

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**Analiză asupra InterPlanetary File System
Aplicația H.O.L.D.I.N.
(Help Over Local Distributed Interconnected Networks)**

propusă de

CRISTEA ALEXANDRU - GABRIEL

Sesiunea: iulie, 2017

Coordonator științific

Conf. Dr. ALBOAIE LENUȚA

**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI
FACULTATEA DE INFORMATICĂ**

**Analiză asupra InterPlanetary File System
Aplicația H.O.L.D.I.N.
(Help Over Local Distributed Interconnected Networks)**

CRISTEA ALEXANDRU - GABRIEL

Sesiunea: iulie, 2017

Coordonator științific

Conf. Dr. ALBOAIE LENUȚA

DECLARAȚIE PRIVIND ORIGINALITATE ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul “*Analiză asupra InterPlanetary File System, Aplicația H.O.L.D.I.N. (Help Over Local Distributed Interconnected Networks)*” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau din străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imaginile etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, iulie 2017

Absolvent Cristea Alexandru - Gabriel

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul “*Analiză asupra InterPlanetary File System, Aplicația H.O.L.D.I.N. (Help Over Local Distributed Interconnected Networks)*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea “Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, iulie 2017

Absolvent *Cristea Alexandru - Gabriel*

Cuprins

A. Introducere & Motivație	5
B. Tehnologii	6
1. InterPlanetary File System	7
1.1 Distributed Hash Tables	8
1.1.1 Kademlia DHT	8
1.1.2 Coral DSHT	8
1.1.3 S/Kademlia DHT	9
1.2 BitTorrent	10
1.3 Git	10
1.4 SFS (Self-Certified Filesystems)	11
1.5 Design	11
1.5.1 Identities	12
1.5.2 Network	15
1.5.3 Routing	21
1.5.4 Block Exchange	25
1.5.5 Merkle DAG & Files	27
1.5.6 IPNS (Naming and Mutable State)	33
2. Javascript IPFS API	35
3. Cordova & Phonegap	36
3.1 Cordova	36
3.2 Phonegap	38
3.3 Plugins	38
4. Vue.js	39
5. Framework7	39
6. NPM & Webpack	40
C. Arhitectură H.O.L.D.I.N.	41
1. Considerații preliminare	41
2. Scheletul general & detaliile implementării	42
D. Cazuri de utilizare	47
1. Pornire și “înregistrare”	47
2. Lipsa rețelei	48
3. Trimiterea și afișarea de mesaje	49
E. Concluzii	50
1. Direcții viitoare de cercetare & probleme întâlnite	50
2. Avantajele IPFS-ului	50
F. Bibliografie	51
G. Anexe	52

A. Introducere & Motivație

În zilele noastre una dintre problemele la care se caută cele mai rapide și sigure rezolvări este localizarea și comunicarea cu victimele catastrofelor naturale de orice tip: uragane, cutremure, inundații dar și alte evenimente dezastruoase. Internetul a “crescut” simțitor în societatea noastră în ultimele decenii, lăsându-și amprenta în orice domeniu. Multe dispozitive și sisteme de calcul sunt interconectate prin acesta dar ce se întâmplă atunci când ele nu au acces la rețeaua globală ? Spre exemplu: ce șanse are o persoană prinsă în subsolul unei clădiri, fără acces la rețeaua globală de internet sau la cea de telefonie mobilă ?

Ne-am putea imagina o lume în care orice persoană în pericol ar putea primi ajutor în cel mai scurt timp ? În practică, autoritățile nu fac mereu față la un număr mare de cereri, resursele de care dispun nu sunt suficiente ori tehnologiilor folosite în prezent le lipsesc multe caracteristici ce se regăsesc în alte inovații și astfel nu reușesc să-și ducă la bun sfârșit misiunea. Putem încerca să îmbunătățim toate aceste aspecte pornind de la tehnologie, aspect ce a cunoscut o evoluție exponențială în ultima vreme. Se pot îmbina noi idei, noi concepte și avantaje ale avansului tehnologic spre o îmbunătățire generală a societății în care trăim.

H.O.L.D.I.N. este o aplicație mobilă (Android) care vrea să rezolve aceste probleme folosind unul dintre cele mai noi protocoale apărute în sfera sistemelor de fișiere distribuite, InterPlanetary File System¹. Acesta din urmă dorește să “ia ce e mai bun” din alte protocoale și tehnologii și să le fuzioneze într-un protocol nou, modern ce ar putea revoluționa în viitor Internetul așa cum îl știm noi.

¹ "ipfs/ipfs - GitHub." <https://github.com/ipfs/ipfs>.

B. Tehnologii

De-a lungul timpului au existat diferite încercări de distribuire cât mai fezabilă, de încredere și sigură a datelor între mașinile active într-o rețea. Deși viteza de transfera este privită de utilizatorii obișnuiți ca fiind principala caracteristică a stocării și distribuirii de date, dezvoltatorii au căutat o cale de mijloc între aceasta, disponibilitate și siguranță.

Termeni generali:

- nod = entitate din cadrul unei rețele
- protocol = set de reguli creat pentru atingerea unui scop
- peer = nod ce respectă un set de protocoale pentru realizarea comunicării
- sistem de fișiere = sistem care stochează date oferind diverse funcționalități pentru administrarea lor
- sistem distribuit = sistem în care se partajează diverse resurse între mai multe entități
- repository (repo) = (cu referire la Git) entitate ce stochează date sub un anumit model

Un **sistem de fișiere distribuit**, la modul general, interconectează mai multe noduri într-o rețea în vederea partajării fișierelor și a datelor de interes comun. Câteva exemple cunoscute sunt: AFS² (Andrew File System), GFS³ (Google File System) și NFS⁴ (Network File System).

În altă ordine de idei, un sistem **Peer2Peer** este un sistem în care se încearcă desprinderea de un punct central de conexiune și partajarea de resurse computaționale sau de stocare. Deși există mai multe tipuri de sisteme P2P, cel care face obiectul analizei este unul **pur descentralizat**, în care nu există nici o entitate centrală de stocare sau de administrare ci nodurile “trăiesc” în mod autonom și interschimbă diverse informații de interese comune spre buna coabitare cu ceilalți și pentru ținerea în viață a “roiurilor” de noduri (swarms), a rețelei întregi.

² "The Andrew File System (AFS) - UW Computer Sciences User Pages."
<http://pages.cs.wisc.edu/~remzi/OSTEP/dist-afs.pdf>.

³ "The Google File System - googleusercontent.com."
<http://static.googleusercontent.com/media/research.google.com/ja//archive/gfs-sosp2003.pdf>.

⁴ "RFC 1094 - NFS: Network File System Protocol specification - IETF Tools."
<https://tools.ietf.org/html/rfc1094>

1. InterPlanetary File System

IPFS (InterPlanetary File System) este atât un sistem de fișiere cât și un protocol de viitor menit să partajeze date de interes comun, aplicații și diferite entități ce trăiesc în cadrul rețelei. Conceput de Juan Benet⁵ cu ținta principală de a “oferi” internetului un nou schelet, unul mai robust și mai de încredere, IPFS se află în continuare sub dezvoltare în regim open-source încă din anul 2014.

Conform comunității IPFS⁶:

Printre principalele caracteristici prin care IPFS-ul iese în evidență se numără:

- permanența datelor
- fundația construită peste Peer2Peer
- descentralizarea completă
- lipsa “punctelor centrale de eșec”
- conectivitate “locală” în lipsa conexiunii cu rețeaua globală.

HTTP este inefficient și “scump”, considerând că acesta descarcă fișiere în mod iterativ de la o singură sursă. IPFS face posibilă distribuirea volumelor mari de date cu eficiență crescută.

O mare parte din conținutul “vechi” al internetului este șters zilnic odată cu vechile versiuni ale fișierelor. IPFS vine în ajutor cu subsistemul său de versionare asemănător Git-ului.

Utilizatorii Internetului depind de punctele de centralizare care controlează funcționalitățile oferite. Astfel, în lipsa conectivității, utilizatorii nu au acces la date și la serviciile dorite 100% din timp. IPFS-ul dorește să spargă aceste bariere, fiind “motorul” unor rețele diverse și reziliente ce asigură disponibilitatea datelor cu sau fără conectivitate la rețeaua globală.

În continuare voi scoate în evidență aspectele teoretice și tehnice al protocolului, după cum sunt ele descrise în fișa tehnică oficială[1] de către creatorul Juan Benet, alături de observațiile personale rezultate din experimentele executate asupra protocolului în mai multe medii și cazuri.

⁵ "Juan Benet - juan.benet.ai." <http://juan.benet.ai/>.

⁶ "IPFS is the Distributed Web." <https://ipfs.io/>.

1.1. Distributed Hash Tables

Tabelele distribuite hash sunt folosite ca scop general pentru coordonarea și menținerea metadatelor într-un sistem P2P. Acestea rețin informații despre nodurile conectate la un anumit moment în rețea și oferă posibilitatea găsirii rapide și eficiente a informațiilor în funcție de diferite metrice.

