly developed on top of different reusable components, which are constantly evolving [1]. Those components are often distributed in the format of packages, described by stanzas of meta-information [2]. Meta-information is often interpreted by a Package Manager by creating a simple interface for the user to manage the *download*, *installation*, *update*, and *removal* of packages. Frequently, package managers also provide functionality for *searching* and *dependency solving* [3].

In Free and Open Source Software (FOSS), a package catalog is also known as repository. Repositories and the infrastructure needed to host them are typically maintained by a community. Hence, it is essential to count on community members for providing contributions in the form of donations, or by establishing partnerships to guarantee the healthy functioning of the repository. The lack of computing power may threaten some functionalities of those repositories, or even the repository itself. As an example, search has been disabled since 2020 on PyPI/PIP, the biggest software catalog for the Python programming language [4].

Distribution architecture for repositories ''in the wild'' [5], [6], [7], [8], [9] is either limited or nonexistent. Using mirrors [10] does not encourage minor or occasional contributors, thus limiting contributions by the few willing or able to share greater computing power. Facilitating contribution in terms of computing power has numerous advantages, including having peers close to each other, reducing network latency and allowing to create local cache via a Peer-to-Peer network [11], [12], [13]. The combination of both kinds of contributors, those with great power, as well as small or occasional contributors, can considerably increase the amount of computing power available for the repositories. As a consequence, enabling off-load processing on the network.

On open-source projects, it is expected that the community holds all the data [14]. Ideally, no central authority concentrates the power to control the repository. Instead, each contributor bears some responsibility, just as in a Blockchain Decentralized, distributed, and often public, a Blockchain consists of a sequence of blocks where a given block contains the cryptographic hash of the previews block. The calculation of the block hash considers the previous block hash. Therefore, validating one block implies on validating its predecessor, until the first block. In a Blockchain, the decision on what is published on each block is made upon Distributed Consensus. In a Peer-to-Peer network, the unknown and unauthenticated peers have to agree on the next block based on a set of simple rules.

The repository hosts metadata and source code for the packages. The metadata contains version information and the dependencies list. Layered security guarantees that an

The associate editor coordinating the review of this manuscript and approving it for publication was Taehong Kim.

exposed server or a malicious mirror will not temper a package going to the end-user. The current model used for the software repositories is well suited to handle the technical challenges associated with providing good security protection to the users. Therefore, the intention is to use the same model, keeping the Blockchain repository backward compatible with the security solutions adopted in the current models. Hence, easing the adoption of the Blockchain.

The initial challenge for the Blockchain-based solution was how to establish an agreement on a new block giving no financial incentives or connection with other financial-oriented Blockchains. The different distributions and maintainers are the most interested ones to keep the correctness of the Blockchain. Thus, it was natural to assume the popularity of the involved ones as a stake, where the forger became selected in a model created for this purpose. Initially, the popularity was set at a fixed pace with stakes. Nevertheless, the popularity can be further dynamic calculated as suggested at *Proof-of-Download*, demonstrated by Zimmerle and Queiroz on *A Blockchain using proof-of-download* [15]. Lastly, the block publication shall happen on time so that security software updates won't be delayed given missing block publication.

One of the advantages of adopting the Blockchain for the package repository is to preserve the history of every package. The repository tools force the most recent package version's installation (or update). Similarly to a regular package repository, a bad package - either poorly constructed or malicious crafted - can be superseded by a new package with incremented version. Atop the software version, distributions also count with the package versions, usually represented by a dash and a number, following the software version [16]. On a Blockchain, even if the software version is the same, the one represented in the topmost block prevails.

The Problem: Centralized repositories or with minimal decentralization narrow the list of potential contributors, therefore limiting the computing power of the repository and hedging user functionalities.

Regarding the state of our work:

Our Goal: Assisting anyone on the internet to share a small portion of computing power for a package repository. Ultimately, we assist in removing the figure of a central node by using a Blockchain.

A. CONTRIBUTIONS

Our work provides the following contributions: 1) *Blockchain*. Description of a Blockchain to support a software repository; 2) *Distributed Consensus*. A consensus-based approach for publishing the latest software versions; 3) *Forger Selection*. New algorithm for semi-random forger selection.

1) BLOCKCHAIN

Unlike financial-based Blockchains, whereas the incentive is merely financial, the Blockchain hereby presented considers the consistency of package publication based on the popularity of the stakeholders. In our Blockchain, every block holds a Merkle root; this provides the manners to check that a giving package version is held on a given block. Nevertheless, we also introduce the native support to exchanging packages over p2p network [11].

2) DISTRIBUTED CONSENSUS

Differently from other Blockchains, we present a method for an agreement based on stake that is ultimately connected to distribution popularity. Considering the FOSS model of *Web-Of-Trust* [17], we propose a thrust worth of the packages based on the previously agreed amount of Tokens. Detainers of those tokens can freely exchange them over the Blockchain. In a further version, the tokens can be calculated using the *Proof-of-Download* as demonstrated by Zimmerle and Queiroz on *A Blockchain using proof-of-download* [15].

3) FORGER SELECTION

The mutual agreement on the package publication is vocalized by the forger. The forger is semi-randomly selected given a group of possible forgers. We propose an algorithm for the group of forgers to randomly select one of them for block publication.

B. EVALUATION

Ultimately, validating the suggested Blockchain is checking the coherence of the published packages. The Blockchain needs to be coherent in terms of: (i) delay on package publishing; (ii) guaranteeing the consistency and integrity of each published package; (iii) no missing or unexpected package published.

The experiment performed in this work shows that latency was not significant. Often, packages were available to the end user even faster than the traditional mirror package publishing. The Blockchain package repository was shown to be consistent and resilient, not only to network instability but also to the presence of impostors.

II. BACKGROUND

Linux distributions heavily depend on package repositories to provide their users with a straightforward manner for installing and updating new software. The repositories are often configurable in terms of packages stability; they also allow the selection of an end-point server (mirror). The selection is usually automatically set to the geographically closest server in an attempt to reduce latency. Most distributions adopt the schema of mirrors and, thus, naturally impose a hierarchy on the distribution. Distributed architecture in the popular package repositories [5], [6], [7], [8], [9] is either limited or nonexistent. **Table 1** contains examples of popular FOSS repositories.

Application Stores: are a slightly different kind of software repositories [18], [19], [20]. In essence, all repositories are meant to provide software for their users. Different from the FOSS repositories, **application stores** provide the functionality for users to buy or subscribe to applications (or services) on their mobile phones, tablets, or computers. Such **application stores** are not in the scope of this work.

TABLE 1. Example of package repositories provided or used by Linux distributions and development communities. (a) Linux distributions; (b) Developement communities.

	Repo	Site
(a)	Ubuntu	https://help.ubuntu.com/community/Repositories
	ArchLinux	https://archlinux.org/packages/
(b)	Python	https://pypi.org/
	Ruby	https://rubygems.org/

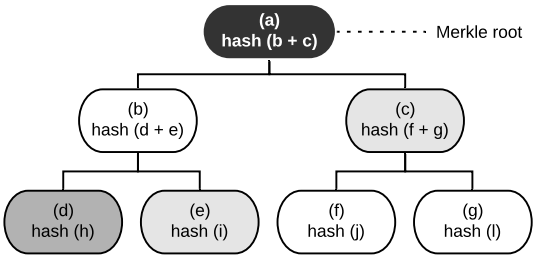


FIGURE 1. States a Merkle tree whereas the proof of information (d) is highlighted. One that presents: d + hash (i) + hash (c) and the Merkle root (a) can confirm that (d) is presented on such tree.

A. MERKLE TREE

A Merkle tree, or Hash tree, allows verifying the contents of large data structures efficiently and safely by using hashes as its kernel [21]. This reduces the size to store the verification that a given data belongs to a given structure. As illustrated on **Fig. 1** verifying whether a data belongs to a tree depends on proof; that proof is given by the hashes of some leaves on the tree and other important data, as follows [22].

