An Evaluation of IPFS as a Distribution Mechanism for RPKI Repository

Dadepo Aderemi
dadepo.aderemi@os3.nl

Woudt van Steenbergen
woudt.vanSteenbergen@os3.nl

Research Project 2 - Master Security and Network Engineering (SNE) - Universiteit van Amsterdam (UvA)

Abstract— **IPFS is a distributed, content-addressable, and peer-to-peer file system that can be used to share content on the internet. RPKI is an approach to securing global inter-domain routing by providing a mechanism to authorize and validate route announcements made on the internet. The RPKI approach requires cryptographically signed objects to be distributed from parties making the authorization to parties validating the authorization. This task is currently achieved using rsync or the RPKI Repository Delta Protocol, which builds on HTTPS. In this research, we investigated the use of IPFS as a distribution mechanism for RPKI objects. We performed both qualitative analysis and quantitative analysis to determine how suitable IPFS is for use in RPKI. Our qualitative analysis revealed features native to IPFS that could complement or obsolete part of the current distribution mechanism within RPKI. Our quantitative study focused on measuring network performance of IPFS, and it shows IPFS is less performant for data transfer than HTTPS.**

Index Terms— **Networks, Network security, Peer-to-peer computing**

I. Introduction

Resource Public Key Infrastructure (RPKI) is a Public Key Infrastructure (PKI) approach to securing the Border Gateway Protocol (BGP) used in global Internet routing. RPKI provides an extra security layer by allowing provable cryptographic attestations regarding the ownership of Internet Number Resources (INR) - Internet Protocol (IP) version 4 and version 6 and the Autonomous System Numbers (ASNs). The current application of RPKI makes it possible to authenticate the relationship between these INRs and route announcements in BGP.

The RPKI materials used for validating BGP announcements are published to known publishing points called RPKI repositories where they are retrieved by validating software. At the moment of writing, rsync and the RPKI Repository Delta Protocol (RRDP) [1] are the only two protocols that are used in publishing to and retrieving contents from RPKI repositories.

The Interplanetary File System (IPFS) is a peer-to-peer distributed file system that uses a content-addressed block storage model. The content-addressed block forms a generalized Merkle Directed Acyclic Graph (DAG), a data structure upon which a versioned file system and a permanent web can be built. IPFS routing is based on a modified version of the Kademlia Distributed Hash Table (DHT), while the identity of peers is based on public-key cryptography [2]. The usage of content-addressing, DHT for routing, and public-key cryptography for node identification ensures IPFS can be a trustless peer-to-peer network with no single point of failure.

A. Motivation

The mechanism used for publishing RPKI content has evolved. The first RPKI publication servers used rsync, which has since shown to have significant limitations in practice. These limitations include the high computation resources in terms of CPU and memory needed by the publication server, as it needs to continually compute diffs of content when responding to clients' requests. This situation does not scale as the adoption of RPKI increases with more clients requesting content from RPKI repositories. Another drawback of rsync is the lack of supported server and client libraries. This forces reliance on installed rsync binaries within RPKI implementations. This dependence on installed binaries makes upgrade difficult and leads to fragility [1].

RRDP was conceived as an alternative mechanism for publishing RPKI contents with an explicit goal of ameliorating the drawbacks of rsync. It is designed to require less computing resources by being able to take advantage of caching. It does this by pre-computing the delta arising from an update to the content in the publication server. This computation is done once, upon content updates and not upon every request made by the clients, reducing the computation resources required to run an RPKI repository. The resulting delta files are essentially immutable content and hence, can be cached indefinitely by an HTTP over Transport Layer Security (HTTPS) server or by a Content Delivery Network (CDN) reducing the computation resources needed to serve contents from an RPKI repository.

IPFS seeks to be a technology that offers versioning, with low global latency, high-throughput oriented file systems for distributing data. [2]. As part of achieving this goal, IPFS incorporates ideas from many past successful systems. A non-exhaustive list of these ideas includes content addressing, which is a manifestation of an aspect of Information-Centric Networking [3] (ICN), PKI based identity of participating nodes, and the embracing of immutable data structures. The ideas implemented as part of IPFS makes it an interesting candidate upon which a mechanism for distributing cryptographic materials within RPKI could potentially be built. In this research, we studied the characteristics of IPFS, performing both a quantitative and qualitative analysis in other to understand how it could be incorporated within RPKI.

B. Research Questions

Our primary research question is:

To what extent can IPFS be used as a distribution mechanism between RPKI repositories and RPKI Relying Parties.

In order to be able to answer this question, we articulated the following sub-questions:

1) How is publishing and retrieving contents currently implemented with RRDP in RPKI?
2) What are the features of IPFS that can replace or augment current RRDP implementation of the RPKI repository?
3) What are the network characteristics of IPFS, and how would these characteristics influence the operations of an RPKI repository.

We did not include a comparison with rsync in the scope of this research.

C. Structure

The rest of this paper is organized as follows. In section II, we survey related work of this research. In section III, we outline the necessary concepts that underpin the results presented in this paper. In section IV, we explained the methodology used to carry out this research and described the experiments performed. The results of our research are presented in section V, while critical reflections on the results are presented in section VI. The conclusion that can be inferred from our results is given in section VII. Finally, in section VIII we give suggestions for future research.

II. Related Work

As at the time of writing, we are unaware of any research targeting RRDP. RRDP is a relatively new protocol specified by the Internet Engineering Task Force (IETF) as *rfc8182* in 2017. We believe that the relative infancy of RRDP is responsible for the paucity of research around it.

rsync, on the other hand, is an older technology that was released in 1996 [4]. Although no formal specification exists for it, its algorithm is well-described [5]. It has been implemented in various applications and has enjoyed reasonable deployment [6], with multiple studies done involving it. These studies include how rsync can be applied to improve file distribution, and also methods on how to enhance rsync itself.

Despite the utility of rsync, it has been found that it brings about operational challenges when used to distribute cryptographic materials in RPKI repositories. The requirement of significant computing resources in terms of CPU and memory, including lack of supported server and client libraries [1], are cited in the RFC that specifies RRDP.

Data storage and distribution are usually done via a traditional server-client architecture where a server stores contents and make it available to clients. The need to improve the storage and distribution of data has led to the exploration of other architecture apart from the traditional server-client approach. One such method is the use of peer-to-peer technology. In [7], a torrent based approach to software distribution in grid computing is explored. While [8] surveys various peer-to-peer content distribution technologies.

IPFS is a specific implementation of peer-to-peer technology and has also been explored as a means for data storage and distribution. [9] proposes a first-class object store service for Fog/Edge facilities built using Scale-out Network Attached Storage systems (NAS) and IPFS. In [10], IPFS is used as the storage environment for storing electronic medical records. IPFS has also been proposed and applied [11] [12] [13] for off-chain storage within Blockchain.

Recently, Netflix Inc. [14] experimented with IPFS for container image distribution across Amazon Web Service regions [15]. The usage of IPFS resulted in reduced download times for containers when compared with the HTTPS-based delivery mechanism of Docker Hub [16] and the Titus Registry [17].

In [18], I/O Performance of IPFS storage from a client's perspective is studied. In [19], the performance of IPFS as an object store for Fog/Edge Computing Infrastructures is studied and compared with Rados [20], Cassandra [21]. At the same time, [22], analyzes the performance of IPFS for Video on Demand (VOD) locally within an ISP, with an ICN network layer enhancement proposed to improve delivery quality.

III. Background

In this section, we overview the technologies and concepts that underpin the research presented in this paper. We first give an overview of the RPKI architecture, covering relevant components and protocols. Next, we describe the RPKI validation process. We then conclude with selected parts within IPFS that are relevant to the research.

A. BGP and Routing Security

BGP is a de facto standard protocol used for inter-domain routing on the internet of today. It is an exterior gateway protocol that provides the mechanism for Autonomous Systems (AS) to share network reach-ability information via BGP announcements. Unfortunately, the original BGP specification provided no security mechanism for validating the authenticity of these network reach-ability information shared with BGP. By default, BGP has traditionally operated with a default-accept mode [23], where any AS can originate any BGP announcement and be accepted by other ASes.

This situation makes BGP vulnerable to exploitation like prefix and sub prefix hijacks, where a malicious actor can black-hole traffic destined for hijacked prefixes or redirect it to unintended destinations [24]. RPKI is a mechanism that seeks to prevent these types of exploitation within the inter-domain routing. It does this by providing a way to authenticate the validity of network reach-ability information contained within BGP announcements made by ASes.

B. RPKI Primer

1) Overview of RPKI: RPKI is a PKI approach to securing global inter-domain routing. RPKI bases its security model upon a top-down hierarchy rooted at the Regional Internet Registries (RIR) that allocates and sub allocate IP address space. This hierarchy forms a chain of trust that can verify the final ownership of an IP address space. It also makes it possible for the owner of an IP address space to grant an AS the authority to originate BGP announcements for that IP space.

2) RPKI Components: RPKI being a PKI, consists of various components involved with the life-cycle management of X.509 digital certificates and the application of public-key cryptography. The essential elements of the RPKI infrastructure are described next. The Certificate Authority (CA) refers to the entity that issues and revokes certificates. In RPKI, these are the RIRs or the National Internet Registries (NIR). The Trust Anchor (TA) is the root CA in a hierarchical chain of trust, from which trust is assumed and not derived. In the case of RPKI, these are the five RIRs (AfriNIC, APNIC ARIN, LACNIC, and RIPE NCC). The Certificate Revocation List (CRL) is a standard PKI component that contains all certificates revoked by a particular CA. The End Entity (EE) refers to a holder of certificates issued by a CA. An EE can make cryptographically verifiable statements using their certificates. In RPKI, the owners of IP address space are EEs, and the certificate they own is referred to as resource certificates. The cryptographically variable statements that are made by certificates are referred to as Signed Objects. In RPKI, a type of signed objects is Resource Origin Authorization (ROAs) [25], which are used to authorize an AS to originate BGP announcement for an IP address space. Another example of a Signed Object within RPKI is the Manifest File [26], which is a file that contains all other Signed objects files in the repository publication point. The RPKI repository is the publication point, and this is where RPKI objects like X.509 certificates, CRLs, Manifests, and Signed Objects can be retrieved. Finally, the Relying Party (RP) is the component that verifies the cryptographically verifiable statements made by EEs. The RP does its verification based on RPKI objects it retrieves from the RPKI repository.

