IIT
University of Dhaka

Software Requirement Specification Report

V 1.0


Prepared by:

**Tahlil, ROLL: bsse0803**

Supervised by:

***Dr. B. M. Mainul Hossain***
***Associate Professor,***
***Institute of Information Technology***



# SRS
for
# A blockchain-based web hosting scheme for content-addressed distributed file system.

PiperNet

Submitted to:

Coordinator

Ahamedul Kabir,
Assistant Professor,
IIT,DU

Course: SE801 **Software Project Lab 3**

**Institute of Information Technology**

**University of Dhaka**

**01-09-2019**

# Software Requirement Specification Report: A blockchain-based web hosting scheme for content-addressed distributed file system

# Table of Contents

# Software Requirement Specification Report: A blockchain-based web hosting scheme for content-addressed distributed file system

# SRS Report

## A blockchain-based web hosting scheme for content-addressed distributed file system

# Chapter 01 Introduction

This chapter describes the purpose of the SRS document, the convention followed throughout the document, the intended audience, the project scopes and references.

## 1.1 Purpose

Identify the broad and specific purposes of the document

## 1.2 Document Conventions

Discusses any standards or typographical conventions followed

## 1.3 Intended Audience and Reading Suggestions

## 1.4 Product Scope

Discusses short description of the software being specified

## 1.5 References

List of documents or Web addresses to which this SRS refers

# 01

## 1.1

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

# Introduction

Purpose

## 1.1 Purpose

This Software Requirement Specification (SRS) document is intended to be prepared for building the project proposed in the course SE801 SPL3(Software project lab 3) which is entitled "**Pipernet**: A block-chain-based web hosting scheme for content-addressed distributed file system". The end product of this project is basically a peer to peer web service for website hosting and file sharing. The project is inspired by the project by Protocol Labs(website link: https://protocol.ai/): the popular open source peer-to-peer hypermedia protocol for distributed file sharing IPFS(Inter Planetary File System). The SRS document starts by setting clear objectives to leverage the power of decentralization and blockchain to overcome the limitation of the centralized web architecture. It establishes what is within the scope of the project and which is not. Then it specifies all the required sub-components to be implemented and all the algorithms and protocols involve in each sub-components with detailed inspection. It also set the milestones to be achieved from different aspects like functionality, developer's library, security and network scalability. The descriptions are complemented by visual models and diagrams for better understanding of the individual complex sub-components.It states the requirements of the project in two sub categories: one is from the user perspective and other is from a developer perspective. A user characteristics analysis is done to understand different the use cases. It identifies the deliverables from a software perspective. Furthermore, it briefly describes the environment set up and minimum system requirements to run on different platforms. Mock-up user interfaces are designed for different platform like mobile, web and desktop, to give a comprehensive idea about how to use the system. On later segment, The document defines the measurable software quality attributes to evaluate the system on an objective-basis. Finally, further use cases of the system is discussed in the "future scope" segment of the SRS report.

01

1.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

# Introduction

Document Conventions

## 1.2 Document Conventions

When writing the SRS document for "blockchain-based web hosting scheme for content-addressed distributed file system" the following abbreviations are used:

| SL. | Abbreviation | Full form |
|---|---|---|
| 1 | P2P(network) | Peer to peer (network) |
| 2 | IPFS | Interplanetary File System |
| 3 | IPLD | Interplanetary Link Data |
| 4 | IPNS | InterPlanetary Naming System |
| 5 | SFS | Self-certifying file system |
| 6 | PN | Provider node |
| 7 | CN | Consumer node |
| 8 | DAG | Directed acyclic graph |
| 9 | DHT | Distributed hash table |
| 10 | HTTP(prtocol) | Hyper Text Transfer Protocol |
| 11 | HTML | Hyper text markup language |
| 12 | IP address | Internet Protocol address |
| 13 | POS | Proof of storage |
| 14 | DBFT | Delegated Byzantine Fault Tolerance |
| 15 | DDoS (attack) | Distributed denial of service (attack) |
| 16 | DNS | Domain Name Server |
| 17 | CID | Content Identifier |
| 18 | PKI | Public Key Infrastructure |
| 19 | LBFS | Last Buffer First Served |

*Table 1: List of abbreviations*

A detailed list of terminology used throughout the document is presented in the Appendix A: Glossary section. To make the document more effective and readable verdana font style is used. The section and sub-section headings highlighted with bold text and italicized text is used to label and identify diagrams and tables.Code snippets used in the document is of javascript language. File object formats used is JSON. Note that this structure is actually binary encoded using protobufs, though PiperNet includes import/export to JSON.

# 01

1.3 1.4

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

# Introduction

Intended Audience
and
Reading Suggestion

Product Scope

## 1.3 Intended Audience and Reading Suggestions

This document was prepared as a part of Bachelor's of Science in Software Engineering(BSSE) 8th semester course SE801, Softwre project lab 3.

For implementation purposes as a developer, reader can go thorugh the chapter 2 . For users of the system chapter 3 and chapter 4 should be the focus.

## 1.4 Product Scope

Pipernet is an application to transfer, share, stream files and web hosting scheme in a decentralized network.

### 1.4.1 What's in the scope

i. Build a distributed file system on the internet following the principles of IPFS.

i. Develop a decentralized web-hosting scheme.

ii. Upload, download, share file of any type.

iii. Use blockchain-based file structure to ensure secure file transfer, secure web-surfing(searching), secure user identification using cryptographic tools.

iv. Content-based addressing instead of the traditional location-based addressing when it comes to surfing decentralized web platform.

v. Ensure proper anonymity by hiding IP of the web-surfer.

vi. Mobile(android & IOS), desktop (windows, mac and linux) and all modern browser(Details in later section) platform to interact with the system.

vii. Display and play media content of the most widely adopted formats(Details in later section).

viii. Store and display history and seeding info.

### 1.4.2 What's not in the scope

i. Support for old browser i. e. Internet explorer prior to 11.0, older version of google chrome, opera, firefox, maxhathon, torch etc.

ii. Estimate completion of download, upload, share time.

iii. Monitor state of the network in a visual way.

01

1.3 1.4

Project name:

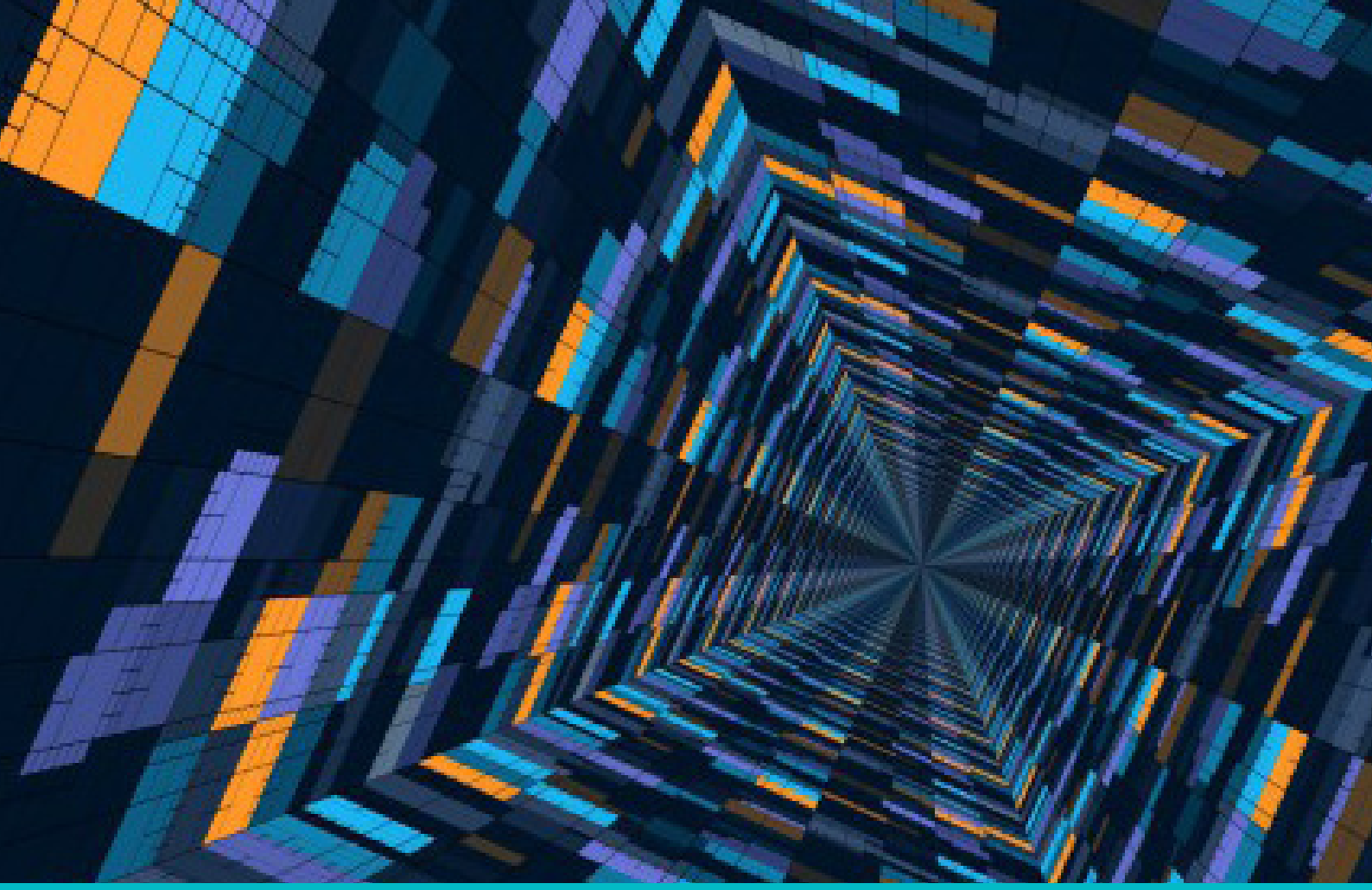
**A blockchain-based web hosting scheme for content-addressed distributed file system**

# Introduction

Reference

## 1.6 References


1. IPFS White Paper: Juan Benet, “IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)” [Online] Available: https://ipfs.io/ipfs/QmV9tSDx9UiPeWExXEeH6aoDvmihvx6jD5eL-b4jbTaKGps
2. IEEE Template for System Requirement Specification Documents: “IEEE Software Requirements Specification Template”, https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc
3. Piper Net Github Page: https://github.com/Tahlil/PiperNet
4. Libp2p: https://medium.com/coinmonks/understanding-ipfs-in-depth-5-6-what-is-libp2p-f8b-f7724d452
5. IPLD: https://hackernoon.com/understanding-ipfs-in-depth-2-6-what-is-interplanetary-linked-data-ipld-c8c01551517b
6. IPNS: https://hackernoon.com/understanding-ipfs-in-depth-3-6-what-is-interplanetary-naming-system-ipns-9aca71e4c13b
7. Authentication: https://medium.com/blockchain-blog/blockchain-based-authentication-of-devices-and-people-c7efcfcf0b32
8. Application Layer: “A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems “ [Online] Available: https://ieeexplore.ieee.org/abstract/document/8400511



# SRS Report

## A blockchain-based web hosting scheme for content-addressed distributed file system

# Chapter 02

# Overall Description

This chapter describes the purpose of the SRS document, the convention followed throughout the document, the intended audience, the project scopes and references.

## 2.1 Project Perspective

Identify the broad and specific purposes of the document

## 2.2 Document Conventions

Discusses any standards or typographical conventions followed

## 2.3 Intended Audience and Reading Suggestions

## 2.4 Product Scope

Discusses short description of the software being specified

## 2.5 References

List of documents or Web addresses to which this SRS refers

# 02

## 2.1, 2.2

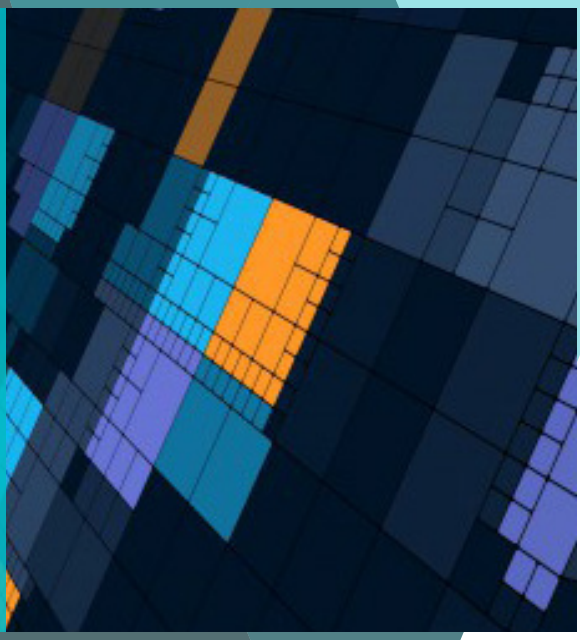
Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

# Overall Description

Project perspective

Project functions



## 2.1 Project Perspective

Pipernet was developed to integrate the best of peer to peer technologies and technologies invented after the emergence of http, the de-facto protocol for the current Internet. The underlying protocol aims to mitigate the problem of http . Primarily, it achieves this A decentralized architecture is adopted to solve the various problem a centralized problem face, but the protocol is designed to integrate with the current Internet .On an application level it focuses on secured download, upload, sharing of file and browsing web content through the internet using any device whether through a browser or application. So, Pipernet is a peer-to-peer file sharing and web-hosting application..