1) The root hash of the Merkle tree;
2) The hash values to be verified;
3) The paths from the root to the nodes containing the values under consideration;
4) The hash values of branches not taken along those paths.

The proof size increases as the amount of information grows, making proof size $\mathcal{O}(log2n)$ where n is the number of elements stored on the tree.

B. InterPlanetary FILE SYSTEM

The InterPlanetary File System (IPFS) is a peer-to-peer hyper media protocol [23]. In a P2P network such as IPFS, if one node is down, other nodes in the network can serve needed files [24].

IPFS uses content addressing to identify content by the content itself, as oppose to where it is located. Every piece of content included in IPFS has a Content Identifier. The Content Identifier (CID) is merely the file representation in the format of a multibase string; The CID includes:

- **Multibase:** The encoding that was used on the CID string-encode representation of the original byte representation [25].
- **Multicodec:** Identifier for the codec used on the ID generation. [26]
- **Multihash:** Identifies the hash type and size used in the CID computation. [27]

By default, the IPFS uses the sha256 hashing algorithm, but virtually, any hashing algorithm can be used. Multiple hashing functions can co-exist on a single IPFS network. As important characteristics of content addressing, we can cite that (i) any difference in the content will produce a new CID; and (ii) the same content will produce the same CID, regardless of the computing node.

In order to download a file from an IPFS network, an user can be either part of the network as a node or to use a gateway. The gateways are implemented to create a compatible layer between IPFS and popular protocols such as HTTP, allowing anyone to fetch a file from an IPFS network using the HTTP protocol. Any node can present itself as a gateway [28].

C. BLOCKCHAIN

In addition to the predecessor's block hash, the blocks in a Blockchain also contains a payload. The **payload** is the data that is meaningful to the final application. The predecessor's hash is part of the block structure, also known as **metadata**.

As blocks contain the hash of its previous Block, the data in any given block cannot be changed unless the subsequent blocks are also changed. Although the blocks have a single parent, a block may have multiple children. Each of those child blocks refers to and contains the hash of the same previews block. This scenario happens on a **Blockchain fork** or **Soft fork**. The fork can be a consequence of two (or more) peers proposing a block in (almost) the same time. That is a temporary situation, as eventually new blocks will be generated atop of those, and the longest chain is considered to the best history [29].

1) NETWORK STRUCTURE

On a Bitcoin peer-to-peer network every node is treated equal. There is no hierarchy, and there are no centralized or master nodes. Every node on Bitcoin is an equal peer. The network has a random topology running over TCP, where nodes are randomly peered with other random nodes [30].

Every participant of the peer-to-peer is capable of validating any given block. The validation of the Blockchain is relatively inexpensive [31]. To be validated, the hash of the block in question needs to be computed. If the calculated hash matches with hash registered in the parent block, it is valid, and every subsequent block is consequently valid.

2) DISTRIBUTED CONSENSUS

The consensus mechanism needs to ensure that the decision upon the newest block is acceptable and fair to all legit nodes as it will define the truth of the Blockchain. This process of generating a new block is referred to as **mining**.

The mining process in Bitcoin use **Proof-of-Work (PoW)**, which consists in guaranteeing that the odds to be selected to craft the next block is given to the node that puts more ''work'' on it. The new block has to follow a set of basic rules; otherwise, it will be treated as invalid by the network.

TABLE 2. Block structure on the proposed blockchain. 225 bytes wide, the blocks are bigger than a normal blockchain block due to support to stake holders list and packages summary.

Size	Field Name	Description
1	Version	Version number
4	Timestamp	Creation timestamp
32	Merkle tree	Merkle tree root
32	Previous block	Hash for the previous block
46	Stake Holders	CID for latest stake holders list
46	Package Summary	CID for the latest package list
64	Block Signature	Ed25519 Forger's signature

Together with **PoW**, there are other methods of Proof, such as **Proof-of-Download (PoDl)**, and **Proof-of-Stake (PoS)**.

a: PROOF-OF-STAKE

In Blockchain terms, stake is what the user has and pledges to participate in the decision on the next Block. Unlike the name suggests, the consensus is not arbitrated exclusively by the one who holds more resources, but by a set of arbitration ''voters'' that decides. Thus, making sure that it is not centralized in one single peer.

In *PoS*, the miner of a new block is known as the forger. Before the selection, in order to participate in the selection party, the forger has to deposit some tokens into the network, using it as collateral to vouch for the Block.

b: PROOF-OF-DOWNLOAD

PoDl selects forgers by popularity. Assuming that forgers provide files, the ones with must downloads (most popular) will be recognized as forgers. PoDl securely establishes the number of downloads for later computing the popularity of the providers; the most popular providers are the ones eligible to be forgers. [15].

III. PROPOSAL: THE BLOCKCHAIN

Several Blockchain implementations already use IPFS as a method for distributed content [32], [33], [34], [35], [36]. The block payloads are somewhat limited in size, infeasible to hold larger information such as package meta-information (described at **Section II-C**) or the package sources. The payloads are usually limited to a hash, that is later used to compose the block Merkle Tree (**Section II-A**).

Using IPFS is natural, as its Peer-to-Peer (p2p) nature allows an easy setup and facilitates contributions in the format of bandwidth and storage space. As the network expands, it increases the chances of having a geographically close node, possibly reducing latency and increasing transmission speed. The setup-less network cache is easy to deploy, serving an entire local network without the need for setup on the clients. The files are served/reachable after its content (in the format of CIDs), therefore auto verifiable and easy to store on a Blockchain.

The file can be fetched from untrusted peers on the p2p network, as the verification can be performed using the file CIDs, stored on the Blockchain. Every block also holds CIDs pointers to digital documents, including the Package Summary. It is expected to have blocks of 225 bytes in total, containing *Block Version*, *Block Timestamp*, *Merkle root*, and the *Previous block hash*. Further details are shown in **Table 2**.

At the cost of fewer bytes, the package summary and stakeholders list are published on every block. This list can be easy computed giving the Blockchain history at any height. The computation comes at a cost, which may pose a difficulty for smaller or ad-hoc contributors. Providing such a list already computed speeds up the contributions, removing the barrier of downloading the blocks information and indexing the package summary.

A. THE PACKAGE SUMMARY

The package summary is a Comma-Separated Values (CSV) [37] list, whereas all the packages already published on the Blockchain are described in its latest version. In the package summary document, it is expected to be encountered: package version, package description, package publication date, and block prove for every package. That information is later used for peers that want to search packages or provide the search functionality. The package summary file also makes it possible to look up the latest package versions.

As the package summary file is available in a p2p manner, via IPFS, the clients may opt to provide the list partially to whatever size fits on the amount of computer power that they are willing to share. Likewise, the local search can be made in a stream fashion, as the download goes. Therefore, limiting the number of resources used to process a file whose size is arbitrary.

B. THE STAKE HOLDER SUMMARY

The validation of stakeholders (possible block forgers) can be made easily with the utilization of the stake holder file. The file contains information on the stakes of every forger. Allowing the easy look up, by any peer who may concern about the authenticity of a block. With such a list, the clients does not need to compute the entire Blockchain in order to identify possible publishers.

The stake holder summary is a CSV list containing the public key of the node, the amount of stakes that it may has at the given height, and the proof of the stake transaction. Oppose to the Package summary, this file does not meant to grow indefinitely. But, at the control of the stake. The file can grow or shrink upon on the amount of possible forgers.

C. FILE AND METADATA STORAGE

All package metadata, as well as source files, are meant to be distributed over IPFS. As the CID is a representation of the data, it automatically suits as self-validation of the file integrity. The self-validation property makes it possible to retrieve files from public sources yet guarantees their integrity. The CIDs for the metadata are published on the blocks.

Critical infrastructure in server rooms or public internet, where restrictions are imposed on internet access, may rely on IPFS web gateways (**Section II-B**) for download packages and meta-information. The utilization of web gateways is

TABLE 3. Resource provide identifications. (a) Package search. (b) Blockchain update.