1.1.1. Kademlia DHT

Kademlia DHT este un popular tabel de hash distribuit care joacă un rol important în cadrul IPFS-ului. Printre avantajele pe care le oferă Kademlia se numără [1]:

- căutarea eficientă în rețele mari
- costuri scăzute de coordonare (optimizarea numărului de mesaje de control trimise la alte noduri)
- rezistență la diferite atacuri (preferă nodurile cu longevitate crescută)
- utilizarea exhaustivă în aplicații P2P renumite (Gnutella⁷ și BitTorrent)

Kademlia⁸ utilizează perechi de tipul (*key*, *value*) pentru stocarea datelor. *Key* (160-bit) identifică unic atât datele cât și nodurile în rețea. Fiecare mașină posedă o astfel de cheie (*NodeId*) și de asemenea reține informații de contact despre ceilalți: IP Address, port UDP și *NodeId*. Pe lângă acestea, DHT-ul se folosește de *metrica XOR* pentru calculul “distanței”. Având la bază arbori binari, pentru a determina unde se stochează o anumită pereche (*key*, *value*), Kademlia realizează distanța dintre doi identificatori ca fiind întregul operației XOR dintre aceștia. Algoritmul de *Lookup* caută succesiv “cele mai apropiate” noduri dat fiind un anumit ID astfel convergând la ținta lookup-ului într-un număr logaritmic de pași.

1.1.2. Coral DSHT

Coral DSHT vine sub forma unei extinderi a tabelor de hash distribuite Kademlia prin următoarele caracteristici [1]:

- Kademlia stochează valori în nodurile ale căror *NodeId*-uri sunt cele mai “apropiate” față de cheie în concordanță cu distanța XOR, ignorând astfel nodurile “îndepărtate” care ar putea avea deja valorile respective și forțează nodurile “cele mai apropiate” să o stocheze, risipind capacitatea de stocare și lățimea de bandă;

⁷ "The World of Peer-to-Peer (P2P)/Networks and Protocols/Gnutella".

[https://en.wikibooks.org/wiki/The_World_of_Peer-to-Peer_\(P2P\)/Networks_and_Protocols/Gnutella](https://en.wikibooks.org/wiki/The_World_of_Peer-to-Peer_(P2P)/Networks_and_Protocols/Gnutella)

⁸ "An Introduction to Kademlia DHT & How It Works | Gleamly."

<http://gleamly.com/article/introduction-kademlia-dht-how-it-works>.

Coral DSHT stochează adresele la noduri care pot distribui practic datele respective.

- Coral este “*sloppy*”⁹: cheile nu sunt mereu stocate la nodurile “cele mai apropiate”, poate distribui numai bucăți ale valorilor la “cele mai apropiate” noduri evitând astfel supraîncărcarea lor când cererea unei anumite chei devine ridicată
- În funcție de regiune și de zonă Coral organizează o serie de “*cluster*e” separate, permițând nodurilor să realizeze căutarea în zona proprie în primii pași ai căutării

1.1.3. S/Kademlia DHT

S/Kademlia oferă un plus de securitate împotriva atacurilor malițioase prin forțarea nodurilor de a-și genera o pereche de chei PKI¹⁰ pe care să o folosească ulterior la generarea identității și în semnarea de mesaje. Una dintre metodele de securizare presupune un “puzzle criptografic”. De asemenea, căutarea valorilor în S/Kademlia se face pe drumuri disjuncte permițând nodurilor de încredere să comunice în prezența adversarilor [1].

În practică, merită aduse în evidență unele observații asupra modului cum interacționează DHT-urile în cadrul implementărilor protocolului:

În prezent există 2 versiuni ale protocolului ce pot fi folosite în implementare: Go[15] (varianta de referință) și Javascript[16] (varianta mai puțin completă decât cea de Go).

În varianta Go, tehnologiile sunt implementate cu succes, ajutând astfel la descoperirea de noi noduri în vederea realizării conexiunilor dar și de obținere a informațiilor despre locația blocurilor de date.

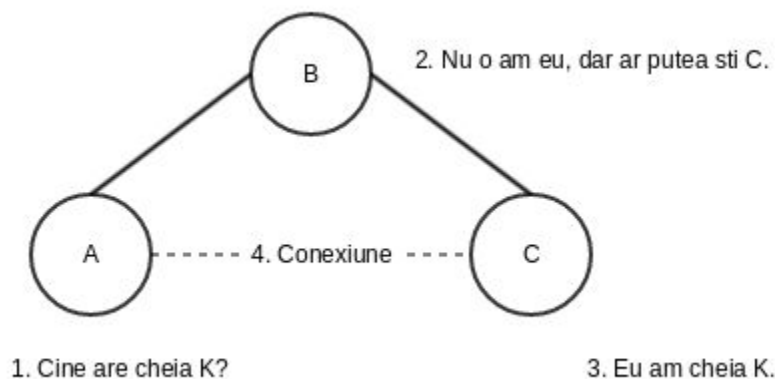


Figura B.1: Realizarea conexiunilor prin discovery

⁹ "Presentation (ppt) - AUEB OpenCourses."

https://opencourses.aueb.gr/modules/document/file.php/INF110/Presentations/PPT/Coral_cdn_pres.pptx.

¹⁰ "8 The Public Key Infrastructure Approach to Security."

https://docs.oracle.com/cd/B12037_01/network.101/b10777/pki.htm

În (Figura B.1) este prezentat la un nivel foarte abstract și concis felul cum se realizează descoperirea în IPFS. La primul pas, *nodul A* cere rețelei informația identificată prin *cheia K*. Considerând că singura conexiune directă pe care o are activă este cu *nodul B*, cererea ajunge numai la acesta. *Nodul B* nu dispune de cheia respectivă dar propagă cererea mai departe spre *nodul C* cu care *B* are o conexiune directă iar acesta răspunde afirmativ. Astfel, se realizează conexiunea cu nodul respectiv iar *A* va primi informația dorită de la *C*.

1.2. BitTorrent

BitTorrent¹¹ este probabil cel mai popular dintre sistemele de distribuire de fișiere P2P, aflându-se în folosință într-o gamă largă de ecosisteme software, cele mai cunoscute fiind tracker-ele de torrent pentru distribuire mai mult sau mai puțin legală de conținut. Acesta pleacă de la ideea că nodurile în rețeaua de distribuire nu au nevoie de încredere unul în celălalt dar pot interschimba datele în siguranță. Enumerăm [1] principalele avantaje de care IPFS se bucură:

- protocolul de schimb de date are la bază o pseudo-strategie “*tit-for-tat*”: nodurile ce contribuie la rețea sunt răsplătite în timp ce acele noduri care doar depind de altele sunt pedepsite
- prioritizează transferul blocurilor “rare” de informații (cele ce pot fi distribuite de un număr mai scăzut de noduri)
- strategia standard “*tit-for-tat*” este uneori vulnerabilă; *PropShare*¹² este propusă ca o alternativă ce rezistă la strategii abuzive asupra lărimii de banda

1.3. Git

IPFS împrumută de la **Git**¹³ tehnologia necesară stocării datelor proprii și identificarea acestora prin hash-uri unice. Sistemul de versionare Git oferă o metodă eficientă de organizare a entităților (fișiere, directoare) și a diferitelor versiuni ale lor în timp. Având la bază Arborele Aciclic Direcționat Merkle (**Merkle DAG**¹⁴) acesta reprezintă un mecanism puternic de modelare a datelor și a schimbărilor ce pot apărea asupra lor [1]:

- entități: obiecte imutabile (fișiere) “blob”, “tree” (foldere), “changes” (schimbări)
- obiectele sunt adresate hash de către propriul conținut, legăturile cu entitățile ascendente sau descendente fiind salvate în interiorul acestora

¹¹ "BitTorrent Protocol Specification - BitTorrent.org."

http://www.bittorrent.org/beps/bep_0003.html.

¹² "PropShare." <http://www.cs.umd.edu/projects/propshare/>.

¹³ "Git-About Version Control." <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>.

¹⁴ "Merkle Tree | Brilliant Math & Science Wiki." <https://brilliant.org/wiki/merkle-tree/>

- informații privitoare la metadata sunt reținute ca simple “referințe”, schimbările asupra versiunilor actualizând referințele sau adăugând noi obiecte; distribuția schimbărilor revine doar la simplul transfer de date și actualizarea referințelor “de la depărtare”

1.4. SFS (Self-Certified Filesystems)

IPFS folosește ideea de “sistem de fișiere auto-certificat” prin utilizarea schemei[1]:

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

Unde `Location` reprezintă adresa de rețea a serverului iar `HostID` este rezumatul hash asupra cheii publice a nodului împreună cu `Location` respectiv. Astfel nodurile pot verifica respectiva cheie, negocia un secret și securiza tot traficul. Acestea nu depind de o entitate centrală ci toate împart același spațiu de nume global pentru o alocare criptografică de identități.

1.5. Design

IPFS din punct de vedere arhitectural îmbină avantajele și caracteristicile mai multor tehnologii moderne, fie ca e vorba de tabele hash distribuite, Git, BitTorrent sau PubSub. Fiecare dintre acestea au individual atât aspecte pozitive cât și negative, accentul căzând pe cele pozitive. IPFS încearcă să fie suma tuturor tehnologiilor și a avantajelor, preluând de la fiecare paradigme și metodologii ce se leagă strâns într-un singur sistem de fișiere distribuit.