C. RPKI and BGP Origin Validation

BGP origin validation refers to the process of checking the authorization of route announcements originating from an AS. RPKI can be used to perform BGP origin validation. When an IP address space is allocated, a resource certificate

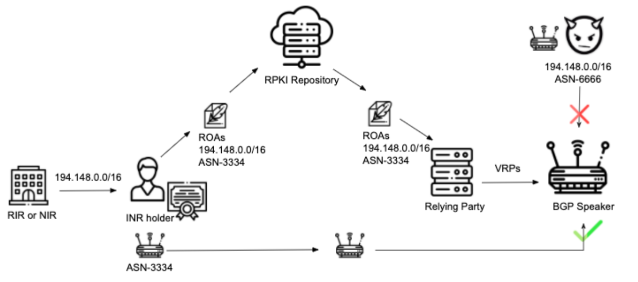


Fig. 1. Overview of RPKI and BGP origin validation.

is also granted with the allocation. The resource certificate can then be used to create ROAs, which are cryptographically verifiable statements about which AS is allowed to make route announcement for the address space. The ROAs and other RPKI objects are then published to a publicly available RPKI repository where they can be retrieved by the RP who then perform the BGP origin validation. The validation process's outcome is the Validated ROA Prefixes (VRP). These VRPs are then ingested into the BGP speakers using the RPKI to Router Protocol (RTR), where it is used to determined routing decisions. The components involved in the validation process is seen in Figure 1

D. RPKI Repository Delta Protocol

RRDP describes a publication and retrieval mechanisms of RPKI objects from the RPKI repositories. It is designed to utilize caching infrastructure as a strategy for scaling. It makes use of Notification Files, which contain pointers to a Snapshot file and Delta Files. An example of the notification file used in RRDP is seen in Figure 2.

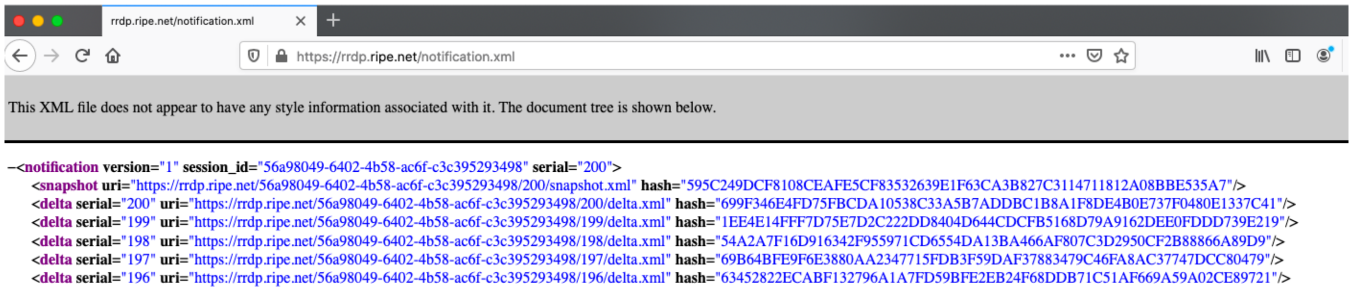
The Snapshot file contains the current snapshot of the RPKI repository at the start of an RRDP session, while the Delta files contain changes to the RPKI repository since the beginning of an RRDP session. The Snapshot file and Delta Files can be retrieved via HTTPS. The Snapshot and Delta files are an immutable record of the RPKI repository state for a given RRDP session. This enables caching via CDNs or other caching infrastructures [1].

E. Trust Anchor Locator

The validation process described in section III-C starts with retrieval of a TA, which then points to the location of the RPKI repository from where the other RPKI objects can be retrieved. In RPKI, The Trust Anchor Locator (TAL) [27] is a file that is used to distribute the location to the TA and its public key. It allows RPs to verify the retrieved TA by checking it against the key listed in the TAL. An example of the TAL is seen in Figure 3

F. IPFS Primer

1) Overview of IPFS: The Interplanetary File System (IPFS) is a peer-to-peer, content-addressable distributed file system upon which a permanent web can be built [2].

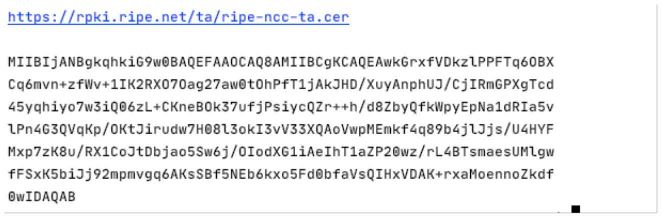


rrdp.ripe.net/notification.xml
https://rrdp.ripe.net/notification.xml

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
–<notification version="1" session_id="56a98049-6402-4b58-ac6f-c3c395293498" serial="200">
    <snapshot uri="https://rrdp.ripe.net/56a98049-6402-4b58-ac6f-c3c395293498/200/snapshot.xml" hash="595C249DCF8108CEAFE5CF83532639E1F63CA3B827C3114711812A08BBE535A7"/>
    <delta serial="200" uri="https://rrdp.ripe.net/56a98049-6402-4b58-ac6f-c3c395293498/200/delta.xml" hash="699F346E4FD75FBCDA10538C33A5B7ADDBC1B8A1F8DE4B0E737F0480E1337C41"/>
    <delta serial="199" uri="https://rrdp.ripe.net/56a98049-6402-4b58-ac6f-c3c395293498/199/delta.xml" hash="1EE4E14FFF7D75E7D2C222DD8404D644CDCFB5168D79A9162DEE0FDDD739E219"/>
    <delta serial="198" uri="https://rrdp.ripe.net/56a98049-6402-4b58-ac6f-c3c395293498/198/delta.xml" hash="54A2A7F16D916342F955971CD6554DA13BA466AF807C3D2950CF2B88866A89D9"/>
    <delta serial="197" uri="https://rrdp.ripe.net/56a98049-6402-4b58-ac6f-c3c395293498/197/delta.xml" hash="69B64BFE9F6E3880AA2347715FDB3F59DAF37883479C46FA8AC37747DCC80479"/>
    <delta serial="196" uri="https://rrdp.ripe.net/56a98049-6402-4b58-ac6f-c3c395293498/196/delta.xml" hash="63452822ECABF132796A1A7FD59BFE2EB24F68DDB71C51AF669A59A02CE89721"/>
```

Fig. 2. Screenshot of RIPE's notification.xml.