-	CID
(a)	QmTp9VkYvnHyrqKQuFPiuZkiX9gPcqj6x5LJ1rmWuSySfU
(b)	QmWoWUjG717ARJDqyxZttVnnh3nmWcFJgXtwHLZifkZi87

discouraged, as it implies not taking full advantage of p2p benefits.

Guarantee the availability of the files on the IPFS network is critical to avoid dangling packages. It is expected multiple IPFS servers hosting the same file. Either because a node used the file and kept a copy or because the node has the file downloaded to serve others. Mirrors are easy to be deployed as a new IPFS node. Different mirror strategies can be adopted, for instance, only mirroring the most recent packages instead of mirroring the entire Blockchain data.

The package publication depends on simple rules to guarantee the file's existence while the block is being forged. In the event of a package publication, it is expected that the publisher has the package available on an IPFS node. If available, the package will also be provided by the block publisher, making sure to have at least two nodes on the IPFS providing the package. Without having the package files (source or meta-information), the forger will discard the package. If any block got to be published without the associated files, it is automatically considered invalid.

In a software life-cycle, it is expected that the package will become deprecated. The files for deprecated packages may not be distributed or available on the IPFS. Deprecated packages may imply security risks, so not serving those could benefit the users. [38].

1) THE DEPRECATED PACKAGE

Different metrics could be used to understand the package deprecation. The deprecation comes as discretion at the distribution that vouches for the package. It is possible to layout package depreciation metrics on a Blockchain. Having the blocks being tentatively published on an equaled spaced time allow us to understand the publication time based on the tip of the Blockchain, therefore having the block publish age. One option for distributions is the re-publication of the package (with the same version) at certain fixed age (or block interval). If the distribution does not re-publish old packages at a given time, automatically consider those as deprecated. Likewise, packages that depend on deprecated ones shall be treated as deprecated.

2) THE MISSING PACKAGE

Network failure and server outage are two of the many reasons why a package may be unavailable. The availability is somewhat proportional to the number of mirrors. Thus, the easy way to avoid the missing packages is to at least provide one copy of the file on a reliable server. One interested in providing such a mirror server may have to walk over the digital ledger, downloading every single file to its instance of the IPFS node. While constructing the mirror, it is also possible to identify the already missing files. The distribution may want to ponder walking the ledger, having as a metric the availability of its files considering external factors such as the geographical position of the potential users.

TABLE 4. Different types of nodes on the suggested blockchain.

	Node Type		resources to share CPU	Disk Space	Bandwidth
(i)	Supply	Search	Heavily	None	Minimal
(ii)	Supply	Storage	Minimal	Heavily	Heavily
(iii)	Supply	Blockchain	Moderate	Moderate	Moderate
(iv)	Client		-	-	-
(v)	Forger		-	-	-

D. PEER-TO-PEER NETWORK

Considering the goal of having an autonomous Blockchain, and so, avoiding any centerpiece on the communication, the p2p was a natural choice. Notice that the nodes communication happens in different network from the IPFS. Having the files distributed over a common IPFS comes with the benefit to count with the already deploy infrastructure, holding about 2 (two) million of new users per week [39].

One of the challenges in a p2p network is ensuring that all nodes are part of the same network. Nodes can find each other on a local area network, benefiting from low latency and fast local area network transfer without overusing the Internet. Yet, the internet node lookup is vital to ensure network integration globally. The node finder method may vary depending on the characteristics of the node's network.

For the local area network node finding, the mDNS [40] was chosen, as it allows nodes to find each other with zero configuration by using a multi-cast system of the DNS records. For the Internet node finder, the bootstrap allows nodes to find each other by querying pre-fixed nodes; once connected to these hard-coded nodes, the local nodes tables are filled with other connected nodes [41]. As soon as the node become part of the network, all the resources (and nodes) lookup are made through a protocol based on Kademlia DHT [42].

Every node on the p2p network has its ID; that ID is a correspondent to its public key. So, if one node wants to establish communication with the other node, they can find each other via its ID. Likewise, the look up could be performed on a resource ID (**Table 3**). The node private key is also use to sign the packages and blocks, validating the origin of the block.

The network contains different types of nodes. All is meant for different functionality. It is worth distinguishing incentives to the sharing of different resource types: *CPU*, *Disk Space* or *Bandwidth*. **Table 4** presents the different types of nodes and the expected resource to be shared.

The forger node (**Table 4-v**) is a special node meant to forge new blocks. The block forger needs to perform a block signature using a key in which the privilege for signing blocks was given.

The client node (**Table 4-iv**) does not cooperate with the network as it may be running in a device with limited resources (such as an IoT device). The client nodes only consume resources from the network.

Implementation wise, the contributors or supply nodes can support the three types of contribution (or any combination of **Table 4-i to iii**) in the same node. Some limited resource

contributors, such as the search node (**Table 4-i**), are minimalist enough to run in a browser.

The p2p network is meant to have two different resources: **A. Package search**; **B. Blockchain update**. Listed on **Table 3**.

E. BLOCKCHAIN

Releasing a new block on every two hours, we suggest having a Blockchain with blocks of 225 bytes, containing *Block Version*, *Block Timestamp*, *Merkle root*, and the *Previous block hash*, as detailed in **Table 2**.

Although the download size is 225 bytes, after verification, the information is saved in 32 bytes. Only the block hash is needed to be saved. By keeping all blocks hash in a file, the offset will denote the block number (1). Therefore, the validation data is proportional to the size of the chain in blocks. The block can always be downloaded and later compared to the saved hash. Keeping the hash for elderly blocks may not be necessary as those may hold outdated or discontinued software.

$$f(x) = x \times 32 \tag{1}$$

where:

$$f(x) = \text{Offset for the Trusted Block Hash}$$
$$x = \text{Block height}$$

F. BLOCKCHAIN UPDATE

All nodes that depend on an up-to-date version of the Blockchain shall update as frequently as 2 hours. The updates could either be via the p2p network or via WEB. The updates are expected to happen with the download of the entire blocks. The nodes entitled to mirror the Blockchain (**Table 4-iii**) would also provide the real information on every package published on the Merkle tree, not only the hash. Over and above that, peers can listen to the block publication broadcast, keeping a live updated version of the ledger.

The p2p mirror nodes are meant to provide the updates per block; The block height can be used to identify blocks. It is possible to have more than one valid block at a given height. Alternatively, blocks can be retrieved by their unique identifier: the block hash. That block hash can be later validated at the ledger's largest chain. Additionally, mirrors can validate the integrity of each block and all the provided information.

Updates on the package's source files (or metadata) are only necessary when packages need to be installed or updated. Mirrors that may want to host packages source files and metadata (**Table 4-ii**) can keep an IPFS copy of every file on the chain.

In practice, the package manager will update the tree on every operation, leading the user to know if packages are available for update. The chain update should be faster if frequently updated. Otherwise, cumulative updates may delay the update process.

G. PACKAGE SEARCH

As the name suggests, the main goal of the functionality is to provide a search under every available package on the Blockchain. The lookup is meant to happen over the **package name** and **package description**. To perform a search, the client first have to identify the nodes that provides the search functionality: over the p2p network, the client has to lookup the "search resource" (**Table 3**) using the Kademlia DHT. Once identified, the client will place a search request for the selected search node (**Table 4-i**). The search supports limited regular expressions patterns. Limitation aimed at avoiding ReDoS attacks [43].

As a return of every search result, the search node provides name, version, description, and block proof for every match. The proof establishes that the given information belongs to a given block. The proof contains all the hashes required to establish whether a given data belongs to a **Merkle root**. By downloading the block, the user can confirm that it is trustfully by comparing its hash with the local register; furthermore, the Merkle root is used to validate that the information obtained is part of the block.

SEARCH as Web API: Via p2p gateway or local database, websites can provide the search functionality. Regardless of the used method the proof for every record are necessary to be presented, providing the user who have requested the search a receipt for later validation. The web interface is specially necessary for infrastructures where the internet access is limited. The p2p gateway can present itself for the network as a regular node, performing the search and giving back the search results to the end user. The local database can be either downloaded giving the summary list on the last block or computed giving the Blockchain history.