## 2.2 Product Functions

User information management related functions:

- ◇ Create an user account to a specific device.
- ◇ Log in to user account for private usage.
- ◇ Search user with their name.
- ◇ Update user info.
- ◇ Change user password.
- ◇ Add multiple user to same device.
- ◇ Synchronize user to multiple device.
- ◇ Manage device info.
- ◇ Give notification when logged in from other devices.

Decentralized file system related functions:

- ◇ Upload a file.
- ◇ Download a file.
- ◇ Search for a file.
- ◇ Share a file with a specific user.
- ◇ Rename a file.

# 02

## 2.3, 2.4

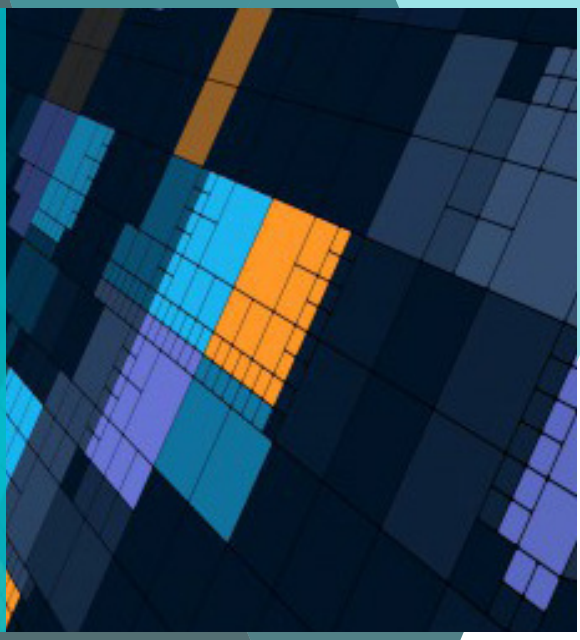
Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

# Overall Description

User Classes and Characteristics

Frameworks and Tools



◇ Delete a file.

Decentralized web hosting related functions:

◇ Host a website.

◇ Visit a hosted site.

◇ Browse for sites.

◇ Update website info(keywords)

## 2.3 User Classes and Characteristics

» Typical User, who wants to use it like a normal torrent application uploading and downloading content, or share file with their acquaintances or hosing personal site.

» Service provider who wants to serve their site or any kind of file to the public and cut cost on maintaining their own server.

» Programmers who are interested in working on this open-source project by further developing fix existing bugs and adding more features.

## 2.4 Frameworks and Tools

The whole application will be developed using javascript frameworks only.


Front: Angular version 8

Back: Node JS version 10

Front: Angular version 8

Back: Ionic version 5



Front: Angular version 8

Back: Electron JS version 6

02

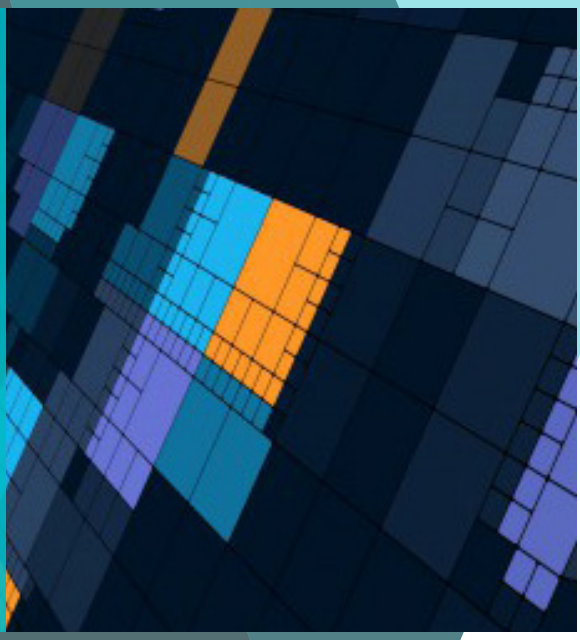
2.5

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

# Overall Description

Operating Environment



Version Control system: GIT GitHub

IDE(Integrated Development Environment): VS Code

## 2.5 Operating Environment

Piper Net can operate in the browser, desktop or in mobile phone

### Desktop

Windows

1. Windows 8
2. Window 10 Home
3. Windows 10 Pro
4. Windows XP

Linux

1. Ubuntu 12.04 +
2. Fedora 21
3. Debian 8
4. Linux mint
5. Kali Linux

Mac

1. Mac Sierra
2. Mac High Sierra
3. Mac Catalina
4. Mac Mojave

### Browser

Firefox based browser: (**Firefox, Seamonkey** ) **v44.0+**

Chrome based browser: (**Chrome, Vivaldi, Brave, Torch**) **v49.0+**

Opera: **v36.0+** Opera Mobile: **v46.0+**

Safari: **v11.0+** iOS Safari: **v11.0+**

Edge: **v12.0+** IE & IE mobile: **v11.0**

Android browser: **v76.0+** Samsung internet: **v5.0+**

Chrome for Android: **v76.0**

Firefox for Android: **v68.0**

UC Browser for Andoid: **v12.12+**

*LATEST VERSION OF OTHER BROWSER LIKE MAXATHON, KALI-BROWSER, YANDEX WOULD ALSO WORK

### Mobile

Android

Min:
**Android 4/ice cream sandwitch**
upto: **Android 9/Pie**

IOS

**IOS Version 10+**

# 02

2.6, 2.7, 2.8

Project name:

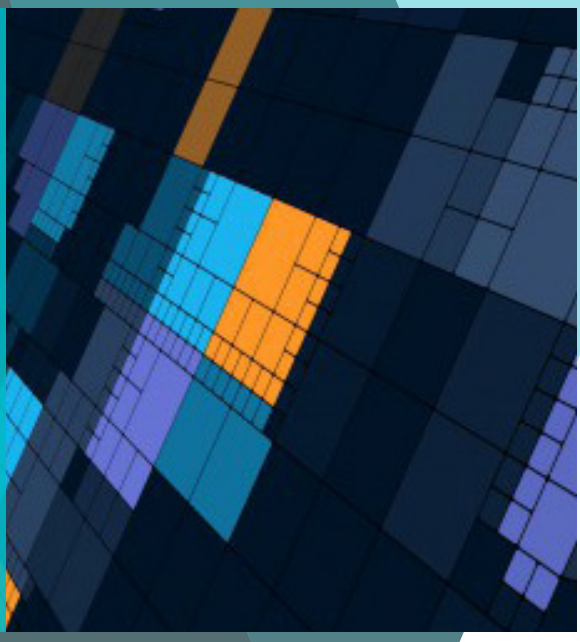
**A blockchain-based web hosting scheme for content-addressed distributed file system**

# Overall Description

Design and Implementation Constraints

User Documentation

Assumptions and Dependencies



## 2.6 Design and Implementation Constraints

Pipernet wiil be written entirely with javascript and its framework angular, iocnic, electron. To use these framework node js and node package manager (npm) must be installed in the development machine. For running the browser application in development mode any browser need to be installed. Furthermore, for running mac application and IOS application in development mode a macintosh machine is required and similarly at least an ubuntu linux is required to run for confirming it will run in linux distros. For these purpose the development machines used will be a windows machine and a Ubuntu 18.0 machine with Mac Sierra installed in virtual box.

## 2.7 User Documentation

To learn how the underlying protocol works one can refer to the below IPFS documention page: https://docs.ipfs.io/introduction/how-ipfs-works/
The complete user manual will be found in:
https://github.com/Tahlil/PiperNet/tree/master/UserManual

## 2.8 Assumptions and Dependencies

To get the network up and running at least two user is to be assumed to be registered to start getting the service.
Software components that would be used in the project:

- Node package: https://www.npmjs.com/
- Ionic project: https://ionicframework.com/
- Angular project: https://angular.io/
- Elctron project: https://electronjs.org/
- Libp2p package: https://libp2p.io/

# Chapter 3

## Pipernet layers

Pipernet protocol stack follow the mainl principals of IPFS (Inter Planetary File System)

A self-certifying PKI namespace

Data structure format: Merkle Direct Acyclic Graph

5 layers of PiperNet Protocol



**3.1**
Naming Layer

**3.2**
Merkle DAG

Higher level

Block transport and replication protocols

Locating peers and objects in the network

Establishing connections between peers




**3.3**
Exchange Layer

**3.4**
Routing Layer

**3.5**
Network Layer

# 03

## 3.1

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

## 3.1 Naming layer

The main protocol for the naming layer is called **IPNS**(Inter Planetary Naming System).It is basically the alternative to the protocol used in a DNS (Domain Name Server) to be used in a peer-to-peer network like Pipernet.

### 3.1.1 WHY IPNS

When we store file in Pipernet the file are addressed as hash for content-addressing for security reason. So, for example one address of a file could be "QmYVd8qstdXtTd1quwv4nJen6XprykxQRLo67Jy7WyiLMB".

But this kind of addressing is inconvinient for a couple of reasons:

- ◇ Firstly, it's hard to read, let alone remember.
- ◇ Secondly, it's an immutable link. What is meant by an immutable link is that this link is permanent(due to the nature of content-addressing). If one were to add even a comma anywhere in the file s/he uploaded or website s/he hosted, the CID of the root folder will change, thus changing the link to the file/website. So, every time s/he change anything on the file/website, I have to give the new link to everyone who wants to access the latest version of the file/website.

### 3.1.2 IPNS Desciption

IPNS generate a mutable link, which:

1. Is human-readable and easy to remember.
2. points to the latest version of your website, profile photo, video etc.

A name in IPNS(the hash follows /ipns/ in a link) is the hash of a public key. It is associated with a record containing information about the hash it links to that is signed by the corresponding private key. New records can be signed and published at any time.

So, in other words, IPNS is a global namespace based on Public Key Infrastructure (or PKI) which allows us to build trust chains (so you can follow a public key to its route peer), giving us encryption and authentication, and is still actually compatible with other name services. So for example, we can even map things like DNS entries, onion, or bit addresses, etc. to IPNS addresses.

# 03

## 3.1

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

### 3.1.3 DNSLink

In top of IPNS another protocol is implemented called DNSLink to make links even more readable. DNSLink uses DNS TXT records to map a domain name (like pipernet.io) to an hashed address. Because one can edit his/her DNS records, s/he can use them to always point to the latest version of an object in PiperNet (object's address changes if one modify the object). Because DNSLink uses DNS records, the names it produces are also usually easy to type and read. A DNSLink address looks like an IPNS address, but it uses a domain name in place of a hashed public key. DNSLink records using a special subdomain named _dnslink. This is useful when one want to improve the security of an automated setup or delegate control over your DNSLink records to a third-party without giving away full control over the original DNS zone. If /ipns/<domain> is a valid domain name, IPFS looks up key ipns in its DNS TXT records. IPFS interprets the value as either an object hash or another IPNS path:

```
# this DNS TXT record
ipns.tahlil.dev TXT "ipfs=XLF2ipQ4jD3U ..."
# behaves as symlink
ln -s /ipns/XLF2ipQ4jD3U /ipns/fs.tahlil.dev
```

### 3.1.4 Object Identity

Nodes are identified by a NodeId, the cryptographic hash of a public-key, created with S/Kademlia's static crypto puz-

zle. Nodes store their public and private keys (encrypted

with a passphrase). For every new user a new node is instantiated with their credentials. The structure of the node is as follows:

```
// self-describing keys
Node = {
      NodeId: NodeID
      PubKey: PublicKey
      PriKey: PrivateKey
}
```

# 03

## 3.1

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

S/Kademlia based IPFS identity generation:

```
difficulty = <integer parameter>
n = Node{}
do {
n.PubKey, n.PrivKey = PKI.genKeyPair()
n.NodeId = hash(n.PubKey)
p = count_preceding_zero_bits(hash(n.NodeId))
} while (p < difficulty)
```

Upon first connecting, peers exchange public keys, and check: hash(other.PublicKey) equals other. NodeId not, the connection is terminated.
Rather than locking the system to a particular set of function choices, pipernet uses self-describing values. Hash digest values are stored in multihash format, which includes a short header specifying the hash function used, and the digest length in bytes. Example:

```
<function code><digest length><digest bytes>
```

This allows the system to (a) choose the best function for the use case (e.g. stronger security vs faster performance), and (b) evolve as function choices change in other words future proof the system. Self-describing values allow using different parameter choices compatibly.

### 3.1.5 SFS (Self-Certified Filesystems)

SFS(Self-Certified Filesystems) proposed compelling implementations of both (a) distributed trust chains, and (b) egalitarian shared global namespaces. SFS introduced a technique for building Self-Certied Filesystems: addressing remote filesystems using the following scheme:

```
/sfs/<Location>:<HostID>
```

where **Location** is the server network address, and:

```
HostID = hash(public_key || Location)
```

Thus the name of an SFS le system certies its server. The user can verify the public key oered by the server, negotiate a shared secret, and secure all trac. All

# 03

## 3.1

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

SFS instances share a global namespace where name allocation is cryptographic, not gated by any centralized body. Using SFS in Pipernet gives us a way to construct self-certfied names, in a cryptographically assigned global namespace, that are mutable. The way SFS is integrated in Pipernet is the following:

1. Object ID is declared in the following way:

```
NodeId = hash(node.PubKey)
```

2. We assign every user a mutable namespace at:

```
/ipns/<NodeId>
```

3. A user can publish an Object to this path Signed by the private key, say at:

```
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/
```

4. When other users retrieve the object, they can check the signature matches the public key and NodeId. This verifies the authenticity of the Object published by the user, achieving mutable state retrieval.