Observatie: În cele ce urmează va fi folosită implementarea Go a sistemului, cea de referință care dispune de majoritatea proprietăților concepute și care oferă un API la linia de comandă (Figura B.2) (în cazul de față pentru sistemul de operare Linux) dar și secvențe de cod din varianta Javascript. Comunitatea IPFS a conceput o interfață generică [5] pentru viitoarele implementări ale IPFS-ului, una pe care în momentul de față numai varianta Javascript a protocolului și API-ul Javascript o implementează într-o oarecare măsură, urmând pe viitor ca și restul să o utilizeze.

```

xsky@info ~ $ ipfs help
USAGE
  ipfs - Global p2p merkle-dag filesystem.

  ipfs [--config=<config>] [-c] [--debug=<debug>] [-D] [--help=<help>] [-h=<h>] [--local=<local>] [-L] [--api=<api>] <command> ...

SUBCOMMANDS
BASIC COMMANDS
  init      Initialize ipfs local configuration
  add <path> Add a file to IPFS
  cat <ref>  Show IPFS object data
  get <ref>  Download IPFS objects
  ls <ref>   List links from an object
  refs <ref> List hashes of links from an object

DATA STRUCTURE COMMANDS
  block      Interact with raw blocks in the datastore
  object     Interact with raw dag nodes
  files      Interact with objects as if they were a unix filesystem
  dag        Interact with IPLD documents (experimental)

ADVANCED COMMANDS
  daemon     Start a long-running daemon process
  mount      Mount an IPFS read-only mountpoint
  resolve    Resolve any type of name
  name       Publish and resolve IPNS names
  key        Create and list IPNS name keypairs
  dns        Resolve DNS links
  pin        Pin objects to local storage
  repo       Manipulate the IPFS repository
  stats      Various operational stats
  filestore  Manage the filestore (experimental)

NETWORK COMMANDS
  id         Show info about IPFS peers
  bootstrap  Add or remove bootstrap peers
  swarm      Manage connections to the p2p network
  dht        Query the DHT for values or peers
  ping       Measure the latency of a connection
  diag       Print diagnostics

TOOL COMMANDS
  config     Manage configuration
  version    Show ipfs version information
  update     Download and apply go-ipfs updates
  commands   List all available commands

Use 'ipfs <command> --help' to learn more about each command.

ipfs uses a repository in the local file system. By default, the repo is located
at ~/.ipfs. To change the repo location, set the $IPFS_PATH environment variable:

  export IPFS_PATH=/path/to/ipfsrepo

EXIT STATUS

The CLI will exit with one of the following values:

0    Successful execution.
1    Failed executions.

```

Figura B.2: ipfs help (Go CLI)

Scopul final al IPFS-ului este sa nu trateze diferit nici un nod din propria rețea, lăsând posibilitatea de stocare, regăsire și transfer al datelor existente local la fiecare nod în parte. Stiva protocoalelor existente în cadrul IPFS-ului se prezintă după cum urmează:

1.5.1. Identities

Fiecare nod în IPFS deține un `NodeId` prin care se identifică unic în rețea. Valoarea respectivă este generată la inițializarea locală în mod criptografic folosind “puzzle-ul” oferit de S/Kademlia.

Inițial, nodurile își generează propriile perechi de chei PKI¹⁵ (publica, privată) stocându-le criptat local. Asupra cheii publice se aplică o funcție de hash din formatul multihash (format ce conține specificarea funcției folosite, lungimea rezumatului și rezumatul propriu-zis) rezultând astfel `NodeId`-ul nodului respectiv [1] (Anexa 2).

¹⁵ "8 The Public Key Infrastructure Approach to Security."
https://docs.oracle.com/cd/B12037_01/network.101/b10777/pki.htm

```

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)

```

Figura B.3: generarea de identitate

Se poate observa în (Figura B.3)[1] un exemplu de pseudocod al generării de identitate. Acesta rulează o buclă în funcție de nivelul de dificultate al “puzzle-ului criptografic” prin repetarea generării perechilor de chei și a identității până la obținerea unui anumit număr de biți de 0 din începutul rezultatului hash asupra `NodeId`-ului.

La realizarea unei conexiuni, nodurile își verifică reciproc identitățile prin aplicarea funcției de hash asupra cheii publice a partenerului și comparând-o cu `NodeId`-ul respectiv. Formatul `multihash` oferă elasticitate în alegerea unei funcții specifice diferitelor cazuri.

```

xsky@inf ~ $ ipfs init --help
USAGE
  ipfs init [<default-config>] - Initializes ipfs config file.

SYNOPSIS
  ipfs init [--bits=<bits> | -b] [--empty-repo | -e] [--] [<default-config>]

ARGUMENTS

  [<default-config>] - Initialize with the given configuration.

OPTIONS

  -b, --bits      int    - Number of bits to use in the generated RSA private key. Default: 2048.
  -e, --empty-repo bool - Don't add and pin help files to the local storage. Default: false.

DESCRIPTION

  Initializes ipfs configuration files and generates a new keypair.

  ipfs uses a repository in the local file system. By default, the repo is
  located at ~/.ipfs. To change the repo location, set the $IPFS_PATH
  environment variable:

  export IPFS_PATH=/path/to/ipfsrepo

```

Figura B.4: comanda de inițializare IPFS (Go CLI)

```

xsky@inf ~ $ ipfs init
initializing IPFS node at /home/xsky/.ipfs
generating 2048-bit RSA keypair...done
peer identity: QmWB7iunGuHBAFzHZqbDpssdAifMRfTn7tNHDQb1h1Dojq
to get started, enter:

  ipfs cat /ipfs/QmVLDahCY3X9P2uRudKArYuQFPM5zqR3Yij1dY8FpGbL7T/readme

```

Figura B.5: inițializarea IPFS (Go CLI)

Înainte de utilizarea sistemului, utilizatorii trebuie să ruleze comanda `ipfs init` ce crează mediul de stocare local necesar împreună cu perechea de chei și identitatea proprie folosind criptosistemul RSA 2048-bit. În imaginea de la (Figura B.4) este descrisă aceasta

împreună cu parametrii de rigoare în timp ce la (Figura B.5) se poate vedea output-ul comenzii ce afișează identitatea rezultată a nodului și locația repo-ului local personal.

```
const node = new IPFS({
  repo: repo,
  init: true, // default
  // init: false,
  // init: {
  //   bits: 1024 // size of the RSA key generated
  // },
  start: true,
  // start: false,
  EXPERIMENTAL: { // enable experimental features
    pubsub: true,
    sharding: true,
    // enable dir sharding
    wrtcLinuxWindows: true
    // use unstable wrtc module on Linux or Windows with Node.js,
    dht: true
    // enable KadDHT, currently not interoperable with go-ipfs
  },
  config: { // overload the default config
    Addresses: {
      Swarm: [
        '/ip4/127.0.0.1/tcp/1337'
      ]
    }
  }
});
```

Figura B.6: inițializarea IPFS (Javascript)

Varianța de Javascript oferă un API aproximativ asemănător celui Go iar în (Figura B.6)¹⁶ se poate observa o inițializare direct din cod cu diferiți parametri prezenți în interfața de programare pentru inițializare.

O altă comandă ce merită specificată este `ipfs config`. Aceasta permite realizarea diferitelor configurări după inițializare precum permisivitatea conexiunilor la daemon-ul IPFS de către API-uri implementate în diverse limbaje (JS, Python, Java etc.) (Figura B.7).

¹⁶ "ipfs js - GitHub." <https://github.com/ipfs/js-ipfs>.

```

xsky@inf ~ $ ipfs config --help
USAGE
  ipfs config <key> [<value>] - Get and set ipfs config values.

SYNOPSIS
  ipfs config [--bool] [--json] [--] <key> [<value>]

ARGUMENTS

  <key>      - The key of the config entry (e.g. "Addresses.API").
  [<value>]  - The value to set the config entry to.

OPTIONS

  --bool bool - Set a boolean value. Default: false.
  --json bool - Parse stringified JSON. Default: false.

DESCRIPTION

  'ipfs config' controls configuration variables. It works
  much like 'git config'. The configuration values are stored in a config
  file inside your IPFS repository.

  Examples:

  Get the value of the 'Datastore.Path' key:

    $ ipfs config Datastore.Path

  Set the value of the 'Datastore.Path' key:

    $ ipfs config Datastore.Path ~/.ipfs/datastore

SUBCOMMANDS
  ipfs config edit      - Open the config file for editing in $EDITOR.
  ipfs config replace <file> - Replace the config with <file>.
  ipfs config show      - Output config file contents.

  Use 'ipfs config <subcmd> --help' for more information about each command.