Local Search: Nodes whereas storage size and processing power are not limited, the user could can keep the last summary list saved, updating as new packages are released. Within the saved list the users could perform search in its own local database. Depending on the CPU power availability it is expected to have faster results within that method.

Building the Search List: Any peer that is willing to cooperate with the network by providing the search functionality (**Table 4-i**) has to either build or download its search table. Computing the search table means indexing the Blockchain by package, keeping the latest version of all packages and their respective name, description, and version.

To facilitate such computation, a summary list is also published on every block via IPFS allowing a node to retrieve latest package information without needing to compute the packages on the entire Blockchain. When downloaded from doubtful peer, the list hash can be validated against the CID published on the last block.

1) SEARCH RESULT TREAT MODEL

The treat model considers the search to prevent bogus results from being presented. For that, the user participation is needed:

- The user shall validate that the search keyword is part of the package description or name.
- The user must validate that the package information was published on the given Block.

If such criteria are not met, the user shall disregard the search result. Furthermore, the user shall consider the latest package's version in the package summary file on the tip block. It is up to the user whether to install a package that is not the most recent version of a given package. The concealment of package information may be at the user's perception via the different responses on the network. Ultimately, the client may consider the results for multiple searches.

The negative response while searching for network nodes that provide the search functionality could be due to the nonexistence of search nodes (**Table 4-i**). The former may indicate the deprecation of the network.

H. DISTRIBUTED CONSENSUS

There are a few nodes that could issue a block, and those are the ones most interested in having a consistent network: the block forger (**Table 4-v**). Since the network is made out of nodes that do not necessarily trust each other, it must agree on the block publications and their contents. The consensus is obtained via a set of pre-established rules that happens over the p2p network.

Forger nodes are nodes that proves the possession of a private key in which the correspondent public key has granted the permission to issue blocks. The permission is given in the format of tokens. Tokens can be interchanged among the peers; every token transaction is also held on the Blockchain.

At the first block, 1,000 tokens are granted to the first forger, and later the forger distributes the tokens to other forgers at convenience. Blocks that are published on the network without a valid signature from a valid forger shall be disregarded. For a block to be considered valid, it has to follow a minimal set of rules:

- The forger signature on the block must be valid.
- The forger must present at least one token.
- All package publication signatures must be valid.
  - All publishers have to be validated.
- Files needs to be reached over the IPFS network. The forger must provide a copy.
- Transferred tokens must be valid.
- Summary for packages and tokens have to be valid accordingly to the Blockchain history.

Blocks that do not meet one of the above-mentioned items are automatically regarded as invalid and should not be trusted by the users. Since all validation data are public, block validation can be held by any peer on the network, including other forgers.

Since it is expected to have more than one forger interested in publishing a block, there is a criterion in which a forger is semi-randomly elected to forge a block. This computation follows an agreement labeled: The forger party.

1) THE FORGER PARTY

Every 118 minutes, one of the forgers initiates the forger party. The forger party is the process where a forger gets selected to forger a block. The party is divided into phases: **Participation announcement**, **Ticket revealing**, **Block publication**. All those phases are due to its time-out. The party takes no longer than two and a half minutes.

TABLE 5. Example of a party-table information, sorted by rand-I. Notice this table is held by each participant in the party while electing the forger for a block.

#	name	ts	rand-I	rand-II	hash
1	Cubert	1608216371	10	19	0ebe27ebb...d6690bad7
2	Nibbler	1608216334	15	1	f89dc13a2...b0e94020b
3	Heber	1608216340	40	100	80cbbf7b6...5db692084

a: PARTICIPATION ANNOUNCEMENT

Participation in the forger party is not mandatory. If the forger happens to be interested in forging a block, it has to announce such interest.

The interest announcement is combined with a payload that is later used to select the forger node. This payload is a sha256 hash that is a representation of the concatenation of the message timestamp and two 32-bits numbers picked at random (2), so-called Ticket.

$$\text{Ticket} = \text{sha256}(\text{timestamp} + \text{rand}_{\text{I}} + \text{rand}_{\text{II}}) \tag{2}$$

This computed hash is later signed (using the forger key) and published on the p2p network for all the listeners. Meanwhile, the forger also listens for the other nodes' publications and saves each publication in a table. If the signature does not match or is not given by a valid forger, the publication will be automatically discarded.

The Participation announcement takes no more than 60 seconds. After that, any new message will be discarded. Likewise, messages published prior to 118 minutes after the publication of the latest block will also be discarded.

b: TICKET REVEALING

Each forger reveals the Ticket over the p2p network one minute after the hash publication. Every other saves this Ticket information on the table. The Ticket information is validated; if there is a mismatch between the Ticket and the publish hash, the information is discarded. In this step, the timestamp will also be validated; it must be befitting to the message publication.

Once the tables are filled, every forger has a similar table. The first random number must sort the table; lower numbers first. By so, every forger (with access to the same announcements) has a common position on the table.

After sorting the table (**Table 5**), a XOR operation takes place, considering all of the random generated numbers. The result of this operation leads to a *seed number*. The remainder of the division of the *seed number* by the number of forgers in the party is the elected node, considering its position on the table.

When two (or more nodes) publish the same random number, the second random number can be used to untie the selection. Suppose the second number also happens to be equal. In that case, the untie is given to the node where the name is lowest, considering its name's ASCII characters.

Since this forger party happens in plain sight, every other node can validate. If a node considers a different list of

forgers, a diverging tree is expected. One tree will be chosen on the next block forger party. The tip common to more forgers will prevail.

c: BLOCK PUBLICATION

Once the node becomes aware of its selection, the announcement of the new block is expected on the p2p network.

If an impostor forger becomes selected, it won't publish invalid data as the block will be invalid. On the occasion of never publishing the block, the chain will be fixed on the next forger party, whereas a valid forger is likely to be picked within 2 hours.

IV. EXPERIMENT

The experiment hereby presented consists of adding a set of nodes acting on the construction (bootstrap) and maintenance/update of the Blockchain, parodying the packages distributed on the PyPi repository. The experiment also counts with the figure of the impostor nodes. The impostors have no other goal other than to play as adversaries, making attempts to subvert the Blockchain. The Blockchain uses as consensus the proposal suggested on **Section III-H** as well as the Blockchain characteristics described in **Section III-E**.

At the very least, it is expected that the Experimental Blockchain will have all the packages published on the PyPi repository, in addition to any other package added by the publisher, straight on the Blockchain.

The Blockchain simulation was planned to count on having twelve different nodes. Those nodes fell under categories depending on their responsibility on the Blockchain. The categories are described below.

- **Whistleblower**: The main responsibility of the whistleblower is to announce on the Blockchain new packages publications. Either by going over the publication history of the PyPi repository or by tracking new publications. ⇒ Node type defined in **Table 4-iv**.
- **Forger**: The forger nodes are the ones with stakes to publish Blocks. Those will compete on the forger Party (**Section III-H1**) to release new Blocks. ⇒ Node type defined in **Table 4-v**.
- **Impostor**: Adversary nodes that are trying to subvert the Blockchain in different manners. Detailed in **Table 7**. ⇒ Node type defined in **Table 4-iv**.
- **Listen**: The Listen nodes are listening the peer-to-peer network, observing packages/block publication to feed their own database. That database will be later used for verifying the packages availability consistence. ⇒ Node type defined in **Table 4-i,ii,iii and iv** .

Each category had a different number of living nodes: four nodes for whistleblower, three nodes for forgers, tree impostors, and finally 2 listing nodes. The nodes were distributed in different geographical locations to simulate a real-world scenario where network latency pays a difference. The nodes compete with each other on the package and block publication. Intending to generating readable statistics, the nodes have received nicknames, as illustrated on **Table 6**. Also on

TABLE 6. Details on each of the nodes used on the blockchain experiment.