5. The IPNS (InterPlanetary Name Space) separate prefix is to establish an easily recognizable distinction between mutable and immutable paths, for both programs and human readers.

6. Because this is not a content-addressed object, publishing it relies on the only mutable state distribution system in Pipernet, the Routing system. The process is (1) publish the object as a regular immutable object, (2) publish its hash on the Routing system as a metadata value:

```
routing.setValue(NodeId, <ns-object-hash>)
```

7. Any links in the Object published act as sub-names in the namespace:

```
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/docs
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/docs/ipfs
```

8. As encouraged by SFS, users can link other users' Objects directly into their

# 03

## 3.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

own Objects (namespace, home, etc). This has the benet of also creating a web of trust (and supports the old Certicate Authority model):

```
# Alice links to bob Bob
ipns link /<alice-pk-hash>/friends/bob /<bob-pk-hash>
# Eve links to Alice
ipns link /<eve-pk-hash/friends/alice /<alice-pk-hash>
# Eve also has access to Bob
/<eve-pk-hash/friends/alice/friends/bob
# access Verisign certified domains
/<verisign-pk-hash>/foo.com
```

## 3.2 Merkle DAG layer

The PiperNet stack forms a peer-to-peer block exchange constructing a content-addressed DAG of objects. It serves to publish and retrieve immutable objects. For the PiperNet architecture, mutable data is necessary in the end, so it had to be build up an immutable Merkle DAG. The properties of PiperNet that fall out of the Merkle DAG: objects can be (a) retrieved via their hash, (b) integrity checked, (c) linked to others, and (d) cached indenitely. In a sense: "Objects are permanent." These are the critical properties of a high-performance distributed system, where data is expensive to move across network links. Object content addressing constructs a web with (a) signicant bandwidth optimizations, (b) untrusted content serving, (c) permanent links, and (d) the ability to make full permanent backups of any object and its references. The Merkle DAG, immutable content-addressed objects, and Naming, mutable pointers to the Merkle DAG, instantiate a dichotomy present in many successful distributed systems. These include the immutable objects and mutable references; and Plan9 ,the distributed successor to UNIX, with its mutable Fossil and immutable Venti filesystems. LBFS also uses mutable indices and immutable chunks.

### 3.2.1 IPLD(InterPlanetary Linked Data)

The software component used in IPFS to handle Merle DAG is called IPLD which would give a broad overview of the whole system.

IPLD is not just a part of the IPFS project, but a separate project in itself. To understand its significance in the decentralized world we have to understand the concept of Linked Data: The Semantic Web.

# 03

## 3.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

The Semantic Web or Linked Data is a term coined by Sir Tim Berners Lee in a seminal article published in Scientific American in 2001. Berners Lee articulated a vision of a World Wide Web of data that machines could process independently of humans, enabling a host of new services transforming our everyday lives. While the paper's vision of most web pages containing structured data that could be analyzed and acted upon by software agents has NOT materialized, the Semantic Web has emerged as a platform of increasing importance for data interchange and integration through the growing community implementing data sharing using international semantic web standards, called Linked Data.

There are many current examples of the use of semantic web technologies and Linked Data to share valuable structured information in a flexible and extensible manner across the web. Semantic Web technologies are used extensively in the life sciences to facilitate drug discovery by finding paths across multiple datasets showing associations between drugs and side effects via genes linked to each. The New York Times has published its vocabulary of approximately 10,000 subject headings developed over 150 years as Linked Data and will expand coverage to approximately 30,000 topic tags; they encourage the development of services consuming these vocabularies and linking them with other online resources. The British Broadcasting Corporation uses Linked Data to make content more findable by search engines and more linkable through social media; to add additional context from supplemental resources in domains like music or sports, and to propagate linkages and editorial annotations beyond their original entry target to bring relevant information forward in additional contexts. The home page of the United States data.gov site states, "As the web of linked documents evolves to include the Web of linked data, we're working to maximize the potential of Semantic Web technologies to realize the promise of Linked Open Government Data." And also all the social media websites use Linked Data to create a web of People to make their platform as engaging as possible.

So we do have some Linked Data but we still have a long way to go in order to harness the true power of Linked Data.

# 03

## 3.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

Now imagine if we could refer our latest commits in a git branch to a bitcoin transaction to timestamp your work. So, by linking your git commit, we can view the commit from your blockchain explorer. Or, if we could link your Ethereum contract to media on IPFS, perhaps modifying it and tracking its changes on each function execution. All this is possible using IPLD. IPLD is the data model of the content-addressable web. It allows us to treat all hash-linked data structures as subsets of a unified information space, unifying all data models that link data with hashes as instances of IPLD. Or in other words, IPLD is a set of standards and implementations for creating decentralized data-structures that are universally addressable and linkable. These structures allow us to do for data what URLs and links did for HTML web pages. Content addressing through hashes has become a widely-used means of connecting data in distributed systems, from the blockchains that run your favorite cryptocurrencies, to the commits that back your code, to the web's content at large. Yet, whilst all of these tools rely on some common primitives, their specific underlying data structures are NOT interoperable(as right now I can't connect my git commit to a blockchain transaction).

IPLD is a single namespace for all hash-inspired protocols. Through IPLD, links can be traversed across protocols, allowing you to explore data regardless of the underlying protocol. The sky's the limit as IPLD allows you to work across protocol boundaries. The point is that IPLD provides libraries that make the underlying data interoperable across tools and across protocols by default. Here are some properties of IPLD:

A canonical data model: A self-contained descriptive model that uniquely identifies any hash-based data structure and ensures the same logical object always maps to the exact same sequence of bits.

- Protocol independent resolution: IPLD brings isolated systems together(like connecting Bitcoin, Ethereum and git), making integration with existing protocols simple.
- Upgradeable: With Multiformats(we will dive more into this in part 4) support, IPLD is easily upgradeable and will grow with your favorite protocols.

# 03

## 3.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

- Operates across formats: Express your IPLD objects in various serializable formats like JSON, CBOR, YAML, XML and many more, making IPLD a cinch to use with any framework.
- Backward compatible: Non-intrusive resolvers make IPLD easy to integrate within your existing work.
- A namespace for all protocols: IPLD allows you to explore data across protocols seamlessly, binding hash-based data structures together through a common namespace.

IPLD is not a single specification, it is a set of specifications. The diagram below explains the nitty-gritty of IPLD:

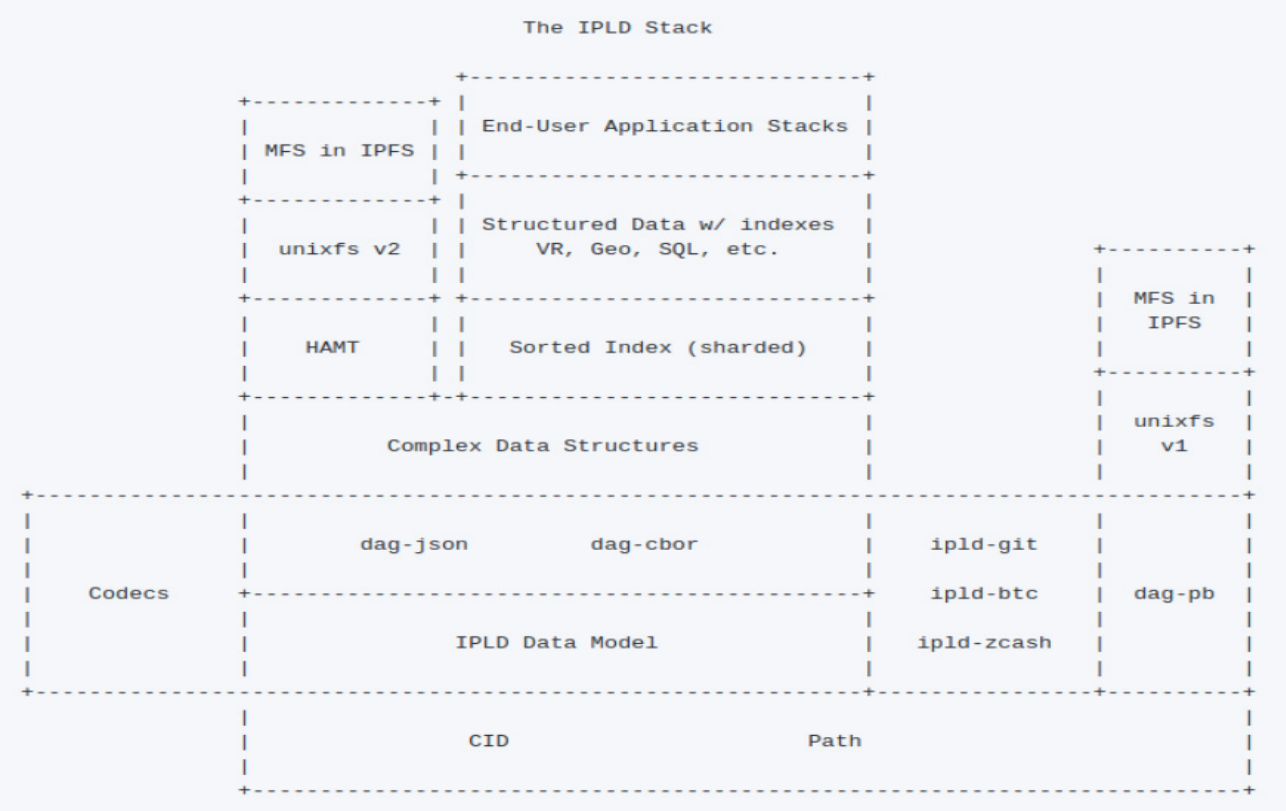



Figure: IPLD stack

### 3.2.2 Object Merkle DAG Basics

PiperNet builds a Merkle DAG, a directed acyclic graph where links between objects are cryptographic hashes of the targets embedded in the sources. This is a generalization of the Git data structure. Merkle DAGs provide PiperNet many useful properties, including:

**1. Content Addressing**: All content is uniquely identied by its multihash

# 03

## 3.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

checksum, including links. For android development android studio is needed to be installed.

**2. Tamper resistance**: all content is veried with its checksum. If data is tampered with or corrupted, PiperNet detects it.

**3. Deduplication**: all objects that hold the exact same content are equal, and only stored once. This is particularly useful with index objects, such as git trees and commits, or common portions of data.

The Object format is:

```
IPFSLink = {

      // name or alias of this link
      Name: name,

      // cryptographic hash of target
      Hash: Multihash,

      // total size of target
      Size: size
}

IPFSObject = {

      // array of links
      links: [IPFSLink],

      // opaque content data
      data: [dataBytes]
}
```

The Merkle DAG is an extremely flexible way to store data. The only requirements are that object references be (a) content addressed, and (b) encoded in the format above. Pipernet grants applications complete control over the data field; applications can use any custom data format they chose, which Pipernet may not understand. The separate in object link table allows Pipernet to:

-> List all object references in an object. For example:

```
ls /XLZ1625Jjn7SubMDgEyeaynFuR84ginqvzbXLYkgq61DYaQ8NhkcqyU7rLcnSa7d-
SHQ16x 189458 less
XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5 19441 script
XLF4hwVHsVuZ78FZK6fozf8Jj9WEURMbCX4 5286 template
<object multihash> <object size> <link name>
```

-> Resolve string path lookups, such as foo/bar/baz. Given an object, Pipernet

# 03

## 3.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

resolves the rst path component toa hash in the object's link table, fetches that second object, and repeats with the next component. Thus, string paths can walk the Merkle DAG no matter what the data formats are.

-> Resolve all objects referenced recursively.

A raw data eld and a common link structure are the necessary components for constructing arbitrary data structures on top of IPFS. While it is easy to see how the Git object model fits on top of this DAG, consider these other potential data structures: (a) key-value stores (b) traditional relational databases (c) Linked Data triple stores (d)linked document publishing systems (e) linked communications platforms (f) cryptocurrency blockchains. These can all be modeled on top of the Merkle DAG, which allows any of these systems to use PiperNet as a transport protocol for more complex applications.

### 3.2.3 Paths

In PiperNet, objects can be traversed with a string path API. Paths work as they do in traditional UNIX lesystems and the Web. The Merkle DAG links make traversing it easy. Note that full paths in PiperNet are of the form:

```
// format
/pnet/<hash-of-object>/<name-path-to-object>
// example
/pnet/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/foo.txt
```

The /net prefix allows mounting into existing systems at a standard mount point without conflict (mount point names are of course configurable). The second path component (first within PiperNet) is the hash of an object. This is always the case, as there is no global root. A root object would have the impossible task of handling consistency of millions of objects in a distributed (and possibly disconnected) environment. Instead, we simulate the root with content addressing. All objects are always accessible via their hash. Note this means that given three objects in path <foo>/bar/baz, the last object is accessible by all:

```
/pnet/<hash-of-foo>/bar/baz
/pnet/<hash-of-bar>/baz
/pnet/<hash-of-baz>
```

# 03

## 3.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

### 3.2.4 Local Objects

PiperNet clients require some local storage, an external system on which to store and retrieve local raw data for the objects PiperNet manages. The type of storage depends on the node's use case. In most cases, this is simply a portion of disk space (either managed by the native filesystem, by a key-value store such as leveldb, or directly by the PiperNet client). In others, for example non-persistent caches, this storage is just a portion of RAM. Ultimately, all blocks available in PipeNet are in some node's local storage. When users request objects, they are found, downloaded, and stored locally, at least temporarily. This provides fast lookup for some configurable amount of time thereafter.