```

Figura B.7: comanda de configurare IPFS (Go CLI)

1.5.2. Network

Nodurile IPFS comunică în rețea cu sute de alte noduri, fie că e vorba de transfer de date, descoperire sau mesaje de control. Printre caracteristicile necesare rețelei se numără [1]:

- **transport**: sistemul poate folosi diverse protocoale de transport: TCP, WebRTC¹⁷, uTP¹⁸ etc.
- **încredere**: sistemul oferă încredere acolo unde este necesar
- **conectivitate**: asigură conectivitate între noduri din rețea ce se află în spatele altor subrețele prin diverse procedee precum ICE NAT¹⁹
- **integritate**: se face verificarea mesajelor prin sume de control
- **autenticitate**: se poate verifica autenticitatea mesajelor folosind sume de control HMAC

¹⁷ "WebRTC API - Web APIs | MDN."

https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API.

¹⁸ "uTorrent Transport Protocol - BitTorrent.org." http://www.bittorrent.org/beps/bep_0029.html

¹⁹ "RFC 5245 - Interactive Connectivity Establishment (ICE): A" <https://tools.ietf.org/html/rfc5245>.

Atât pentru rețea cât și pentru rutare ecosistemul folosește pachetul **libp2p**²⁰, acesta oferind o gamă largă de paradigme și funcționalități ce asigură toate aceste caracteristici.

Protocolul poate folosi diferite alte protocoale de transport precum cele menționate mai sus și în același timp prin folosirea format-ului `multiaddr` pentru toate cazurile posibile de implementare. IPFS nu presupune o neapărată folosire a protocolului IP pentru adresarea peer-urilor, acesta fiind unul din avantajele folosirii format-ului respectiv. (Figura B.8)[3]

```
# IPFS over TCP over IPv6 (typical TCP)
/ip6/fe80::8823:6dff:fee7:f172/tcp/4001/ipfs/QmYJyUMAcXEw1b5bFfbBbzYu5wyyjLMRHXGUkCXpag74Fu

# IPFS over uTP over UDP over IPv4 (UDP-shimmed transport)
/ip4/162.246.145.218/udp/4001/utp/ipfs/QmYJyUMAcXEw1b5bFfbBbzYu5wyyjLMRHXGUkCXpag74Fu

# IPFS over IPv6 (unreliable)
/ip6/fe80::8823:6dff:fee7:f172/ipfs/QmYJyUMAcXEw1b5bFfbBbzYu5wyyjLMRHXGUkCXpag74Fu
```

Figura B.8: diferite tipuri de adrese multiaddr

Alături de agnosticismul adreselor și a protocoalelor, `libp2p` presupune și [2]:

- **multiplexare**: spre exemplu pentru multiple conexiuni particulare prin diferite protocoale transport, spre diferite noduri, folosind diferite tipuri de criptare etc.
- **encripție**: comunicarea poate fi criptată, semnată sau în clar, oferind astfel un echilibru între performanță și securitate
- **traversare NAT**: folosirea unei metodologii asemănătoare protocolului ICE (Interactive Connectivity Establishment) ce permite comunicarea între două noduri aflate în spatele unui NAT din motive de securitate sau virtualizare
- **relay**: mecanism utilizat pentru diferite tipuri de NAT ce ridică probleme în traversare ori alte cazuri în care descoperirea și comunicarea nu este posibilă în mod uzual (ex. NAT simetric²¹) și presupune existența unei entități (ex. server) ce stabilește conexiuni între clienți aflați în arhitecturi de rețea ce ridică astfel de probleme
- **permiterea conectivității** cu diferite topologii de rețea (nestructurată, structurată, hibridă, centralizată)
- **descoperirea de resurse** indiferent de localizarea lor
- **trimiterea de “mesaje”**: folosind `multicast` sau `PubSub`
- **spațiul de nume**: rezolvă cazurile în care rețelele se schimbă în timp și aplicațiile au nevoie să funcționeze corect în cadrul rețelei

²⁰ "GitHub - libp2p/specs" <https://github.com/libp2p/specs>.

²¹ "Symmetric NAT and It's Problems | Think Like A Computer."
<https://www.think-like-a-computer.com/2011/09/19/symmetric-nat/>.

În continuare se vor exemplifica cazuri practice ale diferitelor implementări relativ la partea de rețea a protocolului (Anexa 2).

Daemonii IPFS, în implementările curente, expun 3 port-uri diferite la inițializare printre care unul de comunicare cu rețeaua. În implementarea Go, comanda `ipfs daemon` ridică un astfel de daemon ce conectează mașina respectivă la rețea, permițând descoperirea și transferul de date.

```
xsky@inf ~$ ipfs daemon --help
USAGE
  ipfs daemon - Run a network-connected IPFS node.

SYNOPSIS
  ipfs daemon [--init] [--routing=<routing>] [--mount] [--writable] [--mount-ipfs=<mount-ipfs>] [--mount-ipns=<mount-ipns>] [--unrestricted-api] [--disable-transport-encryption] [--enable-gc] [--manage-fdlimit=false] [--offline] [--migrate] [--enable-pubsub-experiment] [--enable-mplex-experiment=false]

OPTIONS
  --init                bool    - Initialize ipfs with default settings if not already initialized. Default: false.
  --routing             string   - Overrides the routing option. Default: dht.
  --mount              bool     - Mounts IPFS to the filesystem. Default: false.
  --writable            bool     - Enable writing objects (with POST, PUT and DELETE). Default: false.
  --mount-ipfs         string   - Path to the mountpoint for IPFS (if using --mount). Defaults to config setting.
  --mount-ipns         string   - Path to the mountpoint for IPNS (if using --mount). Defaults to config setting.
  --unrestricted-api   bool     - Allow API access to unlisted hashes. Default: false.
  --disable-transport-encryption bool - Disable transport encryption (for debugging protocols). Default: false.
  --enable-gc          bool     - Enable automatic periodic repo garbage collection. Default: false.
  --manage-fdlimit     bool     - Check and raise file descriptor limits if needed. Default: true.
  --offline           bool     - Run offline. Do not connect to the rest of the network but provide local API. Default: false.
  --migrate            bool     - If true, assume yes at the migrate prompt. If false, assume no.
  --enable-pubsub-experiment bool - Instantiate the ipfs daemon with the experimental pubsub feature enabled.
  --enable-mplex-experiment bool - Add the experimental 'go-multiplex' stream muxer to libp2p on construction. Default: true.

DESCRIPTION

The daemon will start listening on ports on the network, which are documented in (and can be modified through) 'ipfs config Addresses'.
For example, to change the 'Gateway' port:

  ipfs config Addresses.Gateway /ip4/127.0.0.1/tcp/8082

The API address can be changed the same way:

  ipfs config Addresses.API /ip4/127.0.0.1/tcp/5002

Make sure to restart the daemon after changing addresses.

By default, the gateway is only accessible locally. To expose it to other computers in the network, use 0.0.0.0 as the ip address:

  ipfs config Addresses.Gateway /ip4/0.0.0.0/tcp/8080

Be careful if you expose the API. It is a security risk, as anyone could control your node remotely. If you need to control the node remotely, make sure to protect the port as you would other services or database (firewall, authenticated proxy, etc).
```

Figura B.9: comanda de ridicare a conexiunii / daemon-ului (Go CLI)

În (Figura B.9) este prezentată o parte din manualul comenzii. Se pot observa parametrii acceptați de către aceasta și diferite alte specificații și detalii de rulare.

```
xsky@inf ~ $ ipfs daemon --enable-pubsub-experiment
Initializing daemon...
Adjusting current ulimit to 2048...
Successfully raised file descriptor limit to 2048.
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/172.17.0.1/tcp/4001
Swarm listening on /ip4/172.18.0.1/tcp/4001
Swarm listening on /ip4/172.19.0.1/tcp/4001
Swarm listening on /ip4/172.20.0.1/tcp/4001
Swarm listening on /ip4/192.168.100.3/tcp/4001
Swarm listening on /ip4/192.168.100.4/tcp/4001
Swarm listening on /ip4/92.86.209.159/tcp/41796
Swarm listening on /ip6:::1/tcp/4001
API server listening on /ip4/127.0.0.1/tcp/5001
Gateway (readonly) server listening on /ip4/127.0.0.1/tcp/8080
Daemon is ready
```

Figura B.10: ridicarea daemon-ului (Go CLI)