```
https://rpki.ripe.net/ta/ripe-ncc-ta.cer

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwkGrxfVDkzlPPFTq6OBX
Cq6mvn+zfWv+1IK2RXO7Oag27aw0tOhPfT1jAkJHD/XuyAnphUJ/CjIRmGPXgTcd
45yqhiyo7w3iQ06zL+CKneBOk37ufjPsiycQZr++h/d8ZbyQfkWpyEpNa1dRIa5v
lPn4G3QVqKp/OKtJirudw7H08l3okI3vV33XQAoVwpMEmkf4q89b4jlJjs/U4HYF
Mxp7zK8u/RX1CoJtDbjao5Sw6j/OIodXG1iAeIhT1aZP20wz/rL4BTsmaesUMlgw
fFSxK5biJj92mpmvgq6AKsSBf5NEb6kxo5Fd0bfaVsQIHxVDAK+rxaMoennoZkdf
0wIDAQAB
```

Fig. 3. Example of an RPKI TAL.

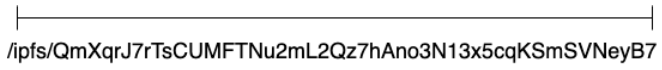


Fig. 4. Example of an IPFS path.

The IPFS specification is captured in [28] and has been implemented in various programming languages [29]. This research is based on *go-ipfs* [30] an implementation in the Go programming language [31].

2) Content Addressing and Content Identifier: As stated in I, IPFS implements content addressing, which allows for contents to be addressed within the network by name and not by their location. The name used in IPFS is derived from the cryptographic hash of contents. IPFS uses SHA-256 by default[32]. This has the side effect of integrity and content deduplication being natively available within IPFS. The hash value, which is used to address the content, is known as the Content Identifier (CID). The algorithm governing how the CIDs are derived from content within IPFS is specified in the InterPlanetary Linked Data (IPLD) [33] [34].

3) Content Chunking, Merkle DAG and Storage: IPFS supports the storage of any type of content. Content is stored as *objects* in its data store. The process of ingesting content into IPFS involves splitting the contents into blocks[35]. Individual CIDs are generated for these blocks and then used to construct a Merkle DAG, which is also specified as part of IPLD. This allows for distributed storage of contents within the network while also supporting integrity checks. The generated CID is also used to address and retrieve content. An example of a CID path is seen in Figure 4

G. Content Discovery and Distributed Hash Table

IPFS is a peer-to-peer distributed network; hence it has no centralized entity that coordinates how nodes find each other and how content is discovered within the network. This network functionality of IPFS is based on the libp2p [36] library, which provides an implementation of a Distributed Hash Table (DHT), through which IPFS achieves content and peer discovery. Joining the so-called swarm is achieved by connecting to a bootstrapping node that is already connected to the swarm and aids in the discovery of other connected nodes.

The DHT is a hash table that is distributed across the IPFS network. It contains information regarding where contents are hosted. It also creates a logical network layer and provides an algorithm that allows for iterative probing of the network for content. The DHT used by IPFS is based on Kademlia [37].

H. Peer Identity

IPFS identifies peers based on a PKI namespace [2] [38] where the hash of a public key indicates the identity of a node. This hash is often referred to as PeerId, and as of writing, RSA, Ed25519, Secp256k1, and ECDSA are the four supported key types, while SHA-256 is the hashing function used [39] to derive the PeerId. A node's public key is also used in establishing secure connections with other nodes while enabling identity verification. The identity verification is done by confirming that the hash of the public key used in the connection process is the same as the PeerId of the node being connected to.

I. InterPlanetary Naming System

Content addressing makes working with immutable content laborious. This is because when content previously retrieved via a CID is mutated, the updated content will have a new CID. This means new CIDs needs to be continually known by clients in other to keep up to date. This problem is solved in IPFS via the InterPlanetary Naming System (IPNS) [40].

IPNS offers a mechanism for self certified, mutable naming based on public key cryptography. An IPNS name is the hash of a public key and it is used to construct a globally unique name space. An IPNS name makes use of the *ipns*

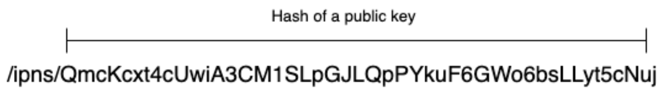


Fig. 5. Example of an IPNS name.

prefix as opposed to *ipfs* which is used as prefix in an IPFS path. An example of an IPNS name is seen in Figure 5:

The public key used is by default, the one used to identify the node, although alternative public key pair can be generated and used.

The corresponding private key is used to sign a record that contains the CID of the recent version of the content that can be retrieved from the IPNS name. This signed record is then put into the IPFS routing system that is based on the DHT. Parties who are interested in updated content found at the IPNS name can then continually fetch these signed records from the routing system where they can then retrieve the most recent published CID.

IV. Methods

To determine how suitable IPFS is for implementing an RPKI repository, we performed both a qualitative and quantitative analysis of IPFS in the context of RPKI. The qualitative analysis involved studying IPFS and RRDP's operational features to understand what aspects of publishing RPKI objects to the RPKI repository can be augmented or replaced by IPFS. The quantitative analysis involved a performance comparison between IPFS and RRDP when used as a mechanism to distribute RPKI objects. We also performed experiments to understand the peer-to-peer network overhead of IPFS as part of the quantitative analysis.

The qualitative analysis took the form of a literature study, in which input from documentation, architecture descriptions, and technical specifications of IPFS and RRDP was gathered and analyzed.

The quantitative analysis took the form of experiments in which we conducted two classes of tests. The first was a performance comparison of both HTTPS and IPFS when used for data transfer. The second involved modifying existing RPKI repository software and an RPKI RP software to use IPFS instead of RRDP. We then observed the performance characteristics due to the introduction of IPFS. The two software that was modified to use IPFS is Krill [41] and Routinator [42]. Krill is an RPKI Certificate Authority with support for being an RPKI repository, while Routinator is an RPKI RP software.

We performed all experiments on a server with the hardware specification listed in Table I

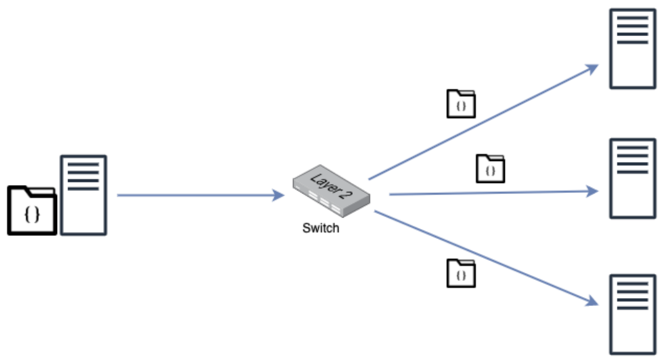


Fig. 6. Topology for HTTPS/IPFS Data Transfer Comparison. On the left side a node hosting data using nginx is connected to a switch. This switch is connected to other nodes that retrieve the data using HTTPS.

TABLE I
Hardware specification of server used for experiments.

Model name	Dell PowerEdge R240
Architecture	x86_64
CPU	Intel(R) Xeon(R) E-2124 CPU @ 3.30GHz
Memory	2x M391A1K43BB2-CTD 8GiB DDR4 @2666Mhz
Storage	Samsung SSD 860 EVO 465GiB
NIC	Virtualized full duplex 10Gbit/s (configurable)

A. Direct HTTPS and IPFS comparison

1) Experiment setup for HTTPS and IPFS comparison:

We used Containernet [43] as the network emulating environment to conduct the experiments that compare the data transfer performance of HTTPS and IPFS. Containernet is an extension of Mininet [44], which allows using Docker containers [45] as a host within the network emulation. We created a network topology [46] from an Ubuntu-based image that was modified to have IPFS installed. As seen in Figure 6, the topology consisted of one node that hosts data, with a switch that connects it to other nodes that download the hosted data using either HTTPS or IPFS. The data being requested are randomly generated files in a directory. In the HTTPS version of the experiment, nginx is used as the server providing the data while Wget is used as the client requesting data. The parallel utility [47] is used to request the data with Wget in parallel. The nginx server is configured to only support TLSv1.2, and the parallel requests spawned by the client are at most four at any given time.

The key parameters that we vary as part of the experiments are the number of nodes that are requesting the data directory, bandwidth between all nodes and connecting switch, the delay between nodes and switch and delay between node hosting data and switch. This is done in an attempt to isolate under what conditions having peering connections can be beneficial.

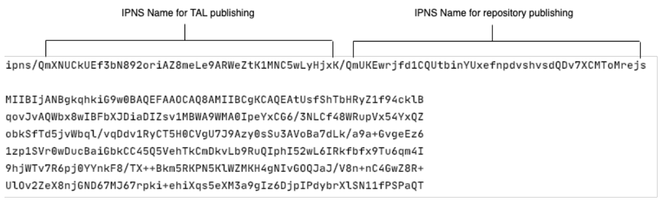
The list of software versions used and their role within the experiment setup is seen in Table II

B. HTTPS and IPFS comparison within RPKI

Various parts of Krill and Routinator were modified to enable the usage of IPFS instead of RRDP. We present an

TABLE II
SOFTWARE AND VERSION USED FOR DIRECT HTTPS AND IPFS COMPARISON.

Software	Version	Description
IPFS	0.5.1	IPFS
nginx	nginx/1.4.6 (Ubuntu)	HTTPS Web server
Parallel	GNU parallel 20161222	Tool to parallelize HTTP requests
Ubuntu	14.04.6 LTS	Operating System
Wget	GNU Wget 1.15	HTTPS Client


IPNS Name for TAL publishing
IPNS Name for repository publishing

```
ipns/QmXNUCkUEf3bN892oriAZ8meLe9ARWeZtK1MNC5wLyHjxK/QmUKEwrjfd1CQUtbinYUxefnpdvshvsdQDv7XCMToMrejs