### 3.2.5 Object Pinning

Nodes who wish to ensure the survival of particular objects can do so by pinning the objects. This ensures the objects are kept in the node's local storage. Pinning can be done recursively, to pin down all linked descendent objects as well. All objects pointed to are then stored locally. This is particularly useful to persist les, including references. This also makes PiperNet a Web where links are permanent, and Objects can ensure the survival of others they point to.

### 3.2.6 Uploading Object

PiperNet is globally distributed. It is designed to allow the files of millions of users to coexist together. The DHT, with content-hash addressing, allows publishing objects in a fair, secure, and entirely distributed way. Anyone can publish an object by simply adding its key to the DHT, adding themselves as a peer, and giving other users the object's path. Note that Objects are essentially immutable, just like in Git. New versions hash dierently, and thus are new objects.

### 3.2.7 Object-level Cryptography

PiperNet is equipped to handle object-level cryptographic operations. An encrypted or signed object is wrapped in a special frame that allows encryption or verification of the raw bytes.

# 03

## 3.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

```
EncryptedObject  = {
      // raw object data encrypted
      Object: objectinBytes,

      // optional tag for encryption groups
      Tag: tagInBytes
}

SignedObject = {
      // raw object data signed
      Object: objectInBytes,

      // hmac signature
      Signature: signatureInBytes,

      // multihash identifying key
      PublicKey: multihash


}
```

Cryptographic operations change the object's hash, denying a different object. PiperNet automatically veries signatures, and can decrypt data with user specified keychains. Links of encrypted objects are protected as well, making traversal impossible without a decryption key. It is possible to have a parent object encrypted under one key, and a child under another or not at all. This secures links to shared objects.

### 3.2.8 Files in DAG

PiperNet defines a set of objects for modelling a versioned file system on top of the Merkle DAG. This object model is similar to Git's:

1. block: a variable-size block of data.
2. list: a collection of blocks or other lists.
3. tree: a collection of blocks, lists, or other trees.

PiperNet uses a similar sturcture to Git but had to consider the following things in mind for a distributed file system:

(a) fast size lookups (aggregate byte sizes have been added to objects),

(b) large le deduplication (adding a list object), and

(c) embedding of commits into trees.

However, in PiperNet File objects are close enough to Git that conversion between the two is possible. Also, a set of Git objects can be introduced to convert

# 03

## 3.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

without losing any information (unix le permissions, etc). Git objects can be introduced to convert without losing any information (unix le permissions, etc).

-> File Object: blob

The blob object contains an addressable unit of data, and represents a le. IPFS Blocks are like Git blobs or file system data blocks. They store the users' data. Note that, in PiperNet files can be represented by both lists and blobs. Blobs have no links.

```
{
      // blobs have no links
      "data": "some data here",
}
```

-> File Object: list

The list object represents a large or deduplicated le made up of several object blobs concatenated together. lists contain an ordered sequence of blob or list objects. In a sense, the object list functions like a filesystem le with indirect blocks. Since lists can contain other lists, topologies including linked lists and balanced trees are possible. Directed graphs where the same node appears in multiple places allow in-le deduplication. Of course, cycles are not possible, as enforced by hash addressing.

```
{
      // lists have an array of object types as data
      "data": ["blob", "list", "blob"],

      // lists have no names in links
      "links": [
            { "hash": "XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x",
            "size": 189458 },
            { "hash": "XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5",
            "size": 19441 },
            { "hash": "XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z",
            "size": 5286 }
      ]
}
```

->File Object: tree

The tree object in PiperNet is similar to Git's: it represents a directory, a map of names to hashes. The hashes reference blobs, lists, other trees, or commits. Note that traditional path naming is already implemented by the Merkle DAG.

# 03

## 3.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

```
{
      // trees have an array of object types as data
      “data”: [“blob”, “list”, “blob”],
      “links”: [
            { “hash”: “XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x”,
            “name”: “less”, “size”: 189458 },
            { “hash”: “XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5”,
            “name”: “script”, “size”: 19441 },
            { “hash”: “XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z”,
            “name”: “template”, “size”: 5286 }
            // trees do have names
]
}
```

In PiperNet, objects can be traversed with a string path API. The File Objects are designed to make mounting PiperNet onto a UNIX lesys- tem simpler. They restrict trees to have no data, in order to represent them as directories.

### 3.2.9 Splitting Files into Lists and Blob

One of the main challenges with distributing large files is finding the right way to split them into independent blocks. Rather than assume it can make the right decision for every type of le, PiperNet offers the following alternatives:

(a) Use Rabin Fingerprints as in LBFS(Last Buffer First Served) to pick suitable block boundaries.

(b) Use the rsync rolling-checksum algorithm, to detect blocks that have changed between versions.

(c) Allow users to specify block-splitting functions highly tuned for specifics files.

### 3.2.9 Path Lookup Performance

Path-based access traverses the object graph. Retrieving each object requires looking up its key in the DHT, connecting to peers, and retrieving its blocks. This is considerable overhead, particularly when looking up paths with many compo- nents. This is mitigated by:

- Tree caching: since all objects are hash-addressed, they can be cached in- definitely. Additionally, trees tend to be small in size so PiperNet prioritizes caching them over blobs.
- Attened trees: for any given tree, a special flattened tree can be constructed to list all objects reachable
- from the tree. Names in the flattened tree would really be paths parting from

# 03

## 3.3

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

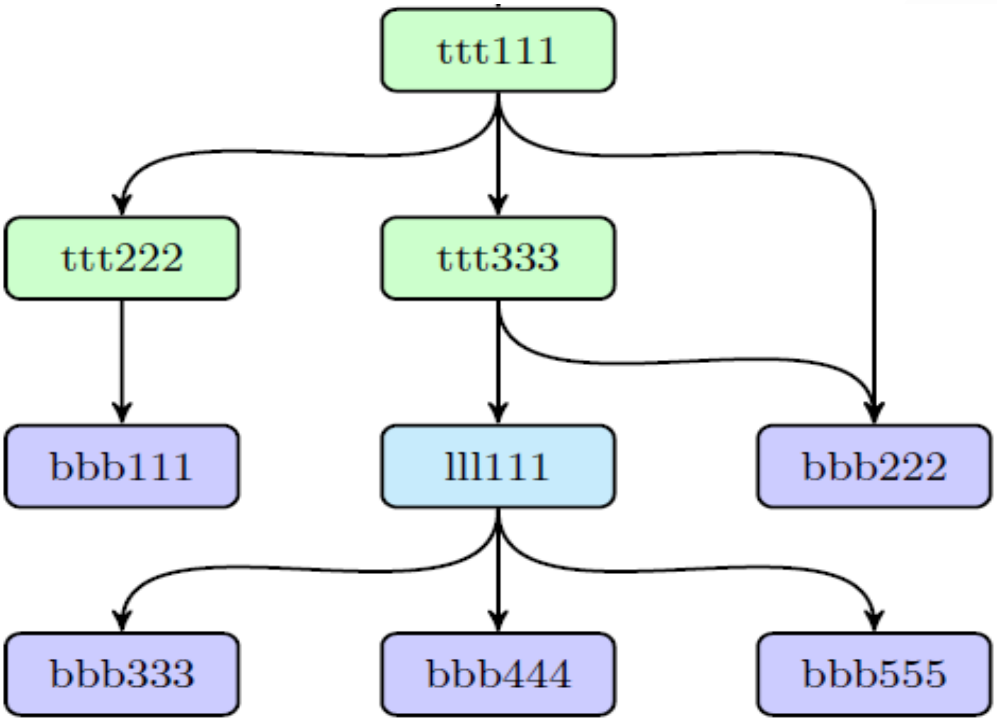



Figure: Sample Object

the original tree, with slashes. For example, flattened tree for ttt111 above:

```
{
      "data":  [
      "tree", "blob", "tree", "list", "blob" "blob"
      ],
      "links": [
            { "hash": "<ttt222-hash>", "size": 1234
            "name": "ttt222-name" },
            { "hash": "<bbb111-hash>", "size": 123,
            "name": "ttt222-name/bbb111-name" },
            { "hash": "<ttt333-hash>", "size": 3456,
            "name": "ttt333-name" },
            { "hash": "<lll111-hash>", "size": 587,
            "name": "ttt333-name/lll111-name"},
            { "hash": "<bbb222-hash>", "size": 22,
            "name": "ttt333-name/lll111-name/bbb222-name" },
            { "hash": "<bbb222-hash>", "size": 22
            "name": "bbb222-name" }
      ]
}
```

## 3.3 Exchange layer

The main Block Exchange protocol for PiperNet is the **BitSwap** Protocol. In Pipernet, data distribution happens by exchanging blocks with peers using a Bit-Torrent inspired protocol: BitSwap. Like BitTorrent, BitSwap peers are looking to acquire a set of blocks (want_list), and have another set of blocks to offer in exchange (have_list). Unlike BitTorrent, BitSwap is not limited to the blocks in

# 03

## 3.3

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

## Pipernet Layers

Exchange Layer

one torrent. BitSwap operates as a persistent marketplace where node can acquire the blocks they need, regardless of what files those blocks are part of. The blocks could come from completely unrelated files in the filesystem. Nodes come together to barter in the marketplace.

While the notion of a barter system implies a virtual currency could be created, this would require a global ledger to track ownership and transfer of the currency. This can be implemented as a BitSwap Strategy, and will be explored in a future paper. In the base case, BitSwap nodes have to provide direct value to each other in the form of blocks. This works fine when the distribution of blocks across nodes is complementary, meaning they have what the other wants. Often, this will not be the case. In some cases, nodes must work for their blocks. In the case that a node has nothing that its peers want (or nothing at all), it seeks the pieces its peers want, with lower priority than what the node wants itself. This incentivizes nodes to cache and disseminate rare pieces, even if they are not interested in them directly.

### 3.3.1 BitSwap Credit

The protocol must also incentivize nodes to seed when they do not need anything in particular, as they might have the blocks others want. Thus, BitSwap nodes send blocks to their peers optimistically, expecting the debt to be repaid. But leeches (free-loading nodes that never share) must be protected against. A simple credit-like system solves the problem:

1. Peers track their balance (in bytes verified) with other nodes.
2. Peers send blocks to debtor peers probabilistically, according to a function that falls as debt increases.

Note that if a node decides not to send to a peer, the node subsequently ignores the peer for an ignore_cooldown timeout. This prevents senders from trying to game the probability by just causing more dice-rolls. (Default BitSwap is 10 seconds).

### 3.3.2 BitSwap Strategy

The differing strategies that BitSwap peers might employ have wildly different

# 03

## 3.3

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

effects on the performance of the exchange as a whole. In BitTorrent, while a standard strategy is specified (tit-for-tat), a variety of others have bee imple-mented, ranging from BitTyrant [8] (sharing the least-possible), to BitThief [8] (exploiting a vulnerability and never share), to PropShare [8] (sharing propor-tionally). A range of strategies (good and malicious) could similarly be imple-mented by BitSwap peers. The choice of function, then, should aim to:

1. maximize the trade performance for the node, and the whole exchange
2. prevent freeloaders from exploiting and degrading the exchange
3. be eective with and resistant to other, unknown strategies
4. be lenient to trusted peers

The exploration of the space of such strategies is future work. One choice of function that works in practice is a sigmoid, scaled by a debt retio:

Let the debt ratio r between a node and its peer be:

$$r = \frac{\texttt{bytes\_sent}}{\texttt{bytes\_recv} + 1}$$

Given r, let the probability of sending to a debtor be:

$$P\Big(\ send \mid r\ \Big) = 1 - \frac{1}{1 + exp(6 - 3r)}$$

As you can see in Figure 1, this function drops off quickly as the nodes' debt ratio surpasses twice the established credit.

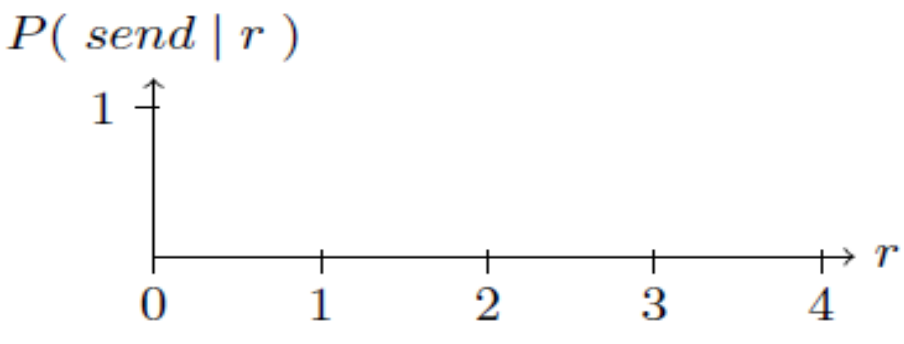



Figure: Probability of Sending as r increases

# 03

## 3.3

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

The debt ratio is a measure of trust: lenient to debts between nodes that have previously exchanged lots of data successfully, and merciless to unknown, untrusted nodes. This (a) provides resistance to attackers who would create lots of new nodes (sybill attacks), (b) protects previously successful trade relationships, even if one of the nodes is temporarily unable to provide value, and (c) eventually chokes relationships that have deteriorated until they improve.