(Figura B.10) reprezintă rularea efectivă a comenzii împreună cu opțiunea `--enable-pubsub-experiment`. Aceasta instanțiază daemon-ul IPFS și scoate în evidență cele 3 porturi standard implementării Go: 4001 (portul de “swarm”, cel prin care se realizează comunicarea cu rețeaua împreună cu diferitele adrese IPv4 și IPv6 în acest caz), 5001 (portul ce permite conexiunea la daemon cu ajutorul unui API), 8080 (portul de Gateway). Un lucru ce merită notat este portul adresat automat (41796) de către sistem pentru IP-ul extern. Aceste porturi pot fi customizate utilizând comanda `ipfs config`. În varianta Javascript, porturile standard sunt 4002 (swarm) și 5002 (API), inițializarea făcându-se precum în (Figura B.6).

```
xsky@inf ~ $ ipfs id
{
  "ID": "QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ",
  "PublicKey": "CARSPglwggEiMA0GCSqGSIb3DQEBBAQUAAIIBDwAwggEKAoIBAQCv2Q3uuFQWw+JQy589XpiyBxFTW1Lq93x9csaiREb1Fu4M2CTEXHyC/pz jAG880F
c7Sgu007b4MHAJgW jlvLw3EbQw1M0aH0030PckwQHA83c14eg/6Pb01+MF+9H7XN5 jkHIdpZwcmSG2dZYxqcIYbot/MDD8pu2Q8m6iPSQM1WvaxBwgordFTYNcb8M0Era1WQTxTPM
fCMAQNKcYyZ90zeU9Adz1UE9KYwLfgb3HNxxdHRC7T+0k+dr1SLZn0FDNFfvPUQ2Tb43hie9MJfLcLvliqqP fT3CkoIf+i17/nIovlpWohRrcWum tUFJADXsvynA6 jUMSGOLsN
GLpAgMBAE=",
  "Addresses": [
    "/ip4/127.0.0.1/tcp/4001/ipfs/QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ",
    "/ip4/192.168.100.3/tcp/4001/ipfs/QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ",
    "/ip4/192.168.100.4/tcp/4001/ipfs/QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ",
    "/ip4/172.20.0.1/tcp/4001/ipfs/QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ",
    "/ip4/172.17.0.1/tcp/4001/ipfs/QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ",
    "/ip4/172.19.0.1/tcp/4001/ipfs/QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ",
    "/ip4/172.18.0.1/tcp/4001/ipfs/QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ",
    "/ip6:::1/tcp/4001/ipfs/QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ",
    "/ip4/92.86.209.159/tcp/19420/ipfs/QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ",
    "/ip4/92.86.209.159/tcp/19420/ipfs/QmMB7iunGuHBAFzHZqbDpssdAi fMRf tN7 tNHdQb1h1Do jQ"
  ],
  "AgentVersion": "go-ipfs/0.4.9-dev/1c50ec8c",
  "ProtocolVersion": "ipfs/0.1.0"
}
```

Figura B.11: informații despre nodul curent (Go CLI)

```
ipfs.id(function (err, identity) {
  if (err) {
    throw err
  }
  console.log(identity)
});
```

Figura B.12[5]: informații despre nodul curent (Javascript)

(Figura B.11) (Figura B.12) Se pot afla informații despre nodul care este activ pe mașina curentă rulând comanda `ipfs id` precum `NodeID`, `PublicKey`, `Addresses` etc.

Analiza nodurilor cu care a fost realizată o conexiune activă într-un anumit moment de timp se poate face utilizând comanda `ipfs swarm <subcmd>`, utilă cât și pentru conectarea / deconectarea manuală. (Figura B.13) (Figura B.14) (Figura B.15)

```
xsky@inf ~ $ ipfs swarm --help
USAGE
  ipfs swarm - Interact with the swarm.

SYNOPSIS
  ipfs swarm

DESCRIPTION

  'ipfs swarm' is a tool to manipulate the network swarm. The swarm is the
  component that opens, listens for, and maintains connections to other
  ipfs peers in the internet.

SUBCOMMANDS
  ipfs swarm addrs          - List known addresses. Useful for debugging.
  ipfs swarm connect <address>... - Open connection to a given address.
  ipfs swarm disconnect <address>... - Close connection to a given address.
  ipfs swarm filters        - Manipulate address filters.
  ipfs swarm peers          - List peers with open connections.

  Use 'ipfs swarm <subcmd> --help' for more information about each command.
```

Figura B.13: comanda ipfs swarm (Go CLI)

```
xsky@inf ~ $ ipfs swarm peers
/ip4/103.57.83.162/tcp/16537/ipfs/QmSmDBeHnBmHDcaF5Qz4wBcPMvZKiSnWHak7inJv95Ecr3
/ip4/104.131.131.82/tcp/4001/ipfs/QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbUtfsmvsqQLuvuJ
/ip4/104.131.20.170/tcp/4001/ipfs/QmQG8e6xjqbwK7BCURayW7X6YahbCognPuAeqsVpnnVifS
/ip4/104.133.2.68/tcp/56008/ipfs/QmTAmvzNBsicnaJpLTUnVqcPankP3pNDogHpAtUNkK2rU7
```

Figura B.15: exemplu de output pentru analiza nodurilor cu care există conexiuni (Go CLI)

```
ipfs.swarm.peers(function (err, peerInfos) {
  if (err) {
    throw err
  }
  console.log(peerInfos)
})
```

Figura B.14[5]: afișarea de informații despre peers (Javascript)

Un alt lucru ce merită menționat este utilizarea suitei de protocoale **uPNP**²². Acestea oferă o metodă de autoconfigurare a mașinilor conectate la subrețeaua respectivă. Deși în unele cazuri reprezintă un posibil risc de securitate, uPNP permite daemon-ului IPFS autoconfigurarea în cazul dependenței de un router local, lăsând posibilitatea de a-și deschide porturile necesare comunicării.

²² "What is Universal Plug and Play (UPnP)? - Definition from WhatIs.com."
<http://whatis.techtarget.com/definition/Universal-Plug-and-Play-UPnP>.

```
xsky@inf:~  
"Containers": {  
  "002e1aea32756b7e71b697b80656ad3b3f745fa451fb9321625b94f713a91028": {  
    "Name": "my_ipfs1",  
    "EndpointID": "c96c53d4d75d043441bfad25fd73c2502926d15235a41bf77732f83a390f87c7"  
  },  
  "9f460f72ccf5a40d2b878511a9e8650ba5dc3ce3a1bec32f5536d029aa891043": {  
    "Name": "my_ipfs3",  
    "EndpointID": "6e6b3af316de94ab637b22b68ace56cf6479424ffeee6682c061cc74a2aa991f"  
  },  
  "de153aec0bfd892d1c87a541e3603e5268a8e9b51e8aca3351040b2819139070": {  
    "Name": "my_ipfs2",  
    "EndpointID": "a1d8bf687fb93e6e64063c76ca0c7c871ad56b7ecbccc9c25e21dafb0fdecdb5"  
  },  
  "Options": {},  
  "Labels": {}  
}  
xsky@inf ~ $  
  
root@my_ipfs3: /ipfs  
root@my_ipfs3:/ipfs# ipfs id -f=<id>\n"  
QmcAtiPs5Qp6Q8PHCcwtx8zHYuuwFRWen57WR6dJK2xNnF  
root@my_ipfs3:/ipfs# ipfs swarm peers  
/ip4/172.18.0.1/tcp/4001/ipfs/QmWB7iunGuHBAFzHZqbDpssdAifMRFtn7tNHDQb1h1DojQ  
/ip4/172.19.0.2/tcp/4001/ipfs/QmWDypxupuDUdczDW4MBphoyZ499vmdRTBF4REdphXBUXA  
/ip4/172.19.0.3/tcp/4001/ipfs/QmassGgK5Rh2Hw9rW6a1HN5eTU71aSp9UYeLbAGbAgEnNw  
root@my_ipfs3:/ipfs#  
  
root@my_ipfs2: /ipfs  
root@my_ipfs2:/ipfs# ipfs id -f=<id>\n"  
QmassGgK5Rh2Hw9rW6a1HN5eTU71aSp9UYeLbAGbAgEnNw  
root@my_ipfs2:/ipfs# ipfs swarm peers  
/ip4/172.18.0.1/tcp/4001/ipfs/QmWB7iunGuHBAFzHZqbDpssdAifMRFtn7tNHDQb1h1DojQ  
/ip4/172.19.0.2/tcp/4001/ipfs/QmWDypxupuDUdczDW4MBphoyZ499vmdRTBF4REdphXBUXA  
/ip4/172.19.0.4/tcp/4001/ipfs/QmcAtiPs5Qp6Q8PHCcwtx8zHYuuwFRWen57WR6dJK2xNnF  
root@my_ipfs2:/ipfs#  
  
root@my_ipfs1: /ipfs  
root@my_ipfs1:/ipfs# ipfs id -f=<id>\n"  
QmWDypxupuDUdczDW4MBphoyZ499vmdRTBF4REdphXBUXA  
root@my_ipfs1:/ipfs# ipfs swarm peers  
/ip4/172.18.0.1/tcp/4001/ipfs/QmWB7iunGuHBAFzHZqbDpssdAifMRFtn7tNHDQb1h1DojQ  
/ip4/172.19.0.3/tcp/4001/ipfs/QmassGgK5Rh2Hw9rW6a1HN5eTU71aSp9UYeLbAGbAgEnNw  
/ip4/172.19.0.4/tcp/4001/ipfs/QmcAtiPs5Qp6Q8PHCcwtx8zHYuuwFRWen57WR6dJK2xNnF  
root@my_ipfs1:/ipfs#
```

Figura B.16: ipfs & docker (Go CLI)

Pe un caz mai individual de experimentare, în (Figura B.16) sunt prezentate 3 noduri IPFS ce rulează individual în câte un container Docker²³. Acestea sunt interconectate de o rețea Docker de tip “internal” (fără acces din / spre exterior). Prin mecanismele de rutare și

²³ "Docker." <https://www.docker.com/>.

descoperire, odată instanțiate, cele 3 noduri reușesc să își realizeze conexiuni reciproce în mod automat, accentul căzând pe lipsa de conexiune cu rețeaua globală de internet (observabilă în listarea nodurilor cu care fiecare are o conexiune activă)

1.5.3. Routing

În IPFS există mai multe mecanisme ce realizează această caracteristică. Arhitectural vorbind, protocolul permite implementarea mai multor astfel de mecanisme pentru permiterea găsirii altor noduri și a diverselor obiecte.