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtUsfShTbHRyZ1f94cklB
qovJvAQWbx8wIBFbXJDiaDIZsv1MBWA9WMA0IpeYxCG6/3NLCf48WRupVx54YxQZ
obkSfTd5jvWbql/vqDdv1RyCT5H0CVgU7J9Azy0sSu3AVoBa7dLk/a9a+GvgeEz6
1zp1SVr0wDucBaiGbkCC45Q5VehTkCmDkvLb9RuQIphI52wL6IRkfbfx9Tu6qm4I
9hjWTv7R6pj0YYnkF8/TX++Bkm5RKPN5KlWZMKH4gNIvGOQJaJ/V8n+nC4GwZ8R+
UlOv2ZeX8njGND67MJ67rpki+ehiXqs5eXM3a9gIz6DjpIPdybrXlSN11fPSPaQT
```

Fig. 7. Modified TAL that supports fetching from IPFS using IPNS. The first address is for fetching the TAL, and the second address is for fetching from the repository.

overview of some of the sections that were modified, after which a description of the experimental setup is provided.

C. Modifying the TAL to support IPFS

The *rpki-rs* [48] library, which is used to implement the RPKI specific features within Krill and Routinator, was modified [49] to allow Routinator to be able to retrieve the TA certificate via IPFS and also the locator where the repository can be found. A modified version of the TAL can be seen in Figure 7.

1) Modifying Krill to publish the RPKI Objects to IPFS: Krill was modified [50] to support publishing RPKI objects to IPFS in addition to publishing to rsync and with RRDP. The modification involved making Krill post the TA certificate to IPFS via an IPNS name and also publish the contents of its repository to IPFS on creation and updates of RPKI materials. It makes use of the IPFS Go [30] binary to communicate over IPFS. A Docker image [51] based on the modified source code was created and used for the experiments.

2) Modifying Routinator to fetch RPKI objects from IPFS: Routinator was modified [52] to use only IPFS to fetch the RPKI objects it needs to perform validation. The changes involved making Routinator use the version of the TAL described in IV-C and to also capture the same duration metrics as it does when using rsync or RRDP. It uses the IPFS Go [30] binary to communicate over IPFS. A Docker image [53] based on the modified Routinator was created and used for the experiments.

3) Experiment setup with Krill and Routinator: The experiments to compare the performance characteristics of a modified instance of Krill and Routinator, which uses IPFS with unmodified versions, were also done using Docker containers without the networking emulating environment provided by Containernet. Instead, Docker-compose [54] was used to orchestrate the hosts that were used as part of the

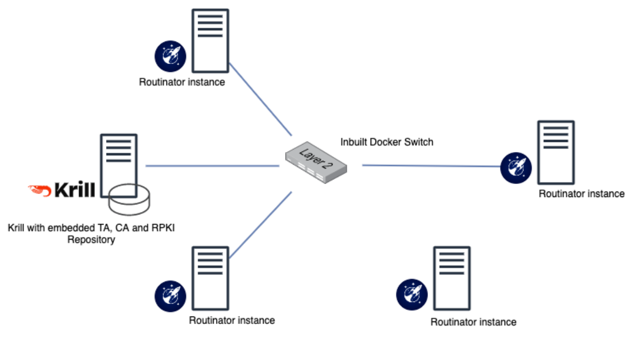


Fig. 8. Topology for HTTPS/IPFS comparison when used within RPKI. The leftmost node is where all RPKI material originates from, created by Krill which acts as a Trust Anchor and repository. This node is connected to the switch in the middle of the figure. This switch is connected to all other nodes that act as Relying Parties through the use of Routinator.

experiment. This decision was made with consideration to practicality as the IPFS versions of Krill and Routinator have incompatibilities with Containernet. The usage of Docker-compose has the disadvantage that network parameters can not be varied easily as within Containernet, but it has the advantage that it provides an environment closer to production since Containernet is not meant to run software in production while Docker-compose is.

Krill provides an inbuilt TA for use in testing scenarios. It also supports having both an embedded CA and an embedded RPKI repository in one running instance. For our test environments, we used the inbuilt TA, together with embedded CA and embedded RPKI repository. As our experiment is not concerned with observing performance characteristics when running Krill with embedded features versus when running Krill with remote features, we opted to simplify the experiment set up by using the embedded features of Krill. This simplification was made taking into cognizance the fact that the metrics we are interested in the tests, which is retrieval duration of RPKI objects by Routinator, are not influenced by running the RPKI repository embedded or remote.

We made it possible to specify in our test setup [55], the number of Routinator instances needed for each run of an experiment, and the number of child CA and ROAs that should be created within Krill. The experiment topology is depicted in Figure 8

The Routinator instance used for testing RRDP was modified to accept invalid Transport Layer Security (TLS) certificates. This modification was done in order to remove the need to set up TLS certificates needed to communicate over HTTPS with Krill. The change skips certificate validation and does not turn HTTPS off or have material impacts on the conducted experiments. A Docker image [56] based on the modified source code was created and used for the experiments.

The list of software versions used and their role within the experiment setup is seen in Table III. The software that was modified also lists the git hash from which the modification was made.

V. Results

We present the results of the qualitative analysis and quantitative analysis in this section. The qualitative study results are a theoretical evaluation of how IPFS can augment the distribution mechanism used by the RPKI repository. We leave out presenting possible ways the theoretical evaluations can be implemented. The quantitative analysis results are the outcome from the experiments performed comparing IPFS and HTTP for pure data transfer and when used within the context of RPKI.

A. Augmenting RPKI with IPFS

1) Remove the need for TAL: x509 certificates usually establish a binding between a public key and an Identity. Identities do not often change; hence it is practically feasible to distribute TA certificate and only have a redistribution at planned and infrequent intervals. This redistribution model will not be practical within RPKI. In RPKI, x509 certificates bind a public key to a set of IP address space, and these change more frequently than identities. Hence if the same model of redistribution of TA is to be applied to RPKI, then this redistribution would need to happen too often to be practical. RPKI fixes this problem by introducing the TAL.

As mentioned in III-E, the TAL is used to distribute the location to the TA and its public key. It allows for a stable locator to a TA certificate while allowing the certificate retrieved at that locator to change as needed. The public key contained within the TAL ensures that the validity of the retrieved certificate can always be checked.

Essentially the TAL provides a stable Uniform Resource Locator (URL) to content that can be updated alongside the mechanism to verify the authenticity of the retrieved content. This functionality of the TAL can be replaced by the inbuilt feature of IPNS within IPFS.

As mentioned in III-I, IPNS allows for self-certified, mutable naming based on public key cryptography. The fact that the record retrieved at an IPNS name is signed makes it possible, on content retrieval, for IPFS to check that the signature matches the public key corresponding to the node where it is published.

Hence having an IPNS name can serve as a replacement for TAL files.

2) Remove the need for checksums in RRDP notification file: The use of checksum is prevalent within RRDP. As described in III-D, the RRDP notification file contains a link to snapshot and links to delta files. Alongside the included links is the hash of the content that can be found at the listed links. This is seen in Figure 2. Including the hash makes verification of data integrity possible.

IPFS is tamper resistance [2]. Its usage of content addressing and the Merkle DAG ensures that data integrity checks are native in IPFS. As illustrated in Figure 9, using IPFS as a publication mechanism for RPKI objects removes the need for having extra integrity checks on retrieval.

3) Remove the need for RPKI Manifest file: The RPKI manifest file [26] is a signed file that allows RP software to detect malicious object deletion and the presence of valid but stale objects within the RPKI repository. It is an enumeration of all signed objects that should be present at an RPKI repository. The existence of this Manifest file then allows RP software to check the contents of the repository against what is listed in the Manifest file to detect any malicious tampering.

IPFS native tamper resistance can be applied in RPKI, removing the need for a Manifest file. IPFS access data by an identity that is derived from its content, rather than by its location. This makes it possible to detect tamperings, such as replacing an RPKI object with an old version or deletion natively.

4) Remove the need for Delta processing: RRDP specifies that updates to a repository are published as Delta files. RP software also needs to be able to process the Delta files accurately in order to arrive at the current state of the repository at a particular time. The algorithm to perform the Delta processing is not trivial [1] for both publishing and retrieval. Processing the Delta files requires checking their validity and considering all the new, replaced, and withdrawn objects. For a more in-depth explanation of Delta file processing, see RFC 8182 [1].

IPFS handles the resolution of content deltas transparently. This resolution is part of the properties of making use of a Merkle DAG. Another consequence of this is that data is deduplicated within IPFS, such that all objects containing the same content are identical, and only need to be stored once. Updating content then involves either creating new content, linking to existing content, or removing the link to existing content in the case of a destructive modification.

This essential feature of IPFS can simplify the process of publishing and retrieving RPKI objects from an RPKI repository.

B. Direct HTTPS and IPFS comparison

In the direct comparison between IPFS and HTTPS, we used a network setup with the following baseline parameters:

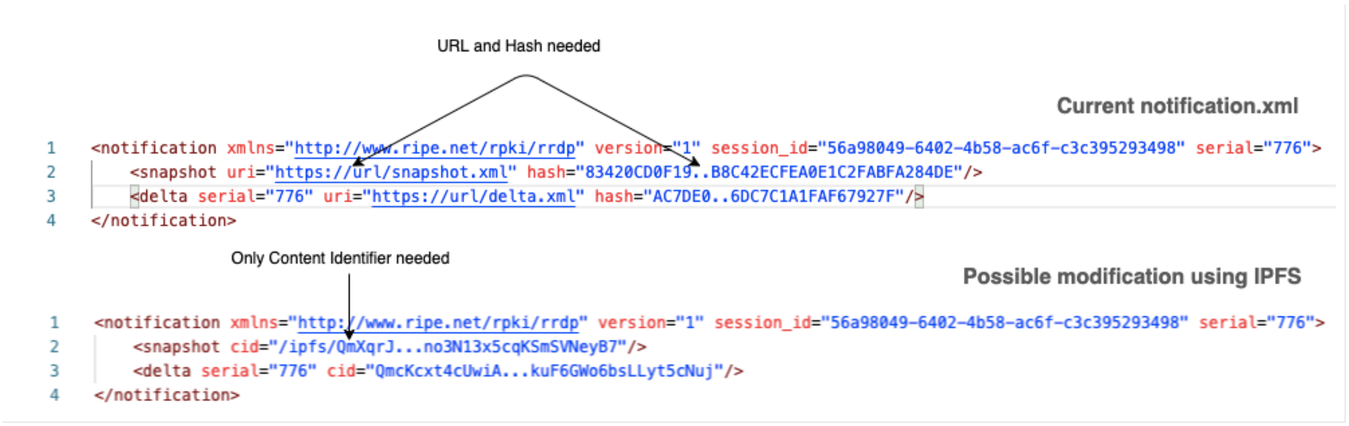
- The round-trip time between all nodes is 10ms
- The bandwidth between all nodes is 1000Mbit/s
- The amount of nodes in the network is 10, one of them being the node hosting the data
- The size of the data that will be retrieved is 500MiB divided into ten files

The resulting retrieval time of this baseline test can be seen in Figure 10 for the IPFS baseline and Figure 11 for the HTTPS baseline.

1) Varying latency: Figure 12 shows the results of running our IPFS test with an increased latency between the node hosting the data and the switch, causing the RTT between the server and the nodes to add up to 250ms, and the RTT between the other nodes themselves is 10ms. We observe that the first node to retrieve the file takes around 191 seconds compared to 59 seconds in the low latency environment, which is an increase of 224%. All subsequent nodes retrieve

TABLE III
SOFTWARE AND VERSION USED FOR HTTPS AND IPFS COMPARISON WITHIN RPKI CONTEXT.

Software	Version	Description
Docker	Community: 19.03.8	Container
Docker compose	version 1.25.5, build 8a1c60f6	Orchestrate containers
Krill	Krill 0.6.2	CA and RPKI repository
Krill modified with IPFS	Krill 0.6.2 (Git: d10c88f0)	CA and RPKI repository
Go-IPFS	0.5.1	IPFS implementation in Go
Routinator with TLS modified.	0.7.0-bis (Git:d6906b2b)	RPKI RP
Routinator with IPFS.	0.7.0-bis (Git:d6906b2b)	RPKI RP


URL and Hash needed
Current notification.xml
Only Content Identifier needed
Possible modification using IPFS

```
<notification xmlns="http://www.ripe.net/rpki/rrdp" version="1" session_id="56a98049-6402-4b58-ac6f-c3c395293498" serial="776">
    <snapshot uri="https://url/snapshot.xml" hash="83420CD0F19..B8C42ECFEA0E1C2FABFA284DE"/>
    <delta serial="776" uri="https://url/delta.xml" hash="AC7DE0..6DC7C1A1FAF67927F"/>
</notification>
```