### 3.3.3 BitSwap Ledger

BitSwap nodes keep ledgers accounting the transfers with other nodes. This allows nodes to keep track of history and avoid tampering. When activating a connection, BitSwap nodes exchange their ledger information. If it does not match exactly, the ledger is reinitialized from scratch, losing the accrued credit or debt. It is possible for malicious nodes to purposefully \lose" the Ledger, hoping to erase debts. It is unlikely that nodes will have accrued enough debt to warrant also losing the accrued trust; however the partner node is free to count it as misconduct, and refuse to trade.

```
Ledger = {
      owner: ownerId,
      partner: partnerId,
      bytes_sent: bytes_sent,
      bytes_recv: bytes_recv
      timestamp: timestamp
}
```

Nodes are free to keep the ledger history, though it is not necessary for correct operation. Only the current ledger entries are useful. Nodes are also free to garbage collect ledgers as necessary, starting with the less useful ledgers: the old (peers may not exist anymore) and small.

### 3.3.4 BitSwap Ledger

BitSwap nodes follow a simple protocol.

```
// Additional state kept
BitSwap = {
      ledgers: ledgerMap, // Ledgers known to this node, inc inactive
      active: peerMap,    // currently open connections to other nodes
      need_list: [],     // checksums of blocks this node needs
      have_list: []     // checksums of blocks this node has
}
```

# 03

## 3.3

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

```
Peer = {
      nodeid: NodeId,

      // Ledger between the node and this peer
      ledger: Ledger,
      // timestamp of last received message
      last_seen: timestampOfLast_seen,
      // checksums of all blocks wanted by peer
      // includes blocks wanted by peer's peers
      want_list: [],
      open: function(nodeid, ledger){...},
      send_want_list: function(want_list){...},
      send_block: function(block){...},
      close (final){...}
}
```

Sketch of the lifetime of a peer connection:

1. Open: peers send ledgers until they agree.

2. Sending: peers exchange want_lists and blocks.

3. Close: peers deactivate a connection.

4. Ignored: (special) a peer is ignored (for the duration of a timeout) if a node's strategy avoids sending.

Peer.open(NodeId, Ledger): When connecting, a node initializes a connection with a Ledger, either stored from a connection in the past or a new one zeroed out. Then, sends an Open message with the Ledger to the peer. Upon receiving an Open message, a peer chooses whether to activate the connection. If - according to the receiver's Ledger - the sender is not a trusted agent (transmission below zero, or large outstanding debt) the receiver may opt to ignore the request. This should be done probabilistically with an ignore_cooldown timeout, as to allow errors to be corrected and attackers to be thwarted. If activating the connection, the receiver initializes a Peer object with the local version of the Ledger and sets the last_seen timestamp. Then, it compares the received Ledger with its own. If they match exactly, the connections have opened. If they do not match, the peer creates a new zeroed out Ledger and sends it.

Peer.send_want_list(WantList): While the connection is open, nodes advertise their want_list to all connected peers. This is done (a) upon opening the connection, (b) after a randomized periodic timeout, (c) after a change in the want_list and (d) after receiving a new block. Upon receiving a want_list, a node stores

# 03

## 3.3

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

it. Then, it checks whether it has any of the wanted blocks. If so, it sends them according to the BitSwap Strategy above.

Peer.send_block(Block): Sending a block is straightforward. The node simply transmits the block of data. Upon receiving all the data, the receiver computes the Multihash checksum to verify it matches the expected one, and returns confirmation. Upon finalizing the correct transmission of a block, the receiver moves the block from need_list to have_list, and both the receiver and sender update their ledgers to reflect the additional bytes transmitted. If a transmission verification fails, the sender is either malfunctioning or attacking the receiver. The receiver is free to refuse further trades. Note that BitSwap expects to operate on a reliable transmission channel, so transmission errors - which could lead to incorrect penalization of an honest sender - are expected to be caught before the data is given to BitSwap.

Peer.close(Bool):The final parameter to close signals whether the intention to tear down the connection is the sender's or not. If false, the receiver may opt to re-open the connection immediately. This avoids premature closes. A peer connection should be closed under two conditions:

- a silence_wait timeout has expired without receiving any messages from the peer (default BitSwap uses 30 seconds).
- The node issues Peer.close(false). the node is exiting and BitSwap is being shut down. In this case, the node issues Peer.close(true).

After a close message, both receiver and sender tear down the connection, clearing any state stored. The Ledger may be stored for the future, if it is useful to do so.

Non-open messages on an inactive connection should be ignored. In case of a send_block message, the receiver may check the block to see if it is needed and correct, and if so, use it. Regardless, all such out-of-order messages trigger a close(false) message from the receiver to force re-initialization of the connection.

# 03

## 3.4

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

## 3.4 Routing layer

PiperNet nodes require a routing system that can find

(a) other peers' network addresses and

(b) peers who can serve particular objects.

PiperNet achieves this using a DSHT based on S/Kademlia and Coral algorithm. At its core, the distributed hashed table is constructed and looked up using the Chord protocol. The size of objects and use patterns of PiperNet are similar to Coral and Mainline, so the DHT makes a distinction for values stored based on their size. Small values (equal to or less than 1KB) are stored directly on the DHT. For values larger, the DHT stores references, which are the NodeIds of peers who can serve the block. The structure of this DSHT is the following:

```
IPFSRouting = {
      // gets a particular peer's network address
      FindPeer: function(nodeId){...},

      // stores a small metadata value in DHT
      SetValue: function(keyInbytes, valueInbytes){...},

      // retrieves small metadata value from DHT
      GetValue: function(keyInbytes){...},

      // announces this node can serve a large value
      ProvideValue: function(keyMultihash){...},

      // gets a number of peers serving a large value
      FindValuePeers: function(keyMultihash, minValue){...}
}
```

### 3.4.1 Chord Protocol

Chord is a protocol and algorithm for a peer-to-peer distributed hash table. A distributed hash table stores key-value pairs by assigning keys to different computers (known as "nodes"); a node will store the values for all the keys for which it is responsible. Chord specifies how keys are assigned to nodes, and how a node can discover the value for a given key by first locating the node responsible for that key. Nodes and keys are assigned an m-bit identifier using consistent hashing. The SHA-1 algorithm is the base hashing function for consistent hashing. Consistent hashing is integral to the robustness and performance of Chord because both keys and nodes (in fact, their IP addresses) are uniformly distributed in the same identifier space with a negligible possibility of collision. Thus, it

# 03

## 3.4

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

also allows nodes to join and leave the network without disruption. In the protocol, the term node is used to refer to both a node itself and its identifier (ID) without ambiguity. So is the term key.

Using the Chord lookup protocol, nodes and keys are arranged in an identifier circle that has at most $2^m$ nodes, ranging from 0 to $2^m$ - 1. m should be large enough to avoid collision.) Some of these nodes will map to machines or keys while others (most) will be empty.

Each node has a successor and a predecessor. The successor to a node is the next node in the identifier circle in a clockwise direction. The predecessor is counter-clockwise. If there is a node for each possible ID, the successor of node 0 is node 1, and the predecessor of node 0 is node $2^m$ - 1; however, normally there are "holes" in the sequence. For example, the successor of node 153 may be node 167 (and nodes from 154 to 166 do not exist); in this case, the predecessor of node 167 will be node 153.

The concept of successor can be used for keys as well. The successor node of a key k is the first node whose ID equals to k or follows k in the identifier circle, denoted by successor(k). Every key is assigned to (stored at) its successor node, so looking up a key k is to query successor(k).

Since the successor (or predecessor) of a node may disappear from the network (because of failure or departure), each node records a whole segment of the circle adjacent to it, i.e., the r nodes preceding it and the r nodes following it. This list results in a high probability that a node is able to correctly locate its successor or predecessor, even if the network in question suffers from a high failure rate.

The core usage of the Chord protocol is to query a key from a client (generally a node as well), i.e. to find successor(k). The basic approach is to pass the query to a node's successor, if it cannot find the key locally. This will lead to a O(N) query time where N is the number of machines in the ring. To avoid the linear search above, Chord implements a faster search method by requiring each node to keep a finger table containing up to m entries, recall that m is the number of bits in the hash key. The $i^{th}$ entry of node n will contain successor:

# 03

## 3.4

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

$((n+2^{i-1}) \bmod 2^m)$. The first entry of finger table is actually the node's immediate successor (and therefore an extra successor field is not needed). Every time a node wants to look up a key k, it will pass the query to the closest successor or predecessor (depending on the finger table) of k in its finger table (the "largest" one on the circle whose ID is smaller than k), until a node finds out the key is stored in its immediate successor.

With such a finger table, the number of nodes that must be contacted to find a successor in an N-node network is O(log N).

Whenever a new node joins, three invariants should be maintained (the first two ensure correctness and the last one keeps querying fast):

1. Each node's successor points to its immediate successor correctly.
2. Each key is stored in {\displaystyle successor(k)}successor(k).
3. Each node's finger table should be correct.

To satisfy these invariants, a predecessor field is maintained for each node. As the successor is the first entry of the finger table, we do not need to maintain this field separately any more. The following tasks should be done for a newly joined node n:

1. Initialize node {\displaystyle n}n (the predecessor and the finger table).
2. Notify other nodes to update their predecessors and finger tables.
3. The new node takes over its responsible keys from its successor.

The predecessor of n can be easily obtained from the predecessor of successor(n) (in the previous circle). As for its finger table, there are various initialization methods. The simplest one is to execute find successor queries for all m entries, resulting inO(M log N) initialization time. A better method is to check whether $i^{th}$ entry in the finger table is still correct for the $(i+1)^{th}$ entry. This will lead to $O(\log_2 N)$. The best method is to initialize the finger table from its immediate neighbours and make some updates, which is O(log N).

### 3.4.1 Coral DSHT

While some peer-to-peer lesystems store data blocks directly in DHTs, this wastes storage and bandwidth, as data must be stored at nodes where it is not needed" . The Coral DSHT extends Kademlia in three particularly important ways:

# 03

## 3.4

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

1. Kademlia stores values in nodes whose ids are "nearest" (using XOR-distance) to the key. This does not take into account application data locality, ignores \far" nodes that may already have the data, and forces\nearest" nodes to store it, whether they need it or not. This wastes significant storage and bandwith. Instead, Coral stores addresses to peers who can provide the data blocks.

2. Coral relaxes the DHT API from get_value(key) to get_any_values(key) (the "sloppy" in DSHT). This still works since Coral users only need a single (working) peer, not the complete list. In return, Coral can distribute only subsets of the values to the "nearest" nodes, avoiding hot-spots (overloading all the nearest nodes when a key becomes popular).

3. Additionally, Coral organizes a hierarchy of separate DSHTs called clusters depending on region and size. This enables nodes to query peers in their region first,"finding nearby data without querying distant nodes" and greatly reducing the latency of lookups.

### 3.4.1 S/Kademlia DSHT

Kademlia is a popular DHT that provides:

1. Efficient lookup through massive networks: queries on average contact dlog2(n)e nodes. (e.g. 20 hops for a network of 10; 000; 000 nodes).

2. Low coordination overhead: it optimizes the number of control messages it sends to other nodes.

3. Resistance to various attacks by preferring long-lived nodes.

4. Wide usage in peer-to-peer applications, including Gnutella and BitTorrent, forming networks of over 20 million nodes.

S/Kademlia extends Kademlia to protect against malicious attacks in two particularly important ways:

1. S/Kademlia provides schemes to secure NodeId generation, and prevent Sybill attacks. It requires nodes to create a PKI key pair, derive their identity from it, and sign their messages to each other. One scheme includes a proof-of-work crypto puzzle to make generating Sybills expensive.

2. S/Kademlia nodes lookup values over disjoint paths, in order to ensure honest nodes can connect to each other in the presence of a large fraction of adversar

# 03

## 3.5

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

ies in the network. S/Kademlia achieves a success rate of 0.85 even with an adversarial fraction as large as half of the nodes.

## 3.5 Network layer

In PiperNet nodes communicate regualarly with hundreds of other nodes in the network, potentially across the wide internet.

The PiperNet network stack features:

**Transport**: Can use any transport protocol, and is best suited for WebRTC DataChannels (for browser connectivity) or UTP(LEDBAT).

**Reliability**: Can provide reliability if underlying networks do not provide it, using UTP (LEDBAT) or SCTP.

**Connectivity**: Uses the ICE NAT traversal techniques.

**Integrity**: Optionally checks integrity of messages using a hash checksum.

**Authenticity**: Optionally checks authenticity of messages using HMAC with sender's public key.

### 3.5.1 Peer Addressing

Pipernet can use any network; it does not rely on or assume access to IP. This allows PiperNet to be used in overlay networks. PiperNet stores addresses as multiaddr formatted byte strings for the underlying network to use. multiaddr provides a way to express addresses and their protocols, including support for encapsulation. For example:

```
# an SCTP/IPv4 connection
/ip4/10.20.30.40/sctp/1234/
# an SCTP/IPv4 connection proxied over TCP/IPv4
/ip4/5.6.7.8/tcp/5678/ip4/1.2.3.4/sctp/1234/
```

### 3.5.2 Libp2p (Also include router and exchange layer)

The Significance of Libp2p: The early research on ARPANET(predecessor of the internet) began around the question that "How do you build a network which is so resilient, so that it could even survive a NUCLEAR WAR?". These were banks of humans operators connecting wires to facilitate country-wide communication. This was a highly centralized system, which could be destroyed easily in a nuclear war.