Sunt folosite tehnologii precum S/Kademlia și Coral (descrise în subcapitolele anterioare). Tabelele hash distribuite din IPFS disting valorile stocate în funcție de dimensiunea lor, cele mai mici sau egale cu 1KB fiind stocate directe în tabele iar cele mai mari fiind reținute ca și referințe (NodeId-urile ce le pot oferi) [1]. O interfață simplificată a acestor tabele poate fi analizată în (Figura B.17).

```
type IPFSRouting interface {
    FindPeer(node NodeId)
    // gets a particular peer's network address
    SetValue(key []bytes, value []bytes)
    // stores a small metadata value in DHT
    GetValue(key []bytes)
    // retrieves small metadata value from DHT
    ProvideValue(key Multihash)
    // announces this node can serve a large value
    FindValuePeers(key Multihash, min int)
    // gets a number of peers serving a large value
}
```

Figura B.17: interfața DHT

Fiecare nod reține informații folosind DHT-ul Kademlia în *k-buckets*, informații ce conțin localizări ale altor noduri din rețea și oferind acestora posibilitatea de regăsire a identităților sau obiectelor dorite [3].

Descoperirea automată, realizată după conectarea la rețeaua IPFS, se face prin mai multe procedee:

O tehnologie importantă folosită de către IPFS este **mDNS**²⁴ (multicast DNS) peste rețelele locale (LAN). Aceasta emite în subrețele “semnale” pentru descoperirea altor peers. Avantajele folosirii acesteia sunt latența mică, descoperirea nodurilor din “vecinătatea locală”, a celor deconectate de la rețeaua de Internet globală dar și rețele ce își pierd temporar conexiunea. Ca un dezavantaj al mDNS-ului ar fi “natura” sa de a face vizibile adresele IP locale [3].

²⁴ "Multicast DNS." <http://www.multicastdns.org/>.

Procedeu de “mers aleator” (**Random-Walk**) este un protocol de descoperire pentru tabelele hash distribuite. Acest procedeu acționează interogând aleator DHT-urile din rețea, descoperind noduri și oferind posibilitatea tabelului de la nodul respectiv sa convergă rapid.

Un alt principiu utilizat extensiv în IPFS este acela de **Bootstrap-List**. Acesta presupune folosirea unei liste de noduri “de încredere” salvată în cache-ul local pentru găsirea practică a restului rețelei odată cu inițializarea nodului. IPFS are o astfel de listă salvată *hardcoded* în fișierul principal de configurare, lăsând posibilitatea de modificare a ei [3]. (Figura B.18)

```
xsky@inf ~ $ ipfs bootstrap --help
USAGE
  ipfs bootstrap - Show or edit the list of bootstrap peers.

SYNOPSIS
  ipfs bootstrap

DESCRIPTION

  Running 'ipfs bootstrap' with no arguments will run 'ipfs bootstrap list'.

  SECURITY WARNING:

  The bootstrap command manipulates the "bootstrap list", which contains
  the addresses of bootstrap nodes. These are the *trusted peers* from
  which to learn about other peers in the network. Only edit this list
  if you understand the risks of adding or removing nodes from this list.

SUBCOMMANDS
  ipfs bootstrap add [<peer>]... - Add peers to the bootstrap list.
  ipfs bootstrap list              - Show peers in the bootstrap list.
  ipfs bootstrap rm [<peer>]...   - Remove peers from the bootstrap list.

  Use 'ipfs bootstrap <subcmd> --help' for more information about each command.

xsky@inf ~ $ ipfs bootstrap
/ip4/104.131.131.82/tcp/4001/ipfs/QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbUufsmvsgQLuuvuJ
/ip4/104.236.151.122/tcp/4001/ipfs/QmSoLju6m7xTh3DuokvT3886QRYqxRzb1kShaanJgW36yx
/ip4/104.236.176.52/tcp/4001/ipfs/QmSoLnS6ccFuZQJzRadHn95W2CrSFmZuTdDWP8HXaHca9z
/ip4/104.236.179.241/tcp/4001/ipfs/QmSoLppuBtQSGwKdZT2M73ULp.jvfd3aZ6ha4oFGL1Kr6M
/ip4/104.236.76.40/tcp/4001/ipfs/QmSoLv4Bbm51jM9C4gDYZQ9Cy3U6aXMDAbzgu2fzaDs64
/ip4/128.199.219.111/tcp/4001/ipfs/QmSoLSafTMBsPKadTEgaXctDQVcqN88CNLHXMKTNuMKPnu
/ip4/162.243.248.213/tcp/4001/ipfs/QmSoLueR4xBelUy9WZ9xGUUxunbKlcrNFTDAadQJmocrNlm
/ip4/178.62.158.247/tcp/4001/ipfs/QmSoLer265NRgSp2LA3dPaeykiS1J6DiFTC88f5uVQKNAd
/ip4/178.62.61.185/tcp/4001/ipfs/QmSoLMwqB7YGVLJN3pNLQpmMEk35v6wYtsMGLzSr5QBUI3
/ip6/2400:6180:0:d0::151:6001/tcp/4001/ipfs/QmSoLSafTMBsPKadTEgaXctDQVcqN88CNLHXMKTNuMKPnu
/ip6/2604:a880:0:1010::23:d001/tcp/4001/ipfs/QmSoLueR4xBelUy9WZ9xGUUxunbKlcrNFTDAadQJmocrNlm
/ip6/2604:a880:1:20::1d9:6001/tcp/4001/ipfs/QmSoLju6m7xTh3DuokvT3886QRYqxRzb1kShaanJgW36yx
/ip6/2604:a880:1:20::1f9:9001/tcp/4001/ipfs/QmSoLnS6ccFuZQJzRadHn95W2CrSFmZuTdDWP8HXaHca9z
/ip6/2604:a880:1:20::203:d001/tcp/4001/ipfs/QmSoLppuBtQSGwKdZT2M73ULp.jvfd3aZ6ha4oFGL1Kr6M
/ip6/2604:a880:800:10::4a:5001/tcp/4001/ipfs/QmSoLv4Bbm51jM9C4gDYZQ9Cy3U6aXMDAbzgu2fzaDs64
/ip6/2a03:b0c0:0:1010::23:1001/tcp/4001/ipfs/QmSoLer265NRgSp2LA3dPaeykiS1J6DiFTC88f5uVQKNAd
/ip6/2a03:b0c0:1:d0::e7:1/tcp/4001/ipfs/QmSoLMwqB7YGVLJN3pNLQpmMEk35v6wYtsMGLzSr5QBUI3
xsky@inf ~ $
```

Figura B.18: comanda `ipfs bootstrap` pentru administrarea listei (Go CLI)

```
xsky@inf ~ $ ipfs dht --help
USAGE
  ipfs dht - Issue commands directly through the DHT.

SYNOPSIS
  ipfs dht

SUBCOMMANDS
  ipfs dht findpeer <peerID>... - Query the DHT for all of the multiaddresses associated with a Peer ID.
  ipfs dht findprovs <key>...   - Find peers in the DHT that can provide a specific value, given a key.
  ipfs dht get <key>...         - Given a key, query the DHT for its best value.
  ipfs dht provide <key>...     - Announce to the network that you are providing given values.
  ipfs dht put <key> <value>   - Write a key/value pair to the DHT.
  ipfs dht query <peerID>...   - Find the closest Peer IDs to a given Peer ID by querying the DHT.

  Use 'ipfs dht <subcmd> --help' for more information about each command.
```

Figura B.19: comanda `ipfs dht <subcmd>` (Go CLI)