#	Name	Region	Description
Whistleb.	Amy	UK-I	Publish new packages
	Hermes	US	
	Kif	Canada	
	Leela	UK	
Forgers	Cubert	UK	Blocks Forgers
	Nibbler	US	
	Heber	Canada	
Impostor	Zapp	UK	Submit packages with different authorship
	Zoidberg	US	Submit packages with wrong signature
	Mon	Canada	Delay secure updates
List.	Scruffy	UK	Watch and keep package publication
	Morbo	Canada	

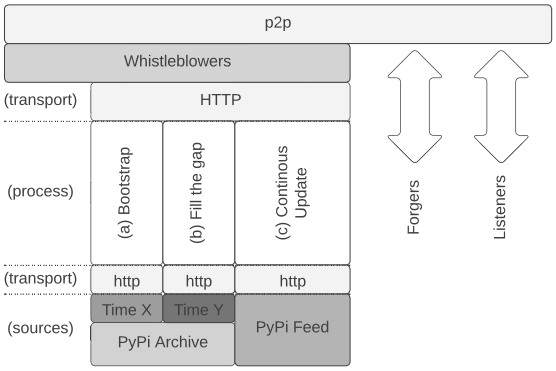


FIGURE 2. Pipeline for feeding the Blockchain with different processes: (a) bootstrap, (b) sync process, and (c) Continuous update.

Table 6 the geographical position of the nodes are revealed. The hardware used in the experiments were Linux virtual machines on a cloud service running an AMD EPYC with 4GB of dedicated RAM for each virtual server. [44].

The whistleblower nodes trust each other. For every package released, they also name the others tree whistleblowers as trusted in that package publication. That way they may compete with each other to figure each of the four whistleblower will publish the next package update.

Under the simulation the stakes are divided equally in between the tree forgers: 333 for each forger. Nibbler is the forger for the genesis block. The subsequent blocks where product of the forger Party exactly how it is described on **Section III-H1**. The source for construction and test of the Blockchain came from two distinct places: the package history and the continuous updates.

A. FEEDING THE BLOCKCHAIN

The mechanism to feed the Blockchain was based on a producer-consumer pipe whereas items got to be removed from the pipe, whenever confirmed to be processed by a node. The consumers are the network nodes (whistleblowers), listening to HTTP requests. The forger considers the package information that came with the first whistleblower over p2p, naturally, packages are validated before got accepted.

The produces are divided into tree different groups based on the source of information. The first group is the one that was enumerating the data from the PyPi history (bootstrap). The second, was considering the continuous update of the packages. Finally a third process fill the gap for the interval

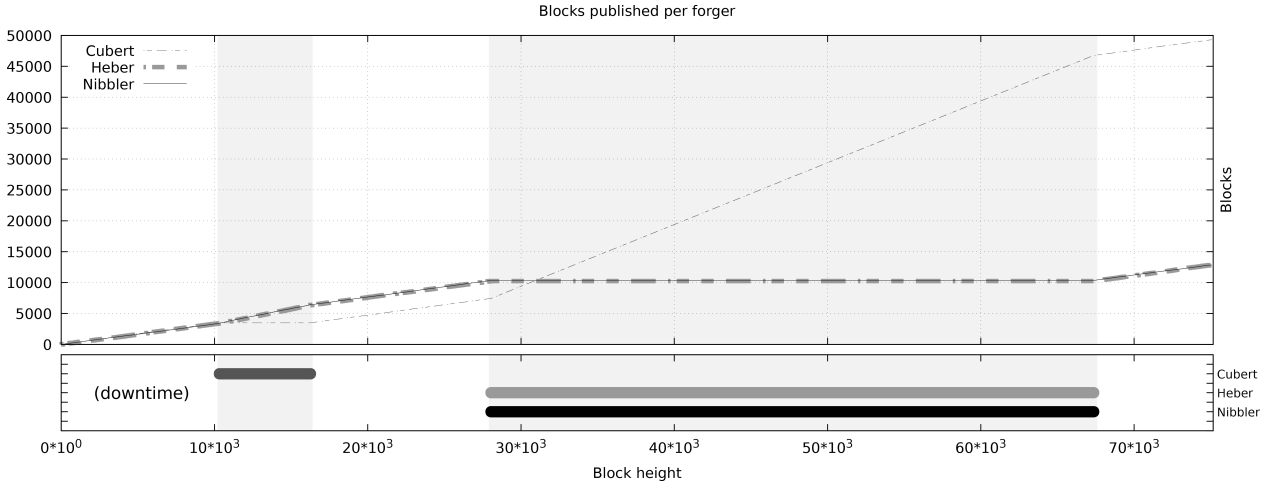


FIGURE 3. Number of times that a forger got selected in the forger party versus the block height. Nodes downtime in highlight.

in between the PyPi snap shoot to the moment that the Blockchain started to get the continuous update. The **Fig. 2** demonstrate the dispositions for the produce and consumers in this Blockchain feeding process.

1) BOOTSTRAP: FEEDING THE BLOCKCHAIN WITH PyPi DATA

All the data from PyPi repository are available on their WEB API. However, avoiding abuse on servers that are maintained for free, there are some restriction on the amount of parallel downloads, so as download frequency. Considering the limitation and volume of data, to scrap all data was a challenging.

The PyPi API counts with a package index that was used to indicate every package that was ever published on PyPi. The repository scrappy counted a process to download all the packages information, making a JSON files for every package. A bit of normalization was used in order to save packages with names that could fit in the used file system (ext4) [45]. This process was gentle and counted with 4 different threads, having a total processing time of 25 minutes and 12 seconds to download over 6.2G of data finishing the snapshot at May 06, 2022 14:10:19 GMT-4

As the packages were already hosted in JSON format, no normalization was needed to be made on the package data. However, the release date for each package was used to organized the data into a Blockchain structure. At first, the script identified the oldest package/release among the downloaded packages. The oldest package timestamp (zconfig / Mar 21, 2005 15:59:25) was the timestamp for creating the genesis block. In turn, for every package release, the timestamp was computed to a block, respecting 2 hours between each block creation. The script appended the package release information on a JSON file named after the block number.

Notice that the same metadata from PyPi was kept on the Blockchain, in additional references for the files CIDs were added. As such, the dependency tracker used in PyPi is still valid and backward compatible with the current PyPi.

Concerning that all package data was aligned with their respective blocks, a different process was used to broadcast the information on to the whistleblowers. The whistleblowers were collecting the received package information and publishing as new packages for the attention of the forgers. Consequently, creating a snapshot of the PyPi database at a given time. Yet, it was still necessary to continue updating the Blockchain with newer package publications outside the package history. The block publishing interval was reduced from 2 hours to about 4 seconds per block during the simulation. On average, it took about 4 seconds per block, finishing the simulation in 3 days and 12 hours of execution. A total of 75062 blocks were used in the experiment.

2) CONTINUING UPDATE OF THE PACKAGES RELEASES

Although the method bootstrapping the Blockchain was shown to be effective, it was somewhat abusive. As the resources for PyPi are limited, the continuous download of the entire PyPi database would be unfair. More elegantly, a continuous process spotting only the new updates would be a good fit. That is exactly what is provided by PyPi updates RSS feed [46]. Available at `https://pypi.org/rss/updates.xml` the PyPi update feed contains the most recent released packages.

Every two minutes a script performs a download of PyPi's update feed and sent to the different whistleblowers in the same way that was done while the Blockchain was being bootstrapped. Those produce nodes are known by the whistleblower that once authenticate the data will proceed with the package announcement/broadcast to the forgers.

If a package is claimed - published by the author straight on the Blockchain - the whistleblower will not be able to update it unless the whistleblower is also marked as publisher by the original publisher. Packages that are already claimed can only be published by whoever claims them or happens to be authorized by the original claimer. The network may count on various whistleblowers; it is alright for the whistleblowers to compete with each other on the package publishing. The block forger should avoid any conflict by picking the first package announced.