# 03

## 3.5

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

Fast-forward to today's world, most of the internet companies today have centralized servers, most of which are hosted in the data centers outside the cities. The internet we see today is still somehow full of services similar to the bank of human operators which are highly centralized and fragile(well, we aren't in a constant threat of a nuclear war today, but any technical or natural disaster can cause great damage).

Today, we are dependent on the internet more than ever. We are dependent on it for life and death situations. But the current internet, as we see it today is quite fragile and has several design problems. Most of these problems stem from the location-addressing. This means that whenever any device(your phone, laptop, fridge, car) connects to the internet(via wifi, broadband, satellite…whatever) it gets a name(and hence the data stored in it) based on its location. The proposed solution to this is content addressing, which allows a much resilient, peer-to-peer model. However, in order to give content addressing properties to the web, we have to go deeper into networking and overcome many of the challenges that the current internet infrastructure has. These challenges include things like:

- NATs: NATs blind devices from each other on the network. Ever tried to host a website from your college network? I bet the experience would have sucked.
- Firewalls: Apply to restrictive filters on the network, which restricts different devices to talk to each other. Every got facebook blocked on your office networks? Yeah, that's firewalls right there.
- High-latency networks: This is why sometimes the internet is slow, and you are mad because your favorite movie on Netflix is buffering.
- Reliability: Networks which are just not reliable and keep failing. We all have a memory of one shitty wifi that would just lose connection every time.
- Roaming: Problems with changing IP addresses while roaming due to location-based addressing. We all use free wifi of the coffee shops that we visit. While using the wifi your device is assigned an IP address(the identity of your device). When you move out of the coffee shop, your IP changes(as you disconnect from the wifi). So, your device never has a single identity that stays the same over a period of time.

# 03

## 3.5

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

- Censorship: The fact that Google is banned in China and children(well, everybody) of Turkey have never used Wikipedia for their school assignments.
- Runtimes with different properties: Lots of devices with very different connectivity properties force developers to work very hard to get around their limitations to get connected with each other, and a lot of times they are just simply not able to connect at all.
- Ever met a network engineer(IT guy) having a bad day? Well, pretty much every day.
- Innovation is super slow: Innovation in networks land is really slowwww…It takes years or even a decade for a new protocol to go down the OSI stack and to be deployed on a large scale. Even with a huge amount of funding, it might take years to decades to deploy new fantastic ideas to the internet's fabric.

These problems are partly because of the location-addressing model. The proposed solution to these problems is content-addressing.

Libp2p is not the first project that is working on the peer-to-peer content-addressing model. There have been a ton of projects like eMule, Kazaa, LimeWire, Napster, BitTorrent, some of which even exist until today. There was a lot of research done on algorithms like Kademlia, Pastry which powered these projects. Even though all these projects existed and got popular, it is really hard to leverage the work done in the past. Most of these projects have the classic software usability problems. These problems include:

- Lack of good documentation, or none at all.
- Restrictive Licensing or no licensing to be found.
- Old with last update more than a decade away.
- No easy to reach, point of contact.
- Closed Source(Product) or the source doesn't exist anymore.
- No specification provided.
- The implementation doesn't expose a friendly API.
- Projects that are tightly coupled with a specific use-case.
- Make a lot of assumptions about how they work.

A good example of this is WebRTC, the peer-to-peer transport of the web

# 03

## 3.5

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

platform. Most of its opensource implementations are build by hacking the Chrome browser and are buggy and unreliable.

So, there is a need for a future-proof framework, that can efficiently encapsulate all the standards, which would allow:

- faster innovation, as the things are modular as-well-as compatible.
- future-proof system; so that things do not break as we add new things in the coming future.
- cross compatible; not only this allows new things to be added but also keeps the support for previously used standards, so that we don't kill the legacy systems.

These are the goals for libp2p software component. The one-liner pitch is that libp2p is a modular system of protocols, specifications, and libraries that enable the development of peer-to-peer network applications.

A peer-to-peer network is one in which the participants (referred to as peers or nodes) communicate with one another directly, on more or less "equal footing". This does not necessarily mean that all peers are identical; some may have different roles in the overall network. However, one of the defining characteristics of a peer-to-peer network is that they do not require a privileged set of "servers" which behave completely differently from their "clients", as is the case in the predominant client/server model.

Because the definition of peer-to-peer networking is quite broad, many different kinds of systems have been built that all fall under the umbrella of "peer-to-peer". The most culturally prominent examples are likely the file-sharing networks like BitTorrent, and, more recently, the proliferation of blockchain networks that communicate in a peer-to-peer fashion.

While peer-to-peer networks have many advantages over the client/server model, there are also challenges that are unique and require careful thought and practice to overcome. In the process of overcoming these challenges, while building IPFS, we(contributors of IPFS) took care to build the solutions in a modular, composable way, into what is now libp2p. Although libp2p grew out of IPFS, it does not require or depend on IPFS, and today many projects(including a way

# 03

## 3.5

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

Libra aka ZuckBucks) use libp2p as their network transport layer. Together we can leverage our collective experience and solve these foundational problems in that benefits an entire ecosystem of developers and a world of users. Here is the layers of the Libp2p library:

->Transport: At the foundation of libp2p is the transport layer, which is responsible for the actual transmission and receipt of data from one peer to another. There are many ways to send data across networks in use today, with more in development and still more yet to be designed. libp2p provides a simple interface that can be adapted to support existing and future protocols, allowing libp2p applications to operate in many different runtime and networking environments.

->Identity: In a world with billions of networked devices, knowing who you're talking to is key to secure and reliable communication. libp2p uses public key cryptography as the basis of peer identity, which serves two complementary purposes. First, it gives each peer a globally unique "name", in the form of a PeerId. Second, the PeerId allows anyone to retrieve the public key for the identified peer, which enables secure communication between peers.

Security: It's essential that we be able to send and receive data between peers securely, meaning that we can trust the identity of the peer we're communicating with and that no third-party can read our conversation or alter it in-flight.

libp2p supports "upgrading" a connection provided by a transport into a securely encrypted channel. The process is flexible and can support multiple methods of encrypting communication. The current default is secio, with support for TLS 1.3 under development.

->Peer Routing: When you want to send a message to another peer, you need two key pieces of information: their PeerId, and a way to locate them on the network to open a connection.

There are many cases where we only have the PeerId for the peer we want to contact, and we need a way to discover their network address. Peer routing is the process of discovering peer addresses by leveraging the knowledge of other peers.

In a peer routing system, a peer can either give us the address we need if they

# 03

## 3.5

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

have it or else send our inquiry to another peer who's more likely to have the answer. As we contact more and more peers, we not only increase our chances of finding the peer we're looking for, we build a more complete view of the network in our own routing tables, which enables us to answer routing queries from others.

The current stable implementation of peer routing in libp2p uses a distributed hash table to iteratively route requests closer to the desired PeerId using the Kademlia routing algorithm.

->Content Discovery: In some systems, we care less about who we're speaking with than we do about what they can offer us. For example, we may want some specific piece of data, but we don't care who we get it from since we're able to verify its integrity. libp2p provides a content routing interface for this purpose, with the primary stable implementation using the same Kademlia-based DHT as used in peer routing.

->Messaging / PubSub: Sending messages to other peers is at the heart of most peer-to-peer systems, and pubsub (short for publish/subscribe) is a very useful pattern for sending a message to groups of interested receivers. libp2p defines a pubsub interface for sending messages to all peers subscribed to a given "topic". The interface currently has two stable implementations; floodsub uses a very simple but inefficient "network flooding" strategy, and gossipsub defines an extensible gossip protocol. There is also active development in progress on episub, an extended gossipsub that is optimized for single source multicast and scenarios with a few fixed sources broadcasting to a large number of clients in a topic.

Libp2p is a collection of building blocks that expose a very well-defined, documented and tested interfaces so they are composable, swappable and hence upgradable.

In other words, Libp2p takes the current "vertical" OSI model and makes it horizontal.

# 03

## 3.5

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

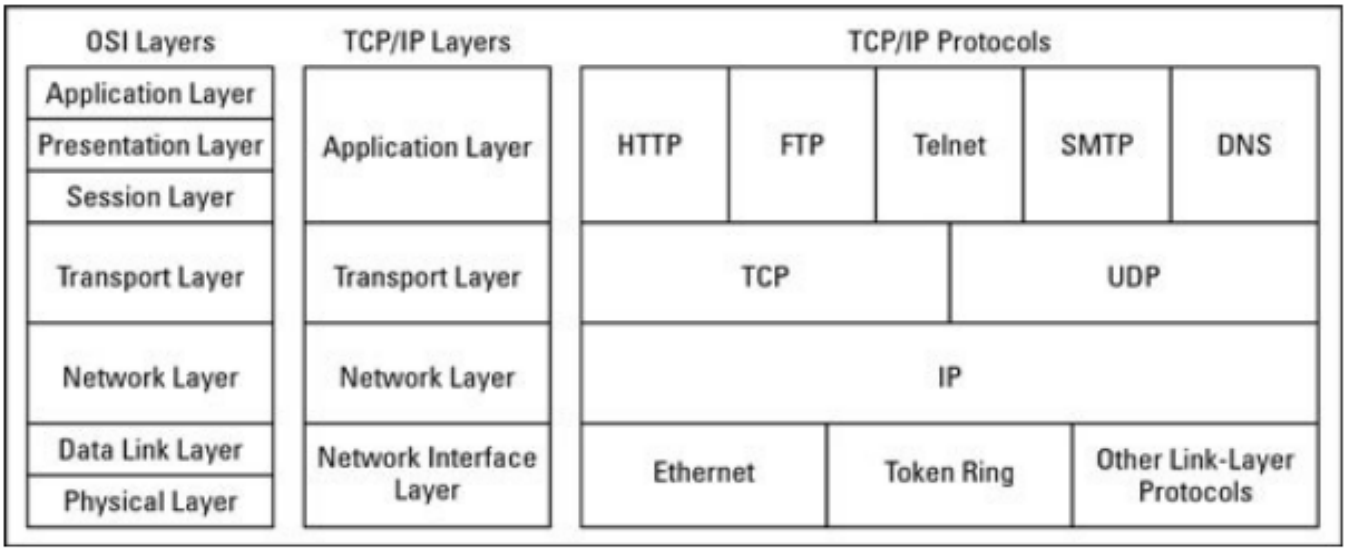



Figure: Current "Vertical" OSI model

This current networking model has a few shortcomings:

1. Repeated actions throughout multiple layers, hence a lot of duplicated work and wasting resources.
2. Information is hidden between the layers, missing significant improvement opportunities.

This is mainly due to the fact that internet protocols are spread across multiple layers(as shown in the diagram) and are very tightly coupled which makes them hard to change, thus discouraging innovation.

Libp2p changes this and just gives you a frame in which all of these protocols can co-exist and co-operate.


Auth
App
UI

Chapter 4

# Higher Layer

4.1 **Authentication**
Detailed description of the authentication steps

4.2 **Application layer**
Step by step description of appliction layer

4.3 **User Interface mockup**
Mock up of all the view using Adobe XD

# 04

## 4.1

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

The previous chapter discussed the internal layers inspired by IPFS essential to operate a full-fledged distributed application. In this chapter, the application layer itself would be discussed in the context of authentication, steps executed in the application layer and the mock up user interface.

## 4.1 Authentication

PiperNet uses the concepts in Primechain-API which combines the power of blockchain technology with public key cryptography to enable:

1. Secure authentication and identification of smart-phones, other devices & users
2. Securing & encrypting online communications
3. Password-less login systems
4. Preventing fake emails & phishing
5. Authenticating DNS records & preventing spoofing attacks
6. Electronic signatures

Blockchain-based authentication has some special features:

1. Signing and decryption keys stay on the device.
2. Verification and encryption keys are stored on the blockchain.
3. Protects against critical cyber attacks such as phishing, man-in-the-middle, replay attacks.