În (Figura B.19) este prezentată de către CLI comanda `ipfs dht`. Prin utilizarea acesteia se poate găsi un anumit nod, se pot găsi toate nodurile ce pot oferi un anumit obiect identificat de `key`, se poate extrage și introduce valoarea unei chei din / în DHT dar și anunțarea rețelei că nodul curent poate oferi o valoare pentru o anumită cheie. Un exemplu de utilizare poate fi identificat în (Figura B.20) unde s-a folosit această comandă pentru identificarea nodurilor ce pot oferi un obiect cu un anumit `key` (hash).

```
xskj@inf ~ $ ipfs dht findprovs QmYwAPJzYv5CZsnA625s3Xf2nemtYgPpHdWEz79oJWnPbdG
QmUexn3jWugGbrX1AkCLJysCdVzsrCiEncHJue1Pnmwh5g
QmWB7iunGuHBAFzHZqbDpssdAi fMRf tN7tNHDQb1h1Do jQ
QmUpnY2gAsWFPYeZoAi fhhhvRA77FNsmaivQKYA jGeEYtM
QmW8xgbEvoHaZykCoXpQCGk7KneyF6QPrv9fg8FbMGXkv4
QmbCSU1i jfcK6H3 jtN6 fFQ8NMRz6wN39fNEqoWKc9QNC5a
QmR1mXyic9 jSbyzL tnBU9g jbfY8K3TFHrpvJK88LSyPnd9
QmPKes5JarRdBkvuYi1aDMtWxptYrSK8VsVexRsyMjLQNT
QmNHxr1ZoyPbwNe2CvYz1CvYvSNwsE8WwDWQ9t9BD jn j5
QmNYWAOc1QC1yc4E9e8ebTs6X7EzY6Tgz fu9bNGqRodG jc
QmNdvrwxuqVEBPJ34e6uRka5KQHZeFz5TrMgevPgsgs2nNG
QmNFEDJmukitBukC8Wp5L37emgo13deyJ4EdqXSFzmyDpN
QmNmraVaJh66sh6wLhAEwTdLSxbs ji taAHncadJ7RPPyPN
QmNMBRw4Ap2tmzihsQPEMaCdKg1scztpNaW9bHJF2XamU
QmP13QaQFpB jgs22DpEW7SeAVZMuSYx9XmPqFMnDxoqYgS
QmP2aknMA7RwL7KXyQMvVyipbhdEgxe7LiBTffLbtTSR
QmP8z jPkdPYn5o j3CUuPR jRyRynYt8DSSElb jt3HqusZHQ
QmQLW2mhJYPmhYmhkA2F2wFGdEXF jnsprB5dfBxCMRdBk9
QmQM1vxVgRwy9fbbckXZY3uGM97FzWzu9VkvUuQeu4d6iE
QmQM8gKpyHAAqVp6W9x3hxTviHUUQ8mVhJvFDPHhrKwVdYC
QmPRRCBGtB94Eu4hSeSNLk8BZ1dYbSu6ZaZxGGnSpvuiAn
```

Figura B.20: utilizarea comenzii `ipfs dht findprovs` (Go CLI)

Varianta de implementare în Go a protocolului dispune de funcționalitățile DHT-ului, a mDNS-ului, fiind procedeele de bază de descoperire a nodurilor în rețea (Anexa 2). Comparativ, rutarea folosind tabelele de hash dar și tehnologia “relay” ce vor permite nodurilor din browser să se conecteze la orice alt nod din rețea fără configurații manuale încă se află sub dezvoltare pentru varianta Javascript care în momentul de față utilizează “Bootstrap-List” și un server de semnalizare WebRTC-Star pentru realizarea de conexiuni (Figura B.21)²⁵ cu alte noduri ale implementării Javascript. Aceasta se folosește de WebSockets²⁶ pentru acceptarea de conexiuni.

```
"Addresses": {
  "Swarm": [
    "/ip4/0.0.0.0/tcp/4002",
    "/ip4/127.0.0.1/tcp/9999/ws"
  ],
  "API": "/ip4/127.0.0.1/tcp/5002",
  "Gateway": "/ip4/127.0.0.1/tcp/9090"
}
```

Figura B.21: configurație pentru folosirea serverului de semnalizare (Javascript)

²⁵ "js-ipfs/examples/transfer-files at master · ipfs/js-ipfs · GitHub."

<https://github.com/ipfs/js-ipfs/tree/master/examples/transfer-files>.

²⁶ "WebSockets - Web APIs | MDN."

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.

Un alt principiu vital pe care IPFS îl folosește (implementat experimental momentan) pentru trimiterea în timp real a mesajelor în rețea este conceptul **Publish-Subscribe**²⁷ (pubsub). Acesta presupune existența unor “publishers” (cei ce trimit mesaje) și a unor “subscribers” (cei ce primesc mesaje). Trimiterea efectivă de mesaje nu se rezumă la “direcționarea” lor către anumite noduri ci folosirea unui sistem de subiecte (“topics”) ce acționează precum niște canale de comunicare la care nodurile se abonează (“subscribe”) și pot primi mesaje publicate (“publish”) pe astfel de subiecte.

```
xsky@inf Downloads $ ipfs pubsub --help
USAGE
  ipfs pubsub - An experimental publish-subscribe system on ipfs.

SYNOPSIS
  ipfs pubsub

DESCRIPTION

  ipfs pubsub allows you to publish messages to a given topic, and also to
  subscribe to new messages on a given topic.

  This is an experimental feature. It is not intended in its current state
  to be used in a production environment.

  To use, the daemon must be run with '--enable-pubsub-experiment'.

SUBCOMMANDS
  ipfs pubsub ls                - List subscribed topics by name.
  ipfs pubsub peers [<topic>]  - List peers we are currently pubsubbing with.
  ipfs pubsub pub <topic> <data>... - Publish a message to a given pubsub topic.
  ipfs pubsub sub <topic>      - Subscribe to messages on a given topic.

  Use 'ipfs pubsub <subcmd> --help' for more information about each command.
```

Figura B.22: comanda generala pubsub (Go CLI)



Figura B.23: exemplificare a folosirii conceptului pubsub (Go CLI)

În (Figura B.22) sunt prezentate funcționalitățile expuse la CLI de către comanda `ipfs pubsub <subcmd>` în varianta Go. (Figura B.23) exemplifică utilizarea acesteia prin existența a 3 entități: prima care face “subscribe” la “mychannel”, a doua care face “subscribe” la “myotherchannel” și o a treia care face “publish” unui mesaj către primul canal. Se poate observa cum doar prima entitate abonată la primul canal va primi mesajul.

²⁷ "Publish/Subscribe - MSDN - Microsoft." <https://msdn.microsoft.com/en-us/library/ff649664.aspx>

Observație: fiind un concept experimental, această funcționalitate presupune inițializarea daemon-ului IPFS cu parametrul `--enable-pubsub-experiment`. Deși testele au fost făcute pe aceeași mașină, “pubsub” va funcționa și între noduri ce rulează pe mașini diferite cât timp va exista o conexiune activă între nodurile ce fac “subscribe” la același canal.

```
const topic = 'mychannel'
const receiveMsg = (msg) => {
  console.log(msg.toString())
}

ipfs.pubsub.subscribe(topic, receiveMsg)

const msg = new Buffer('Hello from IPFS Pubsub')
ipfs.pubsub.publish(topic, msg, (err) => {
  if (err) {
    throw err
  }
  // msg was broadcasted
})
```

Figura B.24: exemplificarea utilizării pubsub (Javascript)

În (Figura B.24)[5] este exemplificată utilizarea conceptului în Javascript prin abonarea la un “topic”, declararea unei funcții ce urmează a fi apelată la primire de mesaje și trimiterea efectivă a mesajului.

1.5.4. Block Exchange

Schimbul de blocuri de date în IPFS este cel ce are grijă de transferul dintre noduri a informațiilor dorite sau oferite. Aceste date sunt salvate în repo-ul local al fiecărui nod sub forma unor blocuri adresate de propriul conținut. **BitSwap** este o generalizare a BitTorrent-ului folosită ca principal protocol de interschimbare a datelor, **HTTP** fiind o alternativă acestuia. Fiecare nod dispune de o lista de blocuri dorite (“wantlist”) și de blocuri oferite (“havelist”) rețelei precum în protocolul BitTorrent, dar elementele acestora nu sunt limitate la un singur “torrent” și nici nu se ține cont de fișierele de proveniență ori de sistemele de fișiere în cadrul cărora sunt salvate[1].

Ca și principal protocol, **BitSwap** se ocupă de administrarea cererilor de transfer de blocuri de date de la / către alte noduri din rețea. La bază acesta este un protocol orientat pe mesaje, acestea conținând liste “wantlist” sau blocuri. La primirea unui “wantlist” un nod va trimite blocurile specificate către cel care realizează cererea dacă le deține. Odată primite blocurile, nodurile trimit o notificare prin care semnalează faptul ca nu mai doresc respectivele blocuri, ștergându-le din “wantlist” și adăugându-le eventual în “havelist”[4].