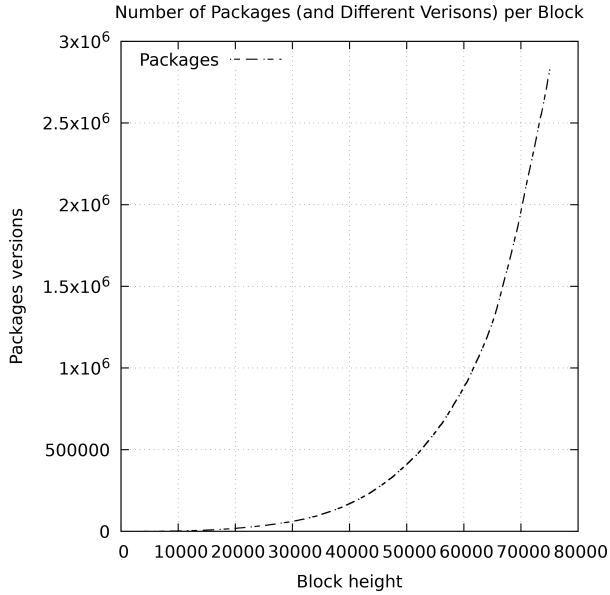


FIGURE 4. Number of packages and packages new versions published at a given block.

There is no guarantee on the update frequency of the PyPi feed. Therefore, a package may be published on PyPi but not published immediately on the Blockchain. Furthermore there is a gap in time between the time that the repository snapshot was taken and the time that the Blockchain was ready to recive updates. That gap is filled by a sync process.

3) SYNC: MISSING PACKAGE VERIFICATION

The verification to check if a package is missing on the Blockchain consists of downloading the packages from PyPi and comparing whether they are present or not on the Blockchain. If not present, the packages are added via whispleblower altogether with the packages for the next block. In this case, it is expected a delay. The package will only appear in a later block, therefore being late to be available to the user. The best effort is used to add the packages to the Blockchain. It is guaranteed that every package will be published in the long run. Eventually, packages may not be published on the next block immediately.

This third process runs with an extended interval to avoid draining the resources from PyPi servers. Hence the need for the three different processes: bootstrap, continuing updates and verification.

B. THE WHISTLEBLOWER

Apart from listen the network for updates, the whistleblower is also responsible for parsing the package JSON, downloading every package file and provide a copy on an IPFS node. Once the CID of the files are generated, the whistleblower adds an extra entry to JSON, mapping the URI with the CID content. In way that the user was provided with both options: regular download, or download via IPFS whenever available. IPFS was made optional in order to reduce the amount of space necessary for storage all the packages files during the simulation.

TABLE 7. The tree adversary nodes used in the blockchain simulation followed by each attack description.

Node	Description
Zapp	Re-assemble package publication having its own name as authorship. The signature matches, but, the whistleblower must prevail.
Zoidberg	Publishes valid packages but posing as an whisteblower. Signature will not match.
Mon	Flood publication of old packages, distracting the forger.

The whistleblower must validate that a package information is coming from a trusted origin. If information is not authenticated or corrupted, it most be discarded. Likewise, every package publication sent by the whistleblower shall be digitally signed, otherwise discarded by forgers.

C. THE LISTENING NODES

Listing to blocks publication the listen nodes are the ones responsible for creating a parallel database contains the information on all the packages. This information will be later used to compare the data published on the PyPi network to data that we have published on the Blockchain. Additionally the listen nodes will also be providing search functionality and Blockchain mirror on the p2p network **Table 4-i,ii,iii and iv**.

Important to note that the listen nodes are not aware the Blockchain construction stage. They meant to be executed in the beginning of the Blockchain creation process and kept alive till the end of the simulation. To facilitate the statistics generation those nodes are creating their own local database. The database will be later used to understanding the package publication timing.

D. BAD ACTORS: THE ADVERSARY NODES

The treat model is hereby defined by tree different actors: Zapp, Zoidberg and Mon. Those three nodes are responsible for bad behaving in different manners, as described in **Table 7**.

Every 10 seconds, the node Zapp retrieved the first package published on the p2p network, re-branding the signature as he has published the package (regardless of the original publisher). Zapp is not trusted as the publisher of any package nor recognized as a whistleblower. In the same 10 seconds, Zoidberg was also publishing a package with an invalid signature. In the Zoidberg attack, the original author's signature was changed in the first byte, leading to a corrupt signature.

Mon's attack was the most relevant, as the node was publishing old packages (at most 100 randomly selected packages from previous blocks) with valid signatures, leading the forger to look up if the packages had already been published.

V. RESULTS

The presence of the impostor nodes was recognized as a great value in the development of the experiment, as it surfaced implementation errors otherwise not noticed. The impostors helped to understand that the signature validation made at the package announcement arrival is less memory intensive.

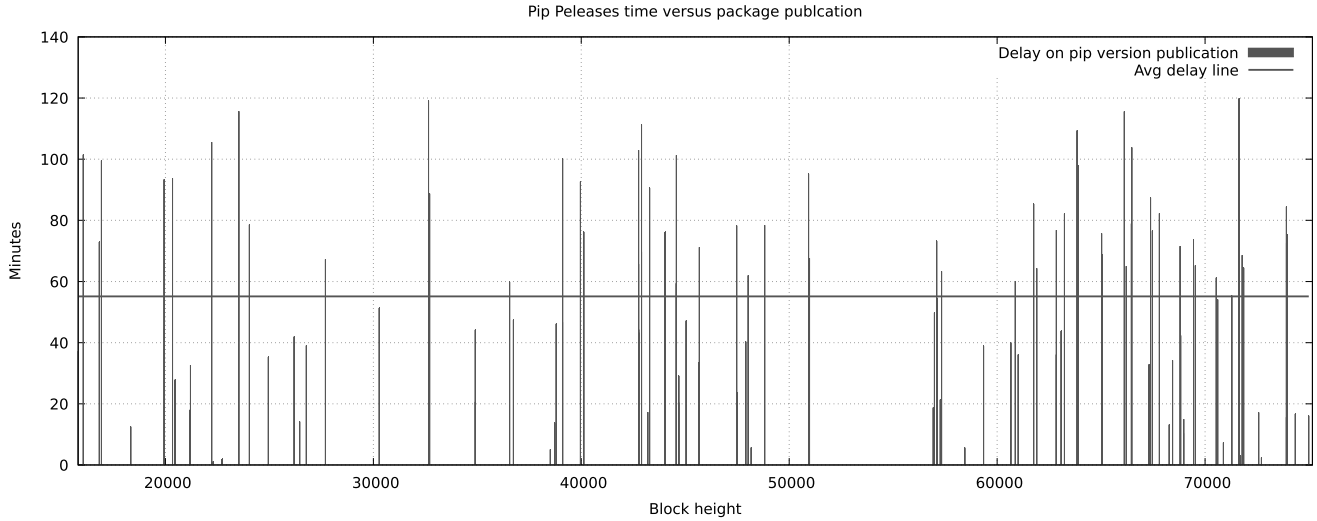


FIGURE 5. Amount of new packages published per whistleblower.

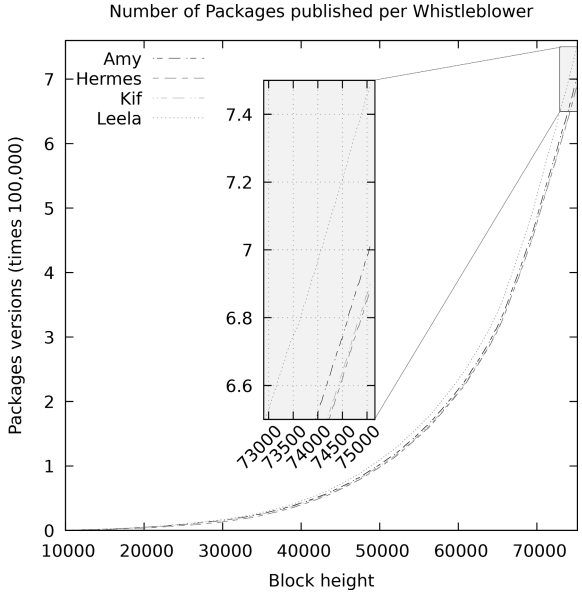


FIGURE 6. Delay on Pip releases in relation to the correspondent block publishing. Average time is highlighted on the dashed line.