The steps:

Step 1 — Retrieving the RSA public key of the verifier

Step 2 — Encrypting the blockchain address of the requester

Step 3 — Sending the encrypted blockchain address to the verifier

Step 4 — Decrypting the encrypted blockchain address

Step 5 — Retrieving the RSA public key of the requester

Step 6 — Generating a random string and timestamp and hash

Step 7 — Sending the hash to the requester

Step 8 — Decryption of the hash

Step 9 — Signing of the hash by the requester

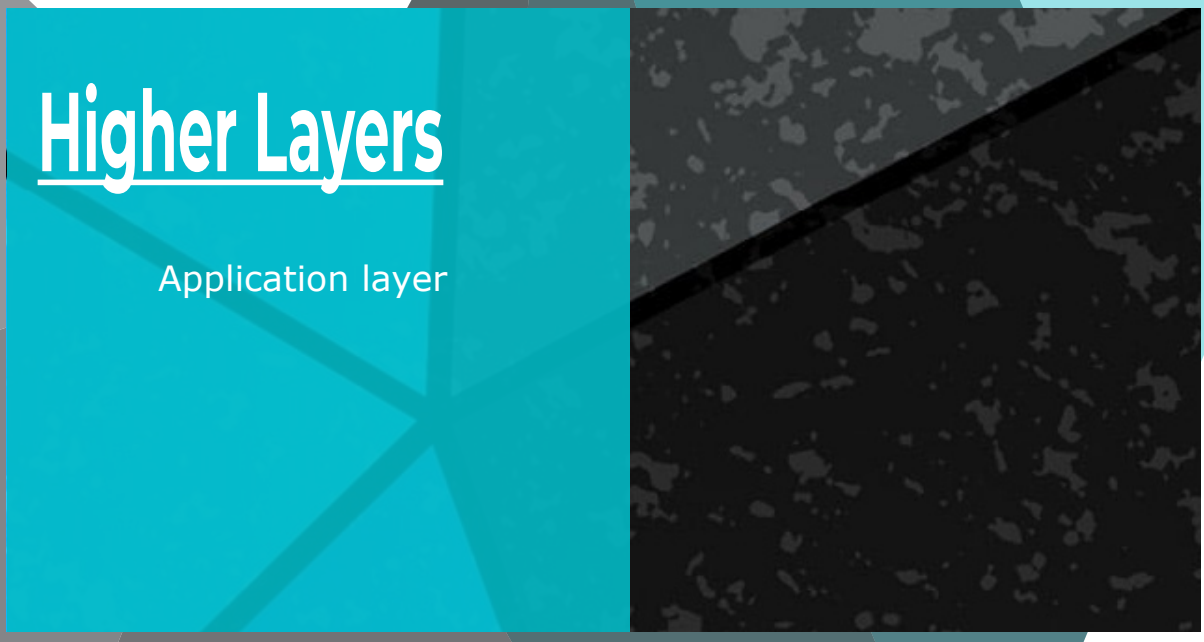
Step 10 — Creating the encrypted envelope

# 04

## 4.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**



Step 11 — Sending the encrypted envelope to the verifier

Step 12 — Decrypting of the encrypted envelope by the verifier

Step 13 — Verification of the digital signature

## 4.2 Application layer

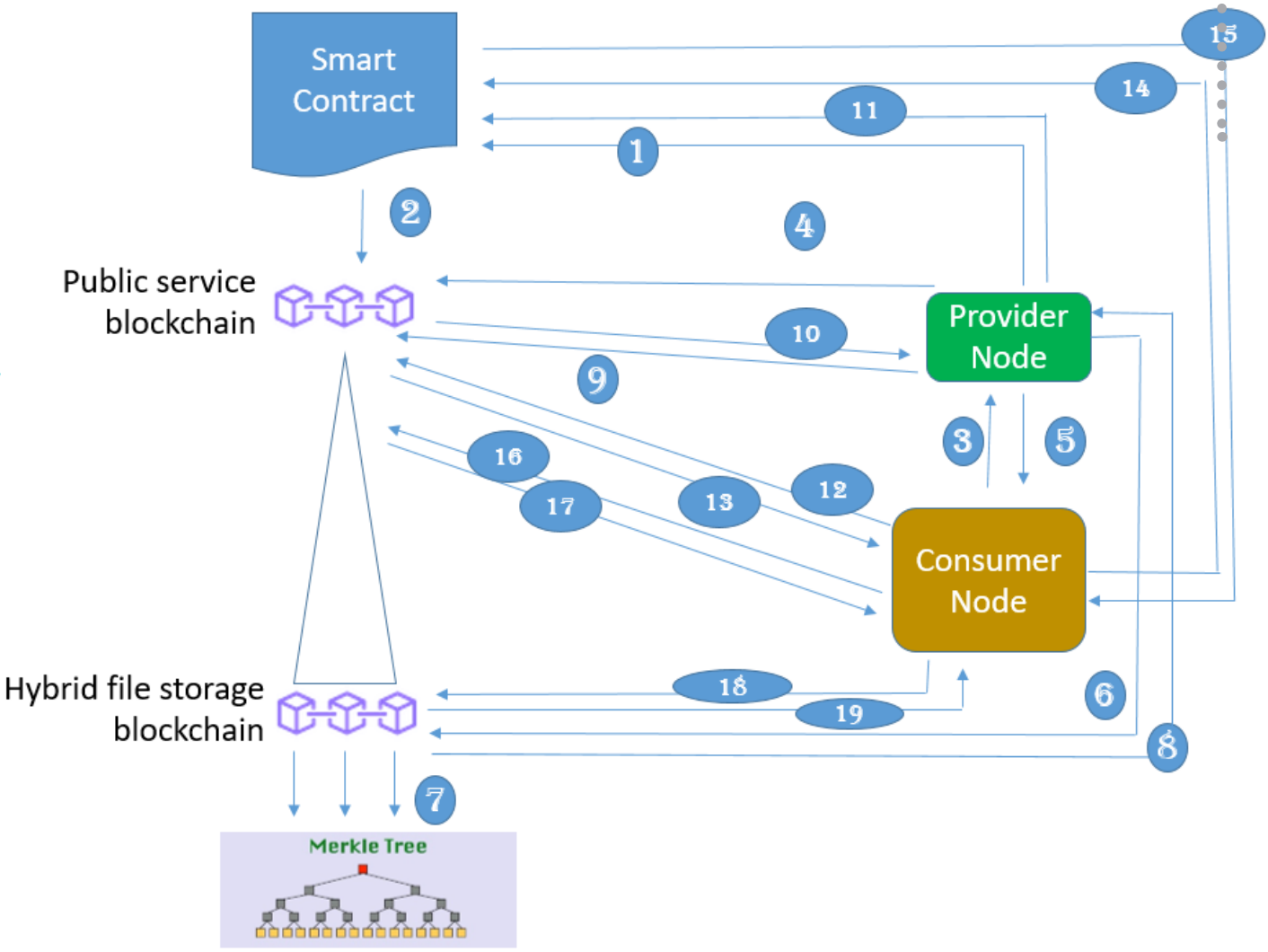



Figure: PiperNet Application Layer

The PiperNet system consists of the following two entities:

Provider Node(PN): PN is a node/machine/entity/person that owns a series of les to share.

Consumer Node(CN): CN are PN's data clients that are authorized to view some of the file.

It should be noted that this distinction is logical, the same physical device can both be a provider node and consumer node.

The corresponding description of each step number in the Figure is shown as follows

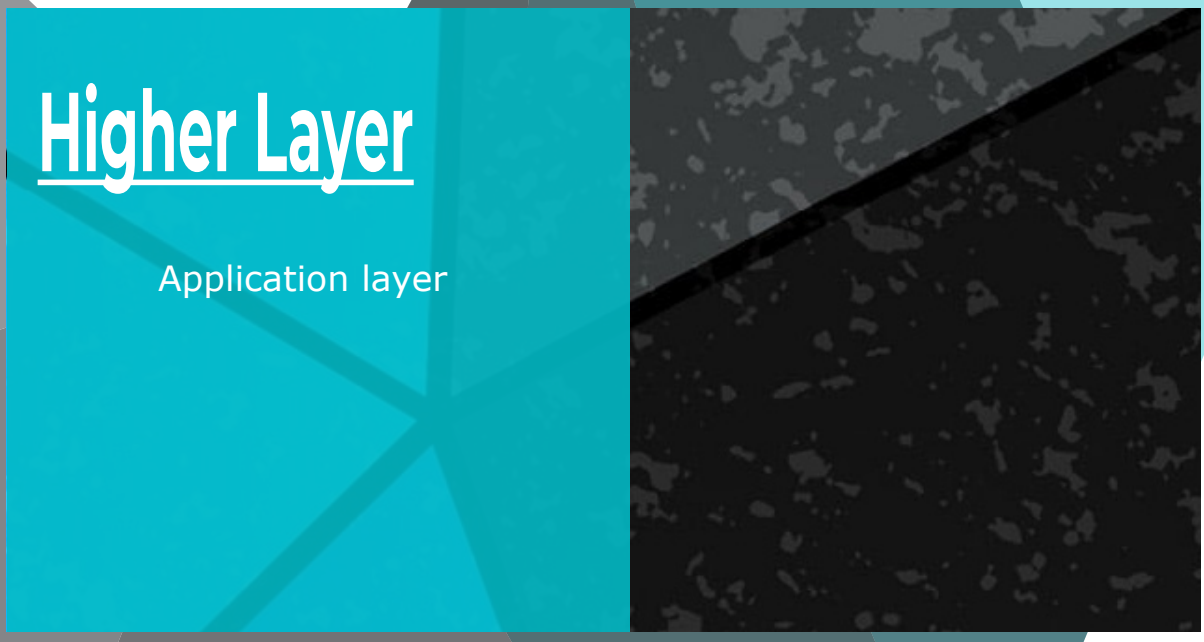
1 PN setups the system. The system master key is encrypted and then it is embedded into the public

# 04

## 4.2

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**



PiperNet blockchain.

2 PN deploys the predefined for PiperNet smart contract on the PiperNet blockchain.

3 CN sends a registration request to PM(this can occur either when searching for file or requesting from a specific user).

4 PN generates secret key for CN, and uses the shared key to encrypt the secret key and embed the encrypted secret key into an Ethereum transaction.

5 PN sends the transaction id related to secret key to CN through a secure channel.

6 PN selects a keyword set from the shared file, and uses AES algorithm to encrypt the file and uploads it to Hybrid file storage blockchain.

7 PN records the file location returned by file storage blockchain.

8 PN uses a selected AES key K to encrypt the file. location, and using eliptic cryptographic algorithm to encrypt AES key K. Then selects a AES key K1 to encrypt these information and embeds it into a PiperNet block transaction.

9 PN records the id of the blockchain transaction and AES key K1.

10 PN generates encrypted keyword indexes and stores it to the smart contract.

11 CN reads transaction data related to secret key from the public blockchain.

12 CN uses the shared key to decrypt transaction data, then gets secret key.

13 CN generates search token and invokes the smart contract.

14 The smart contract searches according to the token and returns the relevant results.

15 CN reads relevant transaction data based on search results returned by smart contracts.

16 CN decrypts the transaction data.

17 CN downloads encrypted file from PiperNet file storage blockchain.

18 CN decrypts the encrypted file.

# 04

## 4.3

Project name:

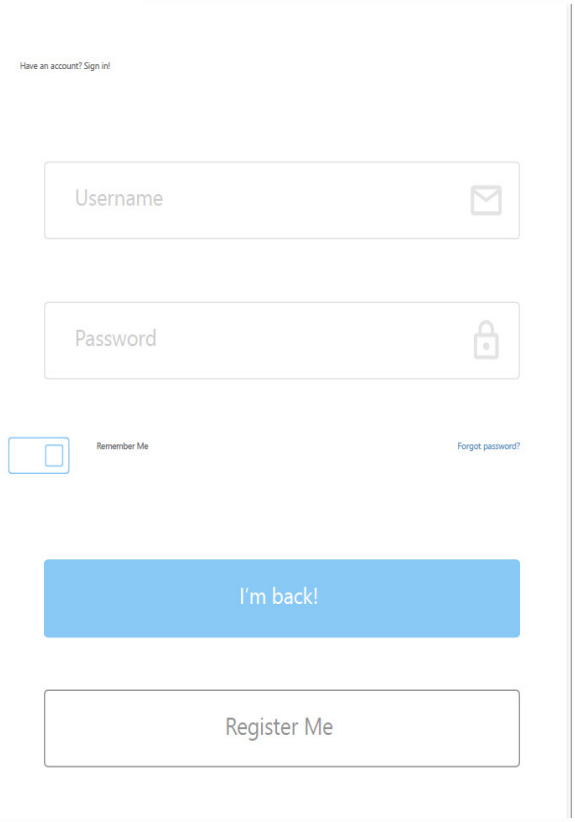
**A blockchain-based web hosting scheme for content-addressed distributed file system**

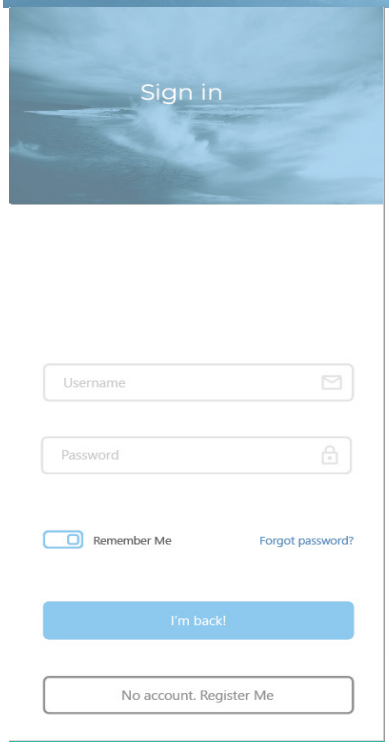
## Higher Layer

User Interface Mockup

## 4.3 User Interface Mockup

This section will consist all the mock up of the views of the PiperNet application using adobe XD. Adobe XD is a vector-based user experience design tool for web apps and mobile apps, developed and published by Adobe Inc. The views that designed as mockups w home view, authentication(sign up and login), your world, piperNet world.