```
<notification xmlns="http://www.ripe.net/rpki/rrdp" version="1" session_id="56a98049-6402-4b58-ac6f-c3c395293498" serial="776">
    <snapshot cid="/ipfs/QmXqrJ...no3N13x5cqKSmSVNeyB7"/>
    <delta serial="776" cid="QmcKcxt4cUwiA...kuF6GWo6bsLLyt5cNuj"/>
</notification>
```

Fig. 9. Example of the Content Identifier removing the need for a hash. The URL is replaced for a CID.

this file in less time. The retrieval time that is observed in each node other than the first has a higher standard deviation. For example, node 1 has a standard deviation of 1 second, whereas node 3 has a standard deviation of 6.8 seconds. These standard deviations are translated to a percentual standard deviation of 0.5% and 10.8%, respectively. Interestingly, node 2 sees an improvement in retrieval time as its median moves from 61 to 56 seconds, and its standard deviation from 1 to 5 seconds. We also see the medians and means of node 3 through node 9 rises by 6% and 10% on average, suggesting that the node with bad connectivity is exerting a negative effect on all peers except for the second to retrieve the data.

This same behavior is not observed in the HTTPS test, as shown in Figure 13. The standard deviation for all nodes lies between 2.2 and 4.6 seconds, which is translated into a percentual standard deviation of 11% and 23%. All HTTPS nodes see an increase in data retrieval duration of 259% on average when comparing the medians. This result is in line with the performance degradation of the first node in the IPFS experiment.

2) Varying bandwidth: After the latency experiment, we performed the next experiment. We reduced the bandwidth from 1000Mbit per second to 100Mbit per second on the link between the node hosting the data and the switch. All other nodes have a bandwidth of 1000Mbit per second, as is the case in the baseline. The results can be seen in Figure 14. When these results are compared with the baseline in Figure 10, we see a performance degradation on node 1 of 22% in relation to the median. On all other nodes, we observe a slight performance degradation or even improvement between 5.1% and -3.6% on node 3 and node 9, respectively. In the case of HTTPS the performance degradation was more severe as can be seen in Figure 15 compared to its baseline in Figure 11. When comparing the median of the tests performed on each node in both environments we see an average performance degradation of 704.4%.

3) Varying amount of nodes: We reduced the number of nodes to observe the effect this would have on the time taken for filesharing. This experiment was performed on IPFS only as it is a peering technology, and in our setup, we created a full mesh between nodes leading them to all communicate with each other. Note that in this test, we do not compare with the baseline, but instead with the results from the increased latency experiment in Figure 12. We reduced the number of nodes in the network from 10 to 4, one of which has the file stored locally. The results of this test can be seen in Figure 16. We observe that the nodes display similar results in both tests, as their medians shift by 0.1%, 2%, and 2%, respectively. For Node 3, however, we note that its 3rd quartile covers a smaller area suggesting more stability in retrieval time.

C. HTTPS and IPFS comparison within RPKI

1) Varying amount of nodes: In our RPKI setup, we performed experiments in which we reduced the number of nodes in the network that are concurrently retrieving RPKI material from the RPKI repository. After running the test for a short duration, we update the repository by creating more

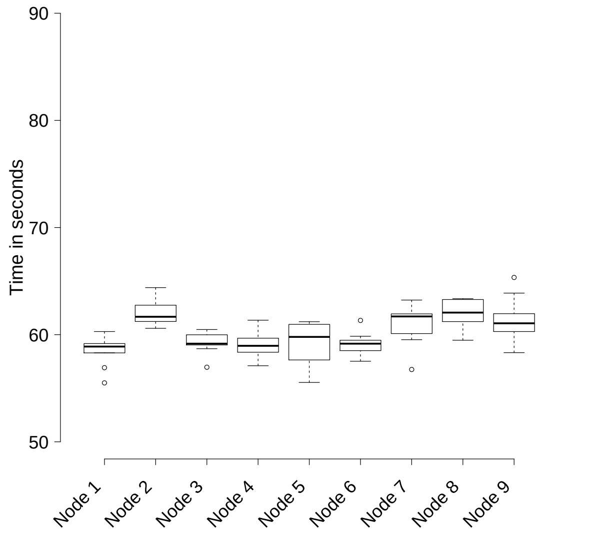


Fig. 10. Baseline for IPFS benchmark. Retrieval duration is plotted against the nodes where the measurement is taken from.

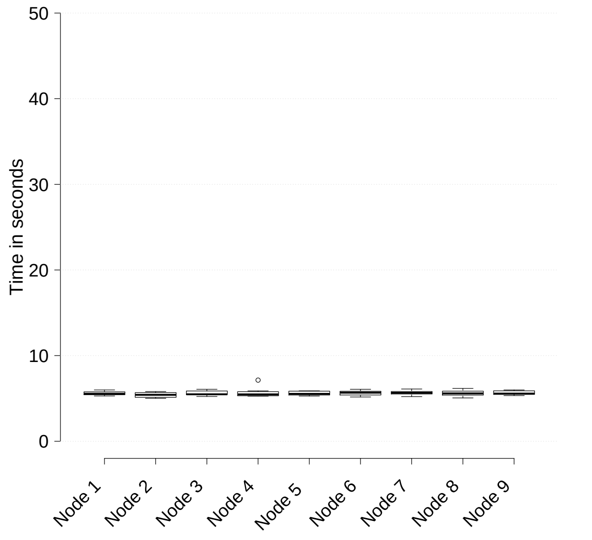


Fig. 11. Baseline for HTTPS benchmark. Retrieval duration is plotted against the nodes where the measurement is taken from.

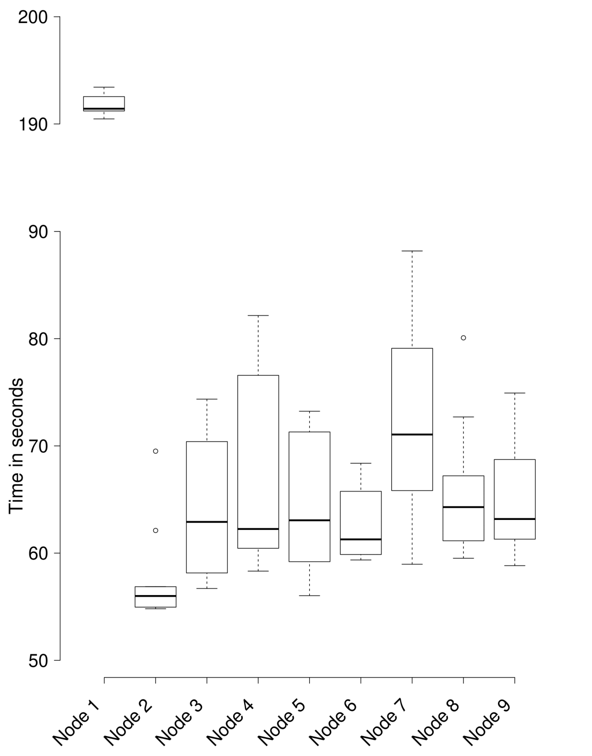


Fig. 12. IPFS latency benchmark. Retrieval duration is plotted against nodes where the measurement is taken from. Each node downloads a file after the previous node(s) have completed their download. Note that the y-axis shows a gap in time for improved readability.

certificate authorities and ROAs. These experiments were performed using IPFS and also with HTTPS. The results can be seen in Figures 17 and 18 for IPFS, as well as Figures 19 and 20 for HTTPS. In the test with ten nodes, both IPFS and HTTPS show an increase in fetch duration while ROAs are being added to the repositories over a period of around 2 minutes. This same result is observed in the four-node HTTPS test. After this increase in fetch duration, we see it stabilize again at a relatively short duration. The results from the four-nodes IPFS experiment show unexpected behavior; see Figure 18, the fetch duration does not decrease as harshly as is observed in the other tests when no updates to the repository are taking place. This behavior is undesirable since the repository is not receiving any updates, and as such, each poll to the unmodified repository should be short because the Relying Parties already have an up to date repository stored locally.

D. Peer-to-Peer Network Overhead

We were also interested in knowing the network overhead incurred due to IPFS being a peer-to-peer technology. To do this involves measuring the passive network activities within IPFS. We set up the topology described in IV-A.1

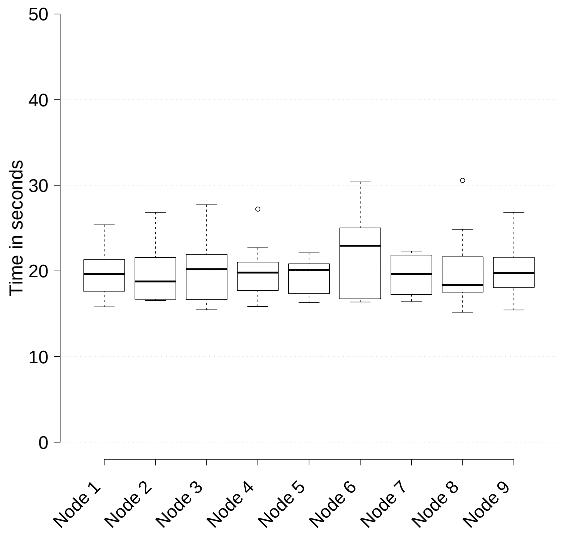


Fig. 13. HTTPS latency benchmark. Retrieval duration is plotted against nodes where the measurement is taken from. Each node downloads a file after the previous node(s) have completed their download.

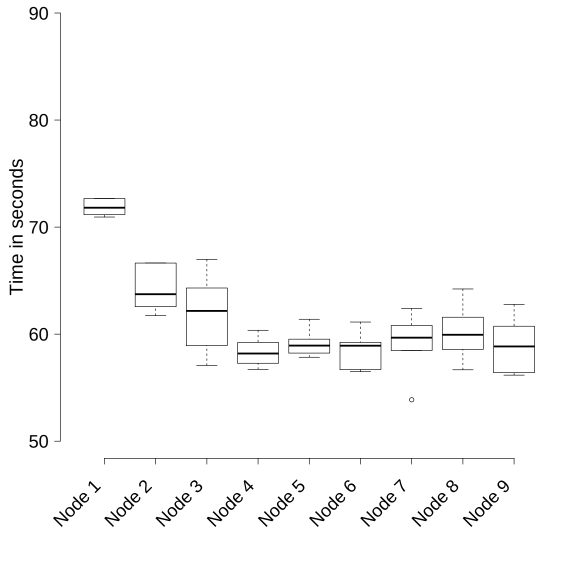


Fig. 14. IPFS reduced bandwidth benchmark. The node hosting the file has a bandwidth of 100Mbit/s available. Retrieval duration is plotted against the nodes where the measurement is taken from.

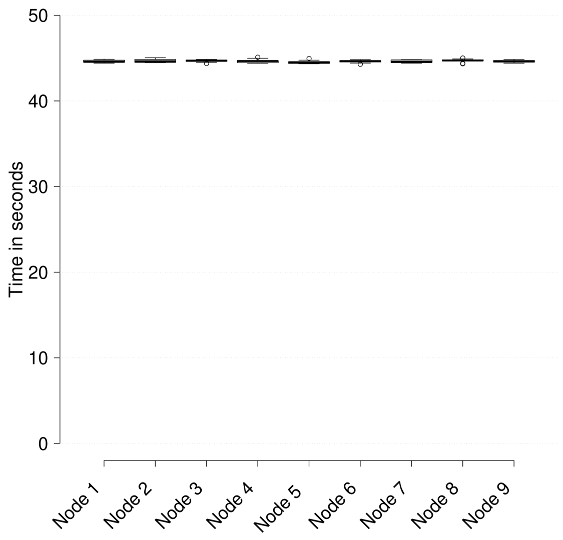


Fig. 15. HTTPS reduced bandwidth benchmark. The node hosting the file has a bandwidth of 100Mbit/s available. Retrieval duration is plotted against the nodes where the measurement is taken from.

but did not start upload or downloading of contents, instead, we made use of an IPFS CLI command [57] that returns total bandwidth in and out, together with the rate at any particular point in time. We ran this command every minute for 10 minutes with topology with an increasing number of nodes, starting from one to ten nodes. This measurement was only sampled once.

Figure 21 shows data captured when the latent bandwidth used is sampled at the 10th minute from the first node, which is the boot node.

Figure 22 shows data captured when the latent bandwidth used is sampled on the last node that gets added to the topology. The last node ranged from being the first node to being the 9th node in the topology with ten nodes.

VI. Discussion

A. Literature Study

IPFS has native features that can augment or replace similar features implemented in RRDP. IPFS naming feature, IPNS could remove the need for having a TAL. The use of content addressing and the Merkle DAG's self-verifying nature removes the need for having checksum and manifest files. The need for Delta processing can also be avoided as IPFS handles content synchronization and deduplication natively.