Thus, making the forger nodes less prone to DoS attacks for invalid signatures or ''fake whistleblowers''.

The forger implementation had to be changed several times for package lookup optimization. In the end, a parallel GDBM[1] database was used, as Mon attacks were leading to CPU spikes before the GDBM adoption.

The problems that became evident in the simulation it is unlike to happens in a real-world scenario, the 4 seconds per second on simulation leads the processing on the simulator nodes to be 100% per almost all the simulation, something that would happen is sparse intervals such as 2 hours per block. The cited problems would not represent a treat.

[1]GNU dbm (or GDBM, for short) is a library of database functions that use extensible hashing and work similarly to the standard UNIX dbm. These routines are provided to a programmer needing to create and manipulate a hashed database.

TABLE 8. Number of packages published by each whistleblower (or bad actor).

Amy	Hermes	Kif	Leela	Zapp	Total
701026	688197	689855	748870	0	2827948
24.79%	24.34%	24.39%	26.48%	0%	

Nevertheless, apart from the possible collateral damage, the attacks proved to fail in their main objective. As shown on **Table 8**, the node Zapp did not managed to publish any package. Also, no package was observed to be published with an invalid signature. Ultimately, Mon's attack did not manage to fake any package version.

The semi-random selection for block forgers proved to be even, as the distribution seems not to favor any particular node, as demonstrated on **Fig. 3**. Network issues and power outages led to downtime for the nodes, Heber and Nibbler. As expected, Heber and Nibbler's downtime did not affect the well functioning of the Blockchain; Cubert was publishing blocks without damage.

Finally, the exact number of packages were encountered in The PyPI repository and Blockchain. There aren't divergences in the package publications. There were 282,794,8 packages (or new package versions) published on the snapshot taken on May 24, 2022. The **Fig. 4** illustrates the growth of the package publication on PyPi, consequently, the growth of packages being published on the Blockchain.

The competition among the whistleblowers did not seem to favor any particular node, as all nodes seem to have a very similar amount of packages published. In the **Fig. 5** it is possible to understand how close is the amount of package publication among the four different whistleblowers. Towards the end of the Blockchain, Leela has a small discrepancy in packages published. The difference seems to increase after block 70,000, where Leela had less latency to reach forgers Herbe and Nibbler. Yet, the difference is negligible given the number of published packages.

The package **pip** was chosen as an example to understand the amount of time in between the package publication (by the

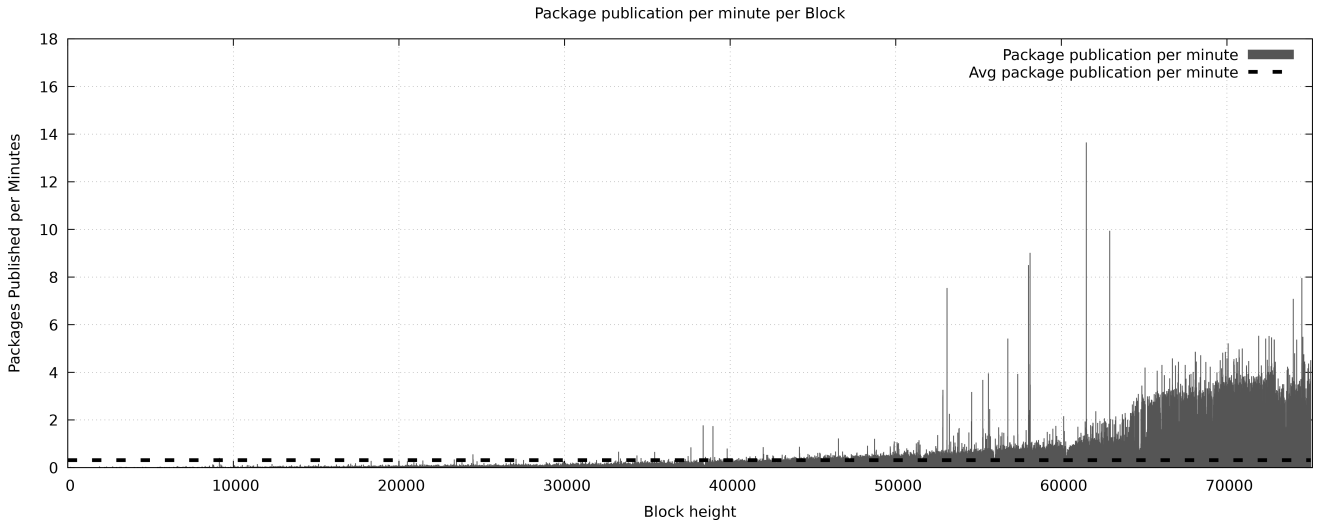


FIGURE 7. Average package publication per minute on every block.

whistleblower) and the package availability to the final user. The Blockchain associates the package availability to the block publication (approximately every 2 hours). The **Fig. 6** illustrates all **pip** releases together with the delay between the package and the block publication. Notice the average delay of 55 minutes. An excellent time compared with the mirrors practice where servers are ''synced'' during the night for easy network traffic.

The Transactions Per Second (TPS) usually seems to be a critical factor on every Blockchain. In this experiment, the TPS was considered low. In **Fig. 7** illustrates the average package publication per minute on every block. The highest amount of publication was given at block 61507, where on average, 13.65 packages were published per minute.

VI. CONCLUSION AND FURTHER WORK

The Blockchain proved to hold the same packages also published by PyPi, even counting on network elements and the presence of impostors. As most packages were added during the simulation, the experiment was not significant enough to show the package publication delay.

The forgers seem to be evenly selected during the Blockchain construction, proving that the selection method is working effectively. Therefore, having an honestly distributed consensus. Furthermore, allowing any contributor on the internet to cooperate with the Blockchain with computer power, disk space, and bandwidth - regardless of the amount to share.

Overall, the Blockchain proves to be resilient and a good fit for real-world deployment scenarios. The addition of PoDl [15] in replacement of the stake is a perfect fit for supporting multiple trails/distributions on the same Blockchain, subject to be addressed in further publications.

REFERENCES

[1] G. Ferreira, L. Jia, J. Sunshine, and C. Kastner, ''Containing malicious package updates in npm with a lightweight permission system,'' in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, May 2021, pp. 1334–1346.

[2] P. Abate, R. Dicosmo, R. Treinen, and S. Zacchiroli, ''Mpm: A modular package manager,'' in *Proc. Federated Events Component-Based Softw. Eng. Softw. Archit. (CBSE)*, 2011, pp. 179–187.

[3] A. Hindle, Z. M. Jiang, W. Koleilat, M. W. Godfrey, and R. C. Holt, ''YARN: Animating software evolution,'' in *Proc. 4th IEEE Int. Workshop Visualizing Softw. Understand. Anal.*, Jun. 2007, pp. 129–136.

[4] PyPi. (2020). *Pip Search Has Been Temporarily Disabled*. [Online]. Available: https://github.com/pypa/pip/issues/9312

[5] P. Software Foundation. *Python Package Index*. Accessed: Sep. 30, 2018. [Online]. Available: https://pypi.org/

[6] A. Maven. (2021). *Apache Maven Project*. [Online]. Available: https://maven.apache.org/

[7] NPM. (2021). *Npm: Build Amazing Things*. [Online]. Available: https://www.npmjs.com/

[8] Crates. (2021). *Crates: The Rust Community's Crate Registry*. [Online]. Available: https://crates.io/

[9] Yarn. (2021). *Yarn: Safe, Stable, Reproducible Projects*. [Online]. Available: https://yarnpkg.com/

[10] G. Lee, T. Moon, M. Jang, and H. Kim, ''EAPT: Enhancing APT with a mirror site resolver,'' in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2020, pp. 117–122.

[11] C. Dale and J. Liu, ''Apt-p2p: A peer-to-peer distribution system for software package releases and updates,'' in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 864–872.

[12] H. Herry, E. Band, C. Perkins, and J. Singer, ''Peer-to-peer secure updates for heterogeneous edge devices,'' in *Proc. NOMS IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2018, pp. 1–5.