**Mockup**

**Sign in view**

# 04

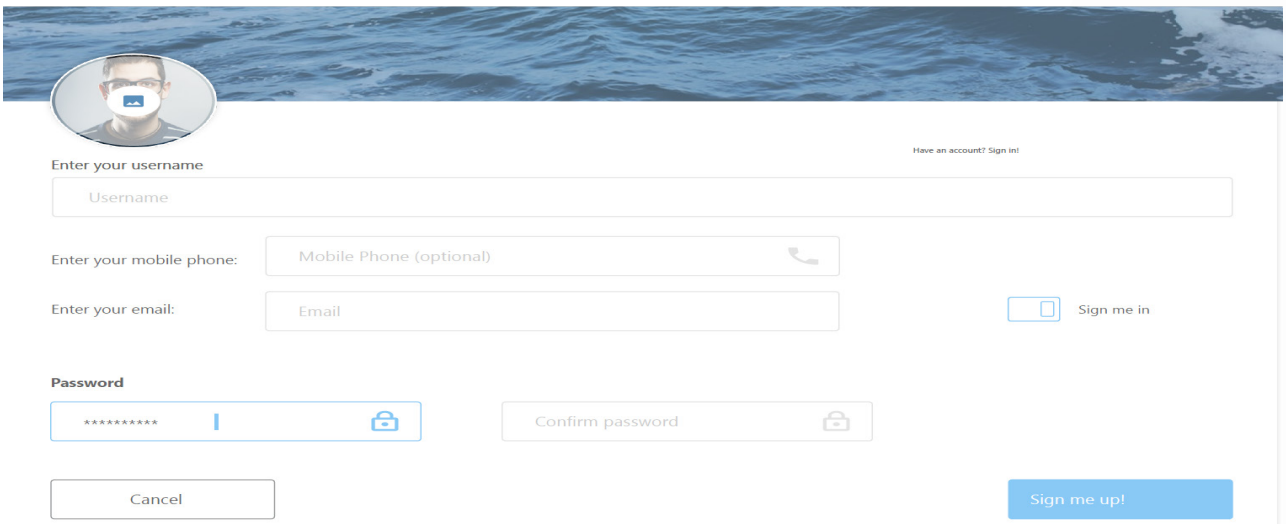
## 4.3

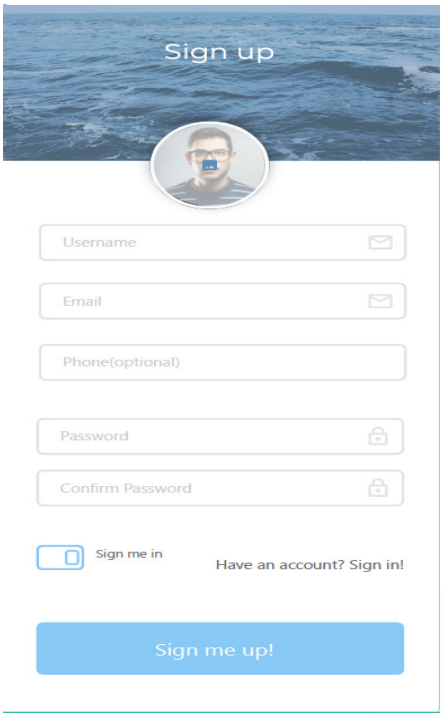
Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

## Higher Layers

User Interface Mockup



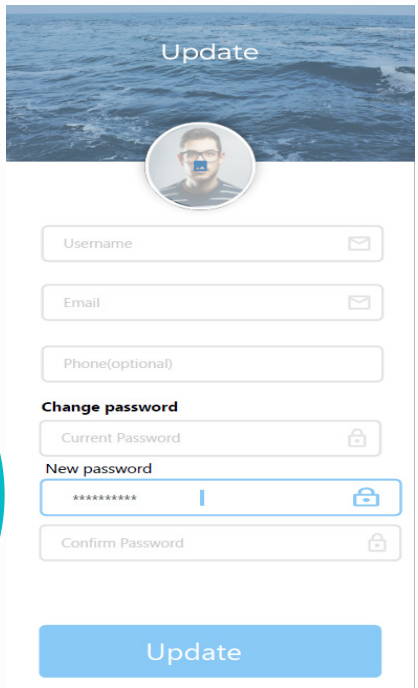



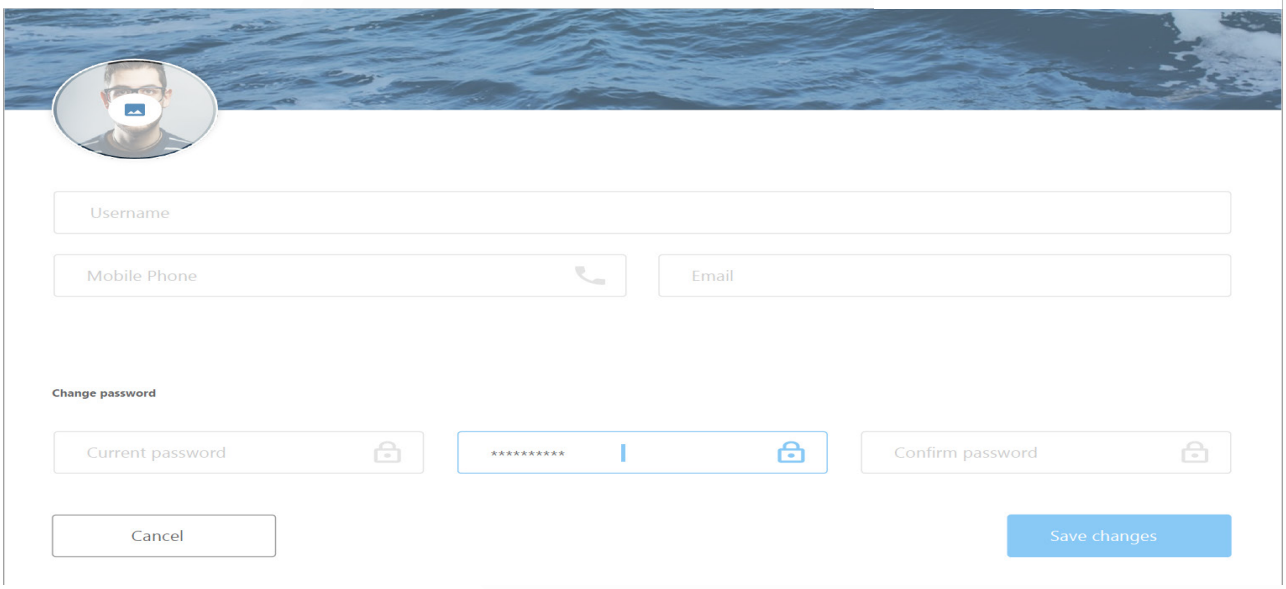

Mockup

**Sign up view**

Mockup

**Update profile view**



# 04

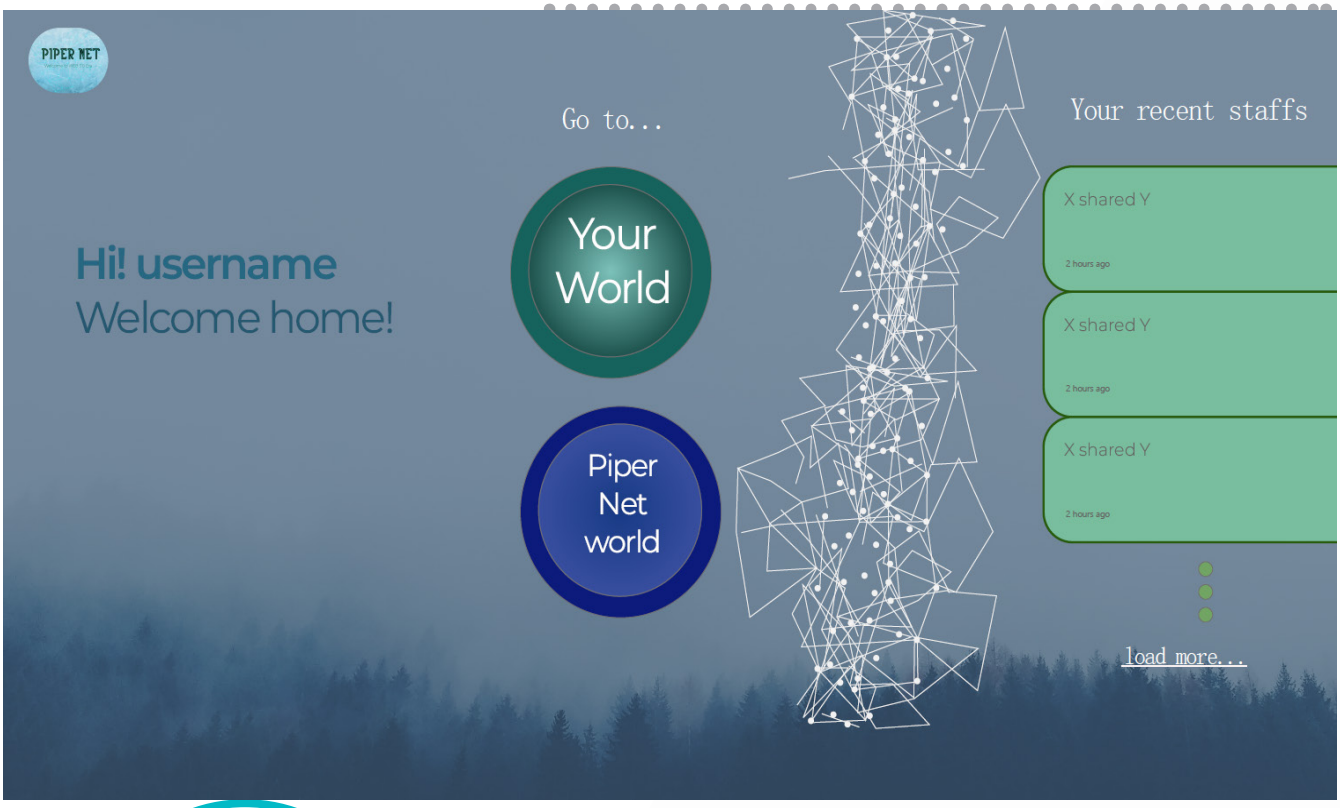
## 4.3

Project name:

**A blockchain-based web hosting scheme for content-addressed distributed file system**

## Higher Layers

User Interface Mockup

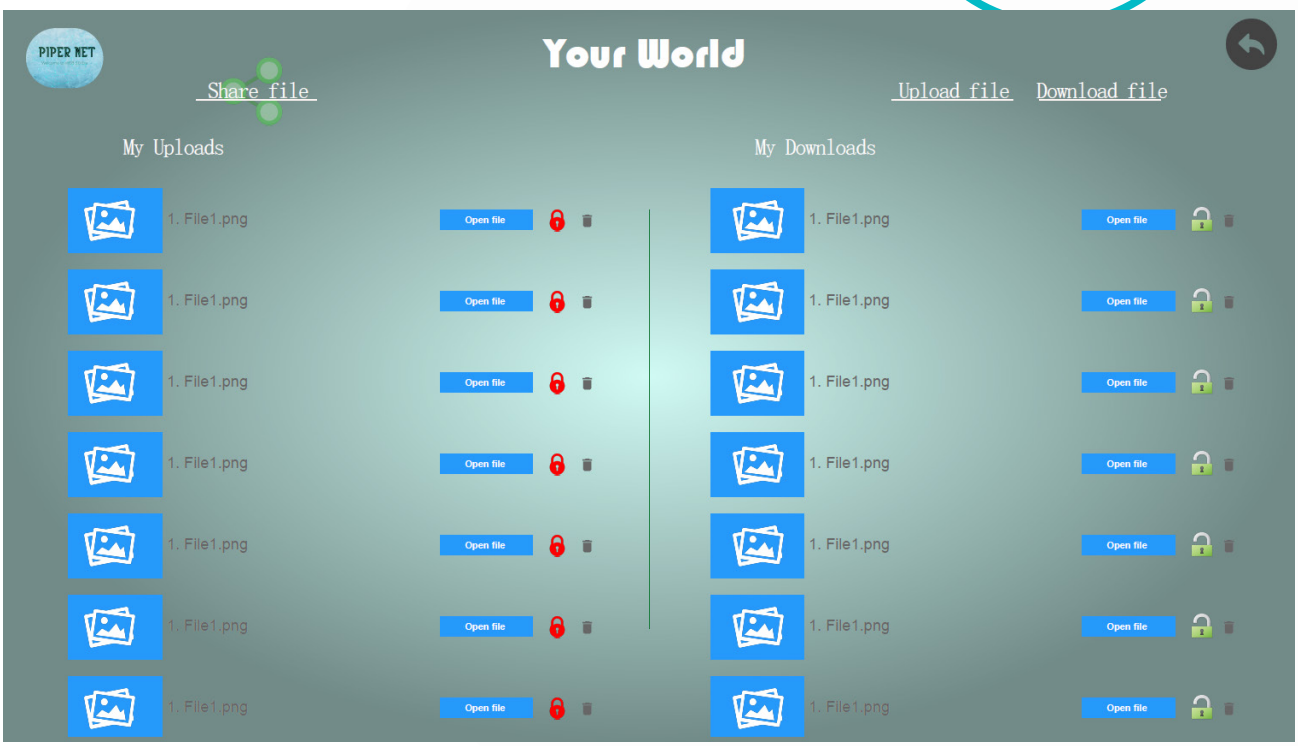



Mockup

## Home view

Mockup

## Your World view




IIT
University of Dhaka