IPFS being a peer-to-peer system, could offer more resiliency than the server-client model of RRDP. RP software would still be able to retrieve RPKI repository contents from the network if the original node that produces the content goes offline. As long as its content has been retrieved, at least

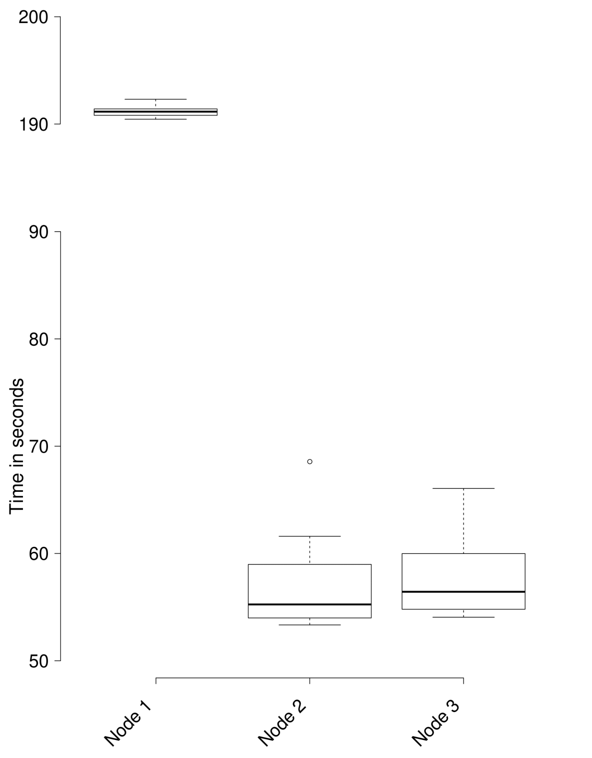


Fig. 16. IPFS experiment with a reduced number of nodes. This network contains four nodes. The node hosting the file has an RTT of 250ms between itself and all other nodes. The RTT between the other nodes is 10ms. Retrieval duration is plotted against the nodes where the measurement is taken from. Note that the y-axis shows a gap in time for improved readability.

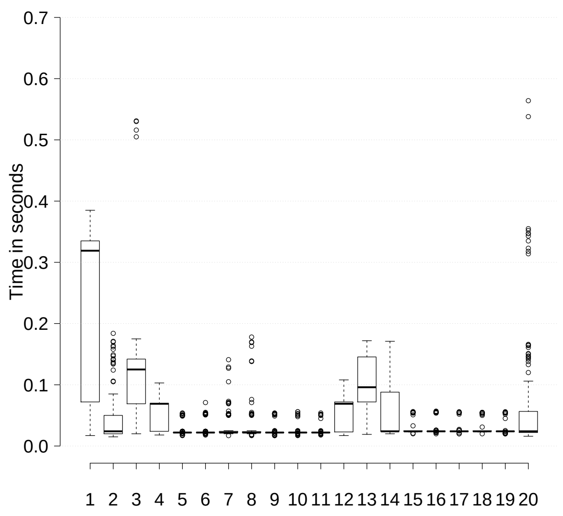


Fig. 17. IPFS experiment in RPKI with 10 nodes. One node is acting as the Trust Anchor and repository. The other nodes fetch from the repository every minute. Fetch duration is plotted on the Y-axis against the current fetch iteration.

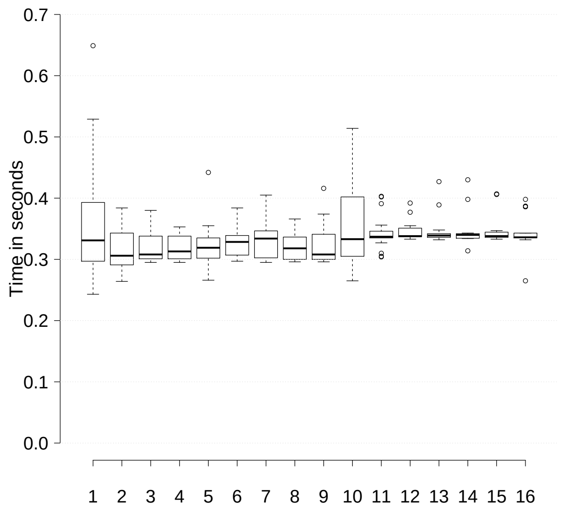


Fig. 18. IPFS experiment in RPKI with 4 nodes. One node is acting as the Trust Anchor and repository. The other nodes fetch from the repository every minute. Fetch duration is plotted on the Y-axis against the current fetch iteration.

once, by another node, it would remain available within the network. In the case of RRDP, if caching infrastructure like CDNs is not in use, once the RPKI Repository goes offline, its contents become unavailable to RP software.

Our study of IPFS also reveals that it cannot guarantee atomic updates while RRDP can. IPFS synchronizes contents over the network; hence updates are eventually consistent. This could lead to a situation where RP software has an inconsistent view of the repository contents. RRDP, on the other hand, specifies that the link to its Snapshot file, which represents the current state of the Repository, must only be published after update to the Repository has been completed [1]. This requirement in RRDP ensures atomic updates and prevent RP software from having an inconsistent view of the RPKI Repository.

B. Experiments

In our experiments, we saw trends suggesting that bandwidth is not the bottleneck in the performance of IPFS. We came to this conclusion by comparing the performance reduction for IPFS with HTTPS when the same network

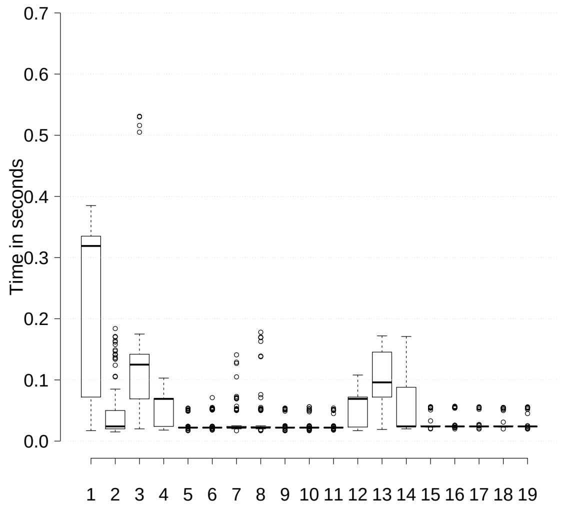


Fig. 19. HTTPS experiment in RPKI with 10 nodes. One node is acting as the Trust Anchor and repository. The other nodes fetch from the repository every minute. Fetch duration is plotted on the Y-axis against the current fetch iteration.

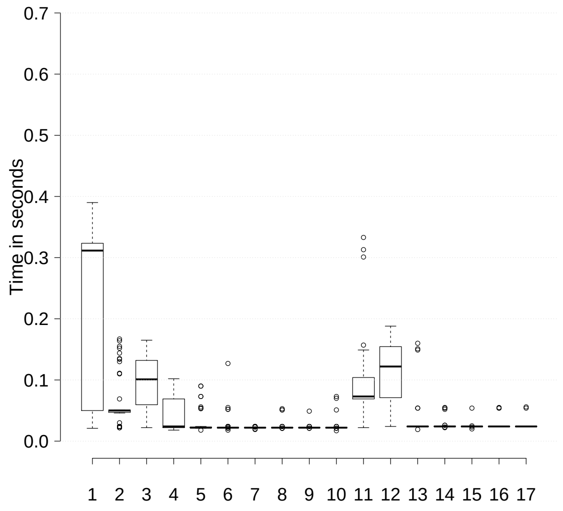


Fig. 20. HTTPS experiment in RPKI with 4 nodes. One node is acting as the Trust Anchor and repository. The other nodes fetch from the repository every minute. Fetch duration is plotted on the Y-axis against the current fetch iteration.

parameters are reduced. In the case of bandwidth, IPFS suffered only a 22% reduction in overall retrieval duration, whereas, in the case of HTTPS, it incurred a reduction of 704.4%. In our experiments with increased latency, we observed that in the case of IPFS, only the first node to retrieve the data incurs a penalty of 224%. The second node sees a minor improvement of 10%, and all other nodes incur a small penalty of 10%. HTTPS suffered an average retrieval time increase of 259%. This suggests that a latency increase may act as a bottleneck for the first node to retrieve the data in IPFS, but that this penalty is only incurred once as all other nodes may fetch data from the first node and so forth. Note that the latter nodes still fetch some data from the node with poor connectivity, but this penalty is less harsh than it is for the first node. In HTTPS, of course, all nodes suffer a similar penalty in relation to each other because of the client-server architecture.

The data sharing with peers that we observed in the IPFS latency test is likely to be less effective when all nodes attempt to retrieve the data concurrently because, at that point in time, only the node with poor connectivity will have the data that is being requested. Since we downloaded data sequentially in our experiment, all nodes after the first node to retrieve data will have one or more nodes with good connectivity that can provide data to them. Additionally, it is unclear what the cause is behind the second node to retrieve the data to perform better in the increased latency experiment. One of the possible causes we considered was the way IPFS attempts to retrieve a file. According to the documentation of IPFS Bitswap[58], a node will attempt to query every peer for the content it is looking for. If none of the peers have the requested content, the node will use the Distributed Hash Table. There is no clarification on the order in which a node will query all of its peers. If we assume a uniform distribution of chance that a node may be asked to provide a block of data to the requesting node, the second node will request more data from the node with bad connectivity than the third node would, and so forth. Our results suggest that this is not the case; however, as we do not see a descending line in retrieval duration for each subsequent node in Figure 12.

Our results for the direct HTTPS and IPFS comparison, and RPKI comparison experiments suggest that having more nodes in the IPFS swarm may cause higher variation in download durations. When we increased the number of nodes in the network, the variation between download times increased. We consider this variance to be noteworthy within the context of the RTR protocol, where VRPs are kept in a cache to be made available to BGP speakers. Different deployment scenarios are possible within RTR. One of which is to have a BGP speaker make use of multiple VRP caches. If the RP software is using IPFS, then the VRP cache they produce will be influenced by the variance we observed. It is worth adding that a strict synchronization should not be a goal, as mentioned in Section 8 of RFC 6810 [59], caches cannot be in strict synchrony due to the distributed nature of RPKI. This observed variance is even less consequential

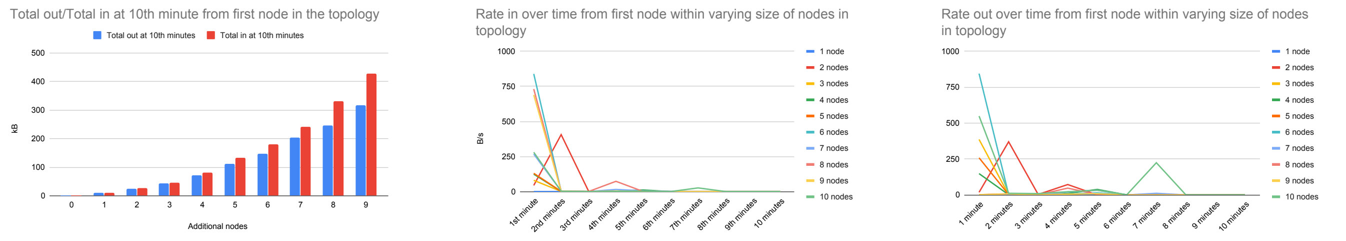


Fig. 21. Total Out/Total In, Rate in, Rate out for first node (bootstrap node).

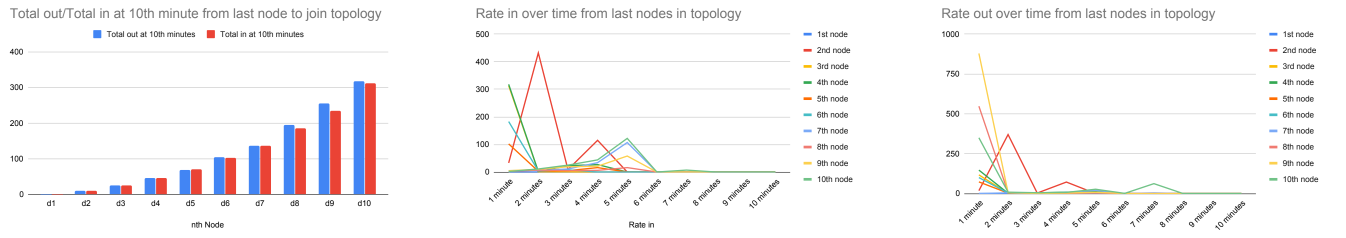


Fig. 22. Total Out/Total In, Rate in, Rate out for last node.

when considered within the global RPKI as different RP software will have different update times they use in polling repositories for new RPKI material. This update frequency may range from the recommendation in RFC 8182 [1] of at most once every minute to any other arbitrary value considered appropriate by the operator of the RP software, preferably at different times than other RPs to avoid peaks in server and network load.