[13] Q. Zhang, J. Yu, L. Luo, J. Ma, Q. Wu, and S. Li, ''An optimized DHT for Linux package distribution,'' in *Proc. 15th Int. Symp. Parallel Distrib. Comput. (ISPDC)*, 2016, pp. 298–305.

[14] D. G. Feitelson, '''We do not appreciate being experimented on': Developer and researcher views on the ethics of experiments on open-source projects,'' 2021, *arXiv:2112.13217*.

[15] F. Z. da N. Costa and R. J. G. B. de Queiroz, ''A blockchain using proof-of-download,'' in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Nov. 2020, pp. 170–177.

[16] Debian. (2022). *Debian Semantic Versioning*. [Online]. Available: https://www.debian.org/doc/debian-policy/ch-controlfields.html

[17] T. Mueller, ''Let's attest! Multi-modal certificate exchange for the web of trust,'' in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2021, pp. 758–763.

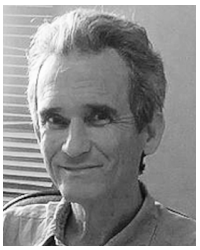
[18] Google. (2022). *Google Play Store*. [Online]. Available: https://play.google.com/store

[19] Microsoft. (2022). *Microsof Application Store*. [Online]. Available: https://www.microsoft.com/en-us/store/apps/windows

[20] Apple. (2022). *Apple Store*. [Online]. Available: https://www.apple.com/store

[21] M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath, "Coded merkle tree: Solving data availability attacks in blockchains," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), vol. 12059, J. Bonneau and N. Heninger, Eds. Cham, Switzerland: Springer, 2020. [Online]. Available: https://fc20.ifca.ai/, doi: 10.1007/978-3-030-51280-4_8.
[22] M. Friedenbach and K. Alm. (2017). *Bip98: Fast Merkle Hash-Tree*. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0098.mediawiki
[23] J. Benet, "IPFS–content addressed, versioned, P2P file system," 2014, *arXiv:1407.3561*.
[24] E. Nyaletey, R. M. Parizi, Q. Zhang, and K.-K.-R. Choo, "BlockIPFS–blockchain-enabled interplanetary file system for forensic and trusted data traceability," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Jul. 2019, pp. 18–25.
[25] PL Inc. (2016). *A Protocol for Disambiguating the Encoding*. [Online]. Available: https://github.com/multiformats/multibase
[26] (2016). *Canonical Table of of Codecs Used by Various Multiformats*. [Online]. Available: https://github.com/multiformats/multicodec
[27] (2016). *Self Identifying Hashes*. [Online]. Available: https://github.com/multiformats/multihash
[28] IPFS-Devs. (2020). *IPFS Gateway*. [Online]. Available: https://docs.ipfs.io/concepts/ipfs-gateway/#overview
[29] Y. Wang, Z. Su, J. Ni, N. Zhang, and X. Shen, "Blockchain-empowered space-air-ground integrated networks: Opportunities, challenges, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 1, pp. 160–209, 1st Quart., 2022.
[30] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[31] A. Antonopoulos, *Mastering Bitcoin*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, May 2017.
[32] Q. Zheng, Y. Li, P. Chen, and X. Dong, "An innovative IPFS-based storage model for blockchain," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. (WI)*, Dec. 2018, pp. 704–708.
[33] M. Steichen, B. Fiz, R. Norvill, W. Shbair, and R. State, "Blockchain-based, decentralized access control for IPFS," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, Jul. 2018, pp. 1499–1506.
[34] Y. Chen, H. Li, K. Li, and J. Zhang, "An improved P2P file system scheme based on IPFS and blockchain," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 2652–2657.
[35] N. Nizamuddin, K. Salah, M. A. Azad, J. Arshad, and M. H. Rehman, "Decentralized document version control using ethereum blockchain and IPFS," *Comput. Electr. Eng.*, vol. 76, pp. 183–197, Jun. 2019.
[36] N. Nizamuddin, H. R. Hasan, and K. Salah, "IPFS-blockchain-based authenticity of online publications," in *Blockchain—ICBC* (Lecture Notes in Computer Science), vol. 10974, S. Chen, H. Wang, and L. J. Zhang, Eds. Cham, Switzerland: Springer, 2018. [Online]. Available: http://blockchain1000.org/2018/, doi: 10.1007/978-3-319-94478-4_14.
[37] C. Format and M. T. Comma-Separated Values (CSV) Files. (2005). *Shafranovich*. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4180
[38] F. R. Cogo, G. A. Oliva, and A. E. Hassan, "Deprecation of packages and releases in software ecosystems: A case study on NPM," *IEEE Trans. Softw. Eng.*, vol. 48, no. 7, pp. 2208–2223, Jul. 2022.
[39] PL Building the Next Generation of the Internet. (2022). *Protolabs*. [Online]. Available: https://protocol.ai/
[40] J. Heun. (2021). *JavaScript Libp2p MulticastDNS Discovery Implementation*. [Online]. Available: https://github.com/libp2p/js-libp2p-mdns
[41] V. Santos. (2021). *JavaScript libp2p Implementation of the Railing Process of a Node Through a Bootstrap Peer List*. [Online]. Available: https://github.com/libp2p/js-libp2p-bootstrap
[42] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Peer-to-Peer Systems* (Lecture Notes in Computer Science), vol. 2429, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Germany: Springer, 2002. [Online]. Available: https://dblp.org/db/conf/iptps/iptps2002.html, doi: 10.1007/3-540-45748-8_5.
[43] M. Claver, J. Schmerge, J. Garner, J. Vossen, and J. McClurg, "REGIS: Regular expression simplification via rewrite-guided synthesis," 2021, *arXiv:2104.12039*.
[44] Linode. (2022). *Linode and AMD*. [Online]. Available: https://www.linode.com/amd/
[45] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: Current status and future plans," in *Proc. Linux Symp.*, vol. 2. Princeton, NJ, USA: Citeseer, 2007, pp. 21–33.
[46] J. Wusteman, "RSS: The latest feed," *Library Hi Tech*, vol. 22, no. 4, pp. 404–413, Dec. 2004.

FELIPE Z. DA N. COSTA is currently pursuing the Ph.D. degree with the Federal University of Pernambuco, where he is conducting researches about blockchains. He is also an Architect of ModSecurity version 3. He was ModSec's Lead Dev for eight years. He has a strong background on the (in-)security of GSM networks. Engaged with open source development, he has contributions on assorted projects, from web browsers to 3D printer software.



RUY J. G. B. DE QUEIROZ received the B.Eng. degree in electrical engineering from the Escola Politécnica de Pernambuco, in 1980, the M.Sc. degree in informatics from the Universidade Federal de Pernambuco, in 1984, and the Ph.D. degree in computing from the Imperial College London, in 1990, for which he defended the dissertation Proof Theory and Computer Programming: An Essay into the Logical Foundations of Computation. He is currently an Associate Professor at the Universidade Federal de Pernambuco and holds significant works in the research fields of mathematical logic, proof theory, foundations of mathematics, and philosophy of mathematics. He is the Founder of the Workshop on Logic, Language, Information and Computation (WoLLIC), which has been organised annually, since 1994, typically in June or July.

GUSTAVO P. BITTENCOURT received the B.Sc. degree in computer engineering from the Federal University of Pernambuco (UFPE), in 2017, where he is currently pursuing the M.Sc. degree in computer science. Since 2006, he has been working in the private sector with cybersecurity, dedicating the last few years of his professional career on the security assessment of digital solutions for financial institutions.

LEOPOLDO TEIXEIRA is currently an Assistant Professor with the Informatics Center (CIn), Federal University of Pernambuco, where he leads the Software Testing and Analysis Research Group, and is also affiliated with the Software Productivity Group. In 2022, he is also a CAPES-Alexander von Humboldt Experienced Research Fellow, working at the Universität des Saarlandes. His research interests include software engineering, with a focus on providing strong foundations for improving software quality and productivity. In particular, he worked on software product lines and configurable systems, refactoring, formal methods, software testing, and mobile development.