We also see that IPFS comes with a network overhead. We saw that even when data is not being transmitted between the nodes, there is still network activity. This is due to the fact that IPFS is a peer-to-peer technology; hence there has to be communication between the nodes to keep the network operational. The DHT, which is used for peer and content discovery, is one aspect that leads to network activities. We see that in a static topology, as is the case in our experiment, where the number of nodes does not change, over time, the network activity reduces and finally stops, indicating an information equilibrium has been reached. In a real-life scenario where nodes are connected to the global IPFS network, we do not expect to see the network overhead stop, as the global IPFS network has a dynamic topology with nodes continually joining and leaving. We can also see that the network overhead increases as the number of nodes increases. Since we did not make a repeated sampling of the measurements, we cannot make any definite statement about the rate of increase and relationship between the observed network overhead and the number of nodes. More experiments would have to be designed to study this. We can also see that the network overhead observed for the first node, which is the bootstrap node is higher than the last node in the topology. Also, the first node has its incoming network activity higher than its outgoing. This observation can be explained by the fact that the first node needs to accept a higher number of incoming requests by the very nature of it being the bootstrapping node.

VII. Conclusion

At the beginning of this research, we set out to understand to what extent IPFS can be used as a distribution Mechanism for RPKI Repository. To do this, we first had to know how RPKI, IPFS, and RRDP works. We also performed various experiments which gave us insights into the network characteristics of IPFS. The result from the tests we performed allowed us to expound on the potential operational effects IPFS could have when used as a distribution mechanism within IPFS.

In this research, we were able to show that it is possible to modify existing RPKI CA software and RPKI RP software to make use of IPFS. Our research shows that one approach to introduce IPFS into RPKI is by augmenting current RRDP implementation. Another approach would be replacing RRDP by creating a new protocol solely based on IPFS.

Regarding networking, we observed that IPFS shows characteristics of peer-to-peer networks. Content once retrieved is shared with other nodes making it possible to pay the price of retrieving contents from a destination with poor network connectivity only once. Since after initial retrieval, nearby nodes no longer necessarily have to fetch the content from the same node with poor network connectivity. Another consequence of being a peer-to-peer network is that IPFS comes with a network overhead. This network overhead is because nodes need to keep their portion of the DHT and also keep communicating with each other to keep the network operational.

IPFS data retrieval time performs poorly relative to HTTPS, and IPFS makes less optimal use of available bandwidth. These observations of IPFS network characteristics are in line with the findings made in [18].

IPFS is a peer-to-peer, distributed network and its usage of ideas like content addressing, imbues it with properties that make some of the implemented features found in RRDP redundant, but IPFS network and IO performance are below par in comparison with HTTPS used within RRDP. One feature of IPFS does show promise, however, if its network and IO performance were better than in its current iteration. Namely, when the data originates from a node with bad connectivity, it will become available on other nodes that have downloaded this data. These nodes may have better connectivity and improve accessibility for other nodes that are looking to download the same data, as we demonstrated in Figure 12. To achieve this same effect using HTTPS, the party that is hosting the data would have to invest in a CDN infrastructure, for example.

VIII. Future Work

The results of the qualitative analysis can serve as the basis for future work. Prototypes can be created based on the identified ways IPFS can augment or replace features within the current implementations of the RPKI repository.

Krill and Routinator were modified to use IPFS binaries installed on the host operating system. This approach to integrating IPFS into these two libraries was due to the IPFS Rust library [60] not being matured enough at the time of this research. It would be interesting to modify both Krill and Routinator with IPFS using a library instead of the current approach of calling to an installed binary and observe if there is any noticeable difference in operational performance. This modification can only be done once the IPFS Rust library becomes mature enough to integrate a full-featured IPFS node into a Rust application.

In this research, we took a look at the bandwidth consumption of an idle IPFS node. It would be interesting to make similar bandwidth measurements with nodes that are actively participating in data transfers. Such measures can serve as another approach to better understand the peer-to-peer network overhead of IPFS.

In the network topology described in IV-B, we only introduced delays between the switch and the node initially hosting the contents. Future research could introduce similar network delays in other parts of the topology and observe the effect. Our research showed a glimpse of the effect that a node with bad connectivity has on the IPFS network, to continue this research with more complex topologies will produce data necessary to construct an efficient network. For example, it may prove or disprove the benefits of clustering nodes into groups based on their connectivity to each other to speed up file transfers.

Efficient energy use could also be the basis for future research, in which power consumption when using IPFS within the RPKI repository is compared with the use of rsync or RRDP. We perceive resource usage in a peer-to-peer system to have an influence on its adoption rate, which is why a study of this sort could give meaningful insights.

IX. Acknowledgements

We would like to thank the NLnet Labs team for having us for the duration of this research. Special thanks to our supervisor, Luuk Hendriks, for his guidance, insights, and support. Also, we would like to appreciate Martin Hoffmann for all the technical support provided.

References

[1] T. Bruijnzeels et al. *The RPKI Repository Delta Protocol (RRDP)*. RFC 8182. RFC Editor, July 2017.

[2] Juan Benet. "IPFS - Content Addressed, Versioned, P2P File System". In: *CoRR* abs/1407.3561 (2014). arXiv: 1407.3561. URL: http://arxiv.org/abs/1407.3561.

[3] B. Ahlgren et al. "A survey of information-centric networking". In: *IEEE Communications Magazine* 50.7 (2012), pp. 26–36.

[4] *First release of rsync - rcp replacement*. URL: https://groups.google.com/forum/#!msg/comp.os.linux.announce/tZE1qtTcQaU/IF8GhGQ_uTsJ (visited on 06/02/2020).

[5] *Efficient Algorithms for Sorting and Synchronization*. URL: https://www.samba.org/~tridge/phd_thesis.pdf (visited on 05/29/2020).

[6] *The rsync algorithm*. URL: https://rsync.samba.org/tech_report/ (visited on 05/29/2020).

[7] V. Kotlyar et al. "Torrent Base of Software Distribution by ALICE at RDIG". In: (2012), pp. 171–175.

[8] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. "A Survey of Peer-to-Peer Content Distribution Technologies". In: *ACM Comput. Surv.* 36.4 (Dec. 2004), 335–371. ISSN: 0360-0300. DOI: 10.1145/1041680.1041681. URL: https://doi.org/10.1145/1041680.1041681.

[9] B. Confais, A. Lebre, and B. Parrein. "An Object Store Service for a Fog/Edge Computing Infrastructure Based on IPFS and a Scale-Out NAS". In: *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. 2017, pp. 41–50.

[10] Sihua Wu and Jiang Du. "Electronic medical record security sharing model based on blockchain". In: *Proceedings of the 3rd International Conference on Cryptography, Security and Privacy*. 2019, pp. 13–17.

[11] R. Norvill et al. "IPFS for Reduction of Chain Size in Ethereum". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pp. 1121–1128.

[12] Q. Zheng et al. "An Innovative IPFS-Based Storage Model for Blockchain". In: *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. 2018, pp. 704–708.

[13] Bruno Produit. *Using blockchain technology in distributed storage systems*. 2018.

[14] *Netflix Inc - Company Profile and News - Bloomberg Markets*. URL: https://www.bloomberg.com/profile/company/NFLX:US (visited on 05/29/2020).

[15] *New improvements to IPFS Bitswap for faster container image distribution*. URL: https://blog.ipfs.io/2020-02-14-improved-bitswap-for-container-distribution/ (visited on 05/28/2020).

[16] *Docker Hub*. URL: https://hub.docker.com/ (visited on 05/28/2020).

[17] *A container management platform that provides scalable and reliable container execution and cloud-native integration with Amazon AWS*. URL: https://netflix.github.io/titus/ (visited on 05/28/2020).

[18] J. Shen et al. "Understanding I/O Performance of IPFS Storage: A Client's Perspective". In: *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*. 2019, pp. 1–10.

[19] B. Confais, A. Lebre, and B. Parrein. "Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures". In: *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2016, pp. 294–301.

[20] *RADOS Object Store — Ceph documentation*. URL: https://docs.ceph.com/docs/bobtail/rados/ (visited on 06/26/2020).

[21] *Apache Cassandra*. URL: https://cassandra.apache.org/ (visited on 06/26/2020).

[22] Onur Ascigil et al. "Towards Peer-to-Peer Content Retrieval Markets: Enhancing IPFS with ICN". In: *Proceedings of the 6th ACM Conference on Information-Centric Networking*. ICN '19. Macao, China: Association for Computing Machinery, 2019, 78–88. ISBN: 9781450369701. DOI: 10.1145/3357150.3357403. URL: https://doi.org/10.1145/3357150.3357403.

[23] Ethan Heilman et al. "From the Consent of the Routed: Improving the Transparency of the RPKI". In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2014. ISBN: 9781450328364. DOI: 10.1145/2619239.2626293. URL: https://doi.org/10.1145/2619239.2626293.

[24] Hitesh Ballani, Paul Francis, and Xinyang Zhang. "A Study of Prefix Hijacking and Interception in the Internet". In: *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '07. Kyoto, Japan: Association for Computing Machinery, 2007, 265–276. ISBN: 9781595937131. DOI: 10.1145/1282380.1282411. URL: https://doi.org/10.1145/1282380.1282411.

[25] M. Lepinski, S. Kent, and D. Kong. *A Profile for Route Origin Authorizations (ROAs)*. RFC 6482. RFC Editor, Feb. 2012.

[26] R. Austein et al. *Manifests for the Resource Public Key Infrastructure (RPKI)*. RFC 6486. RFC Editor, Feb. 2012.

[27] G. Huston et al. *Resource Public Key Infrastructure (RPKI) Trust Anchor Locator*. RFC 8630. RFC Editor, Aug. 2019.

[28] *ipfs/specs: Technical specifications for the IPFS protocol stack*. URL: https://github.com/ipfs/specs (visited on 06/27/2020).

[29] *IPFS Implementations*. URL: https://github.com/ipfs-implementations (visited on 05/21/2020).

[30] *ipfs/go-ipfs: IPFS implementation in Go*. URL: https://github.com/ipfs/go-ipfs (visited on 06/27/2020).

[31] *The Go Programming Language*. URL: https://golang.org/ (visited on 06/27/2020).

[32] *Content addressing | IPFS Docs*. URL: https://docs.ipfs.io/concepts/content-addressing/#identifier-formats (visited on 07/01/2020).

[33] *IPLD - The data model of the content-addressable web*. URL: https://ipld.io/ (visited on 06/27/2020).

[34] *ipld/specs: Content-addressed, authenticated, immutable data structures*. URL: https://github.com/ipld/specs (visited on 06/27/2020).

[35] *ipfs/go-ipfs-chunker: go-ipfs-chunkers provides Splitter implementations for data before being ingested to IPFS*. URL: https://github.com/ipfs/go-ipfs-chunker (visited on 06/28/2020).

[36] *libp2p*. URL: https://libp2p.io/ (visited on 07/02/2020).

[37] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65. ISBN: 978-3-540-45748-0.

[38] *ipfs/specs: Technical specifications for the IPFS protocol stack*. URL: https://github.com/ipfs/specs (visited on 06/28/2020).

[39] *specs/peer-ids.md at master · libp2p/specs*. URL: https://github.com/libp2p/specs/blob/master/peer-ids/peer-ids.md (visited on 07/01/2020).

[40] *IPNS | IPFS Docs*. URL: https://docs.ipfs.io/concepts/ipns/ (visited on 07/02/2020).

[41] *NLnet Labs - RPKI Tools - Krill*. URL: https://www.nlnetlabs.nl/projects/rpki/krill/ (visited on 06/27/2020).

[42] *NLnet Labs - RPKI Tools - Routinator*. URL: https://www.nlnetlabs.nl/projects/rpki/routinator/ (visited on 06/27/2020).

[43] *Overview | Containernet*. URL: https://containernet.github.io/ (visited on 06/27/2020).

[44] *Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet*. URL: http://mininet.org/ (visited on 06/27/2020).

[45] *Empowering App Development for Developers | Docker*. URL: https://www.docker.com/ (visited on 06/28/2020).

[46] *Benchmark of HTTPs and IPFS*. URL: https://github.com/sne-os3-rp2/ipfs_http_benchmark (visited on 06/27/2020).

[47] O. Tange. "GNU Parallel - The Command-Line Power Tool". In: *;login: The USENIX Magazine* 36.1 (Feb. 2011), pp. 42–47. DOI: http://dx.doi.org/10.5281/zenodo.16303. URL: http://www.gnu.org/s/parallel.

[48] *NLnetLabs/rpki-rs: An RPKI library for Rust*. URL: https://github.com/NLnetLabs/rpki-rs (visited on 06/28/2020).

[49] *sne-os3-rp2/rpki-rs: An RPKI library for Rust*. URL: https://github.com/sne-os3-rp2/rpki-rs (visited on 06/28/2020).

[50] *sne-os3-rp2/krill: RPKI Certificate Authority and Publication Server written in Rust*. URL: https://github.com/sne-os3-rp2/krill (visited on 06/28/2020).

[51] *Docker image of Krill modified to use IPFS*. URL: https://hub.docker.com/repository/docker/dadepo/krill-ipfs (visited on 06/28/2020).

[52] *sne-os3-rp2/routinator: An RPKI Validator written in Rust*. URL: https://github.com/sne-os3-rp2/routinator (visited on 06/28/2020).

[53] *Docker image of Routinator modified to use IPFS*. URL: https://hub.docker.com/repository/docker/dadepo/routinator-ipfs (visited on 06/28/2020).

[54] *Overview of Docker Compose | Docker Documentation*. URL: https://docs.docker.com/compose/ (visited on 06/28/2020).

[55] *sne-os3-rp2/lab: Scripts, and Docker build files for creating Docker compose file that is to be used to orchestrate Krill and routinator instances for experiments purposes*. URL: https://github.com/sne-os3-rp2/lab (visited on 06/28/2020).

[56] *Docker image of Routinator modified to accept invalid TLS certificates*. URL: https://hub.docker.com/repository/docker/dadepo/lenient-ssl-routinator (visited on 06/28/2020).

[57] *CLI commands | IPFS Docs | IPFS Stats Bandwidth*. URL: https://docs.ipfs.io/reference/cli/#ipfs-bitswap-stat (visited on 07/02/2020).

[58] *Bitswap | IPFS Docs*. URL: https://docs.ipfs.io/concepts/bitswap/ (visited on 07/04/2020).

[59] R. Bush and R. Austein. *The Resource Public Key Infrastructure (RPKI) to Router Protocol*. RFC 6810. RFC Editor, Jan. 2013.

[60] *rs-ipfs/rust-ipfs: The Interplanetary File System (IPFS), implemented in Rust*. URL: https://github.com/rs-ipfs/rust-ipfs (visited on 07/01/2020).