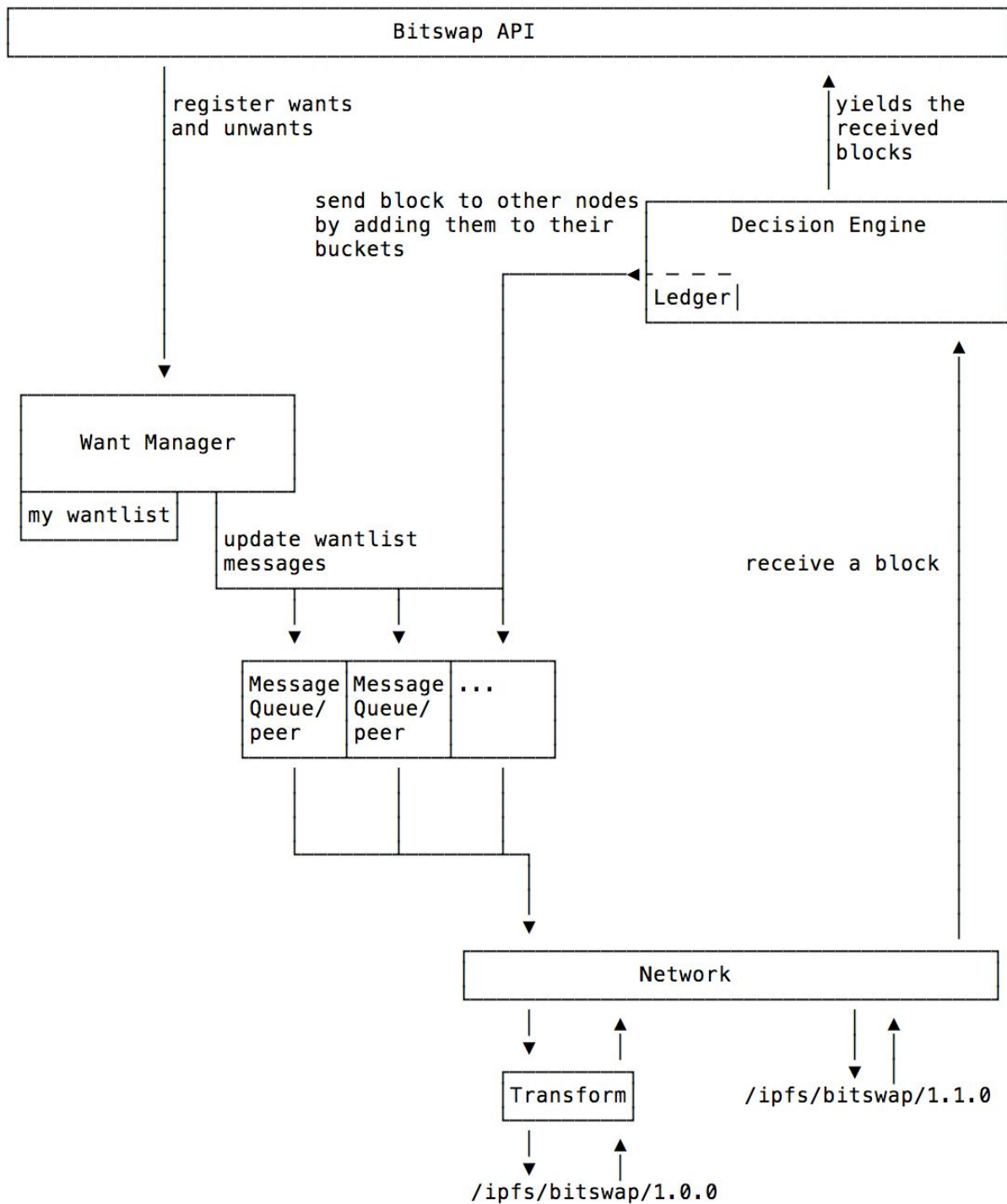


Figura B.25²⁸: structura protocolului BitSwap exemplificată pe implementarea Javascript

²⁸ "GitHub - ipfs/js-ipfs-bitswap: JavaScript implementation of Bitswap"
<https://github.com/ipfs/js-ipfs-bitswap>.

1.5.5. Merkle DAG & Files

Merkle Directed Acyclic Graph (Merkle DAG) reprezintă modelul principal de stocare a datelor în IPFS, model generalizat din structurile de date Git. În cadrul acestuia nodurile grafului reprezintă obiectele propriu-zise adresate prin hash-ul criptografic al propriului conținut iar legătura dintre ele fiind chiar referințele (adresările hash) din interiorul unui nod.

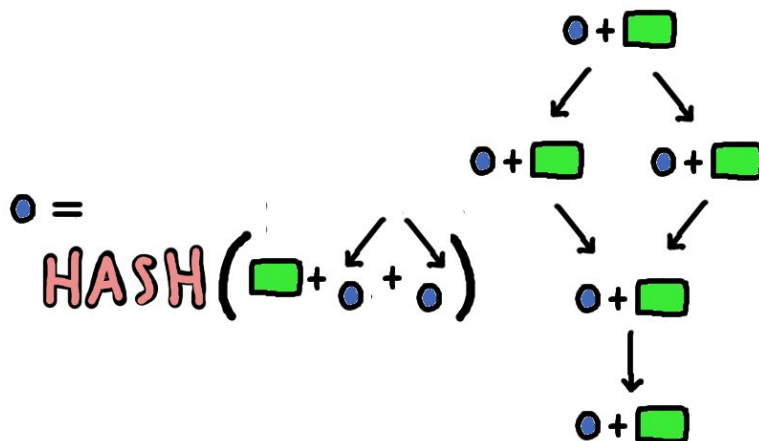


Figura B.26: Merkle DAG simplificat²⁹

Printre proprietățile acestui model se numără [1]:

- **adresarea pe bază de conținut:** toate obiectele sunt identificate unic de către propriul multihash împreună cu referințele
- **integritate:** dacă datele sunt corupte sau modificate, sistemul detectează aceste cazuri folosind sumele de control
- **deduplicare:** obiectele cu aceeași referință sunt salvate o singură dată

```
type IPFSLink struct {
    Name string // name or alias of this link
    Hash Multihash // cryptographic hash of target
    Size int // total size of target
}

type IPFSObject struct {
    links []IPFSLink // array of links
    data []byte // opaque content data
}
```

Figura B.27: formatul unui obiect IPFS

²⁹ "DTN-2015 slides." <https://mafintosh.github.io/slides/dtn-2015/merkle-dag-example.png>

În (Figura B.27)[1] este prezentat formatul unui “link” și a unui “object” IPFS: numele, multihash-ul, dimensiunea (o referință - link), lista de referințe și datele propriu-zise (un obiect - object). Acest format poate fi folosit în multe cazuri de modelare a datelor, de la stocări (key, values) la baze de date tradiționale și altele.

Paths (căile) în acest model de date sunt de forma:

```
/ipfs/<hash-of-object>/<name-path-to-object>
```

unde `/ipfs` este un prefix general utilizat pentru obiectele din ecosistem, a doua cale reprezintă hash-ul obiectului rădăcină iar ultima cale reprezintă numele entității din obiectul rădăcină. Există posibilitatea de accesare a ultimei și prin propriul ei hash, lipsind astfel numele obiectului la care referim [1].

Local Objects (obiectele locale) se află în sistemul de fișiere local necesar stocării repo-ului fiecărui nod. Mediul de stocare poate diferi de la un caz la altul, în majoritatea cazurilor fiind reprezentat de o porțiune de spațiu de pe disk, fie că vorbim de sistemul de fișiere nativ, un mediu de tipul (key, value) ori o zonă volatilă de persistență (ex. RAM) [1]. Atunci când un nod cere un bloc de informații rețelei, acesta este găsit, descărcat și salvat în stocarea locală. Fișierele de dimensiuni mari sunt “sparte” în blocuri de date de 256 KB ce referențiază la părintele din care fac parte.

Object Pinning (fixarea obiectelor) este un procedeu ce asigură longevitatea obiectelor aflate sunt acest concept, asigurând persistența lor în mediul local de stocare. Fixarea asigură caracteristica de “permanență” a datelor.

Publishing Objects (publicarea obiectelor) reprezintă acțiunea de adăugare a referințelor obiectelor în tabelele hash distribuite de către noduri cu propria identitate ca fiind o sursă de oferire a obiectelor respective. Obiectele sunt “imutabile”, de fiecare dată când se modifică unul realizându-se crearea unor noi referințe.

Object-level Cryptography (criptografia la nivel de obiecte) este asigurată prin diverse mecanisme de verificare și criptare prin “învăluirea” obiectelor într-un cadru criptografic. (Figura B.28) [1]

```
type EncryptedObject struct {
    Object []bytes // raw object data encrypted
    Tag []bytes // optional tag for encryption groups
}
type SignedObject struct {
    Object []bytes // raw object data signed
    Signature []bytes // hmac signature
    PublicKey []multihash // multihash identifying key
}
```

Figura B.28: cadru criptografic destinat obiectelor

În practică (Anexa 2) interfața CLI a variantei Go dar și implementarea Javascript oferă diverse funcționalități pentru adăugare, ștergere, fixare și oferire de obiecte împreună cu multe altele pentru interacțiunea cu arborele Merkle.

```
xsky@inf test_ipfs $ ipfs add --help
USAGE
  ipfs add <path>... - Add a file or directory to ipfs.

SYNOPSIS
  ipfs add [--recursive | -r] [--quiet | -q] [--quieter | -Q] [--silent] [--progress | -p] [--trickle | -t] [--only-hash | -n] [--wrap-with-directory | -w] [--hidden | -H] [--chunker=<chunker> | -s] [--pin=false] [--raw-leaves] [--nocopy] [--fscache] [--] <path>...

ARGUMENTS

  <path>... - The path to a file to be added to ipfs.

OPTIONS

  -r,      --recursive      bool    - Add directory paths recursively. Default: false.
  -q,      --quiet          bool    - Write minimal output.
  -Q,      --quieter        bool    - Write only final hash.
  --silent      bool    - Write no output.
  -p,      --progress       bool    - Stream progress data.
  -t,      --trickle        bool    - Use trickle-dag format for dag generation.
  -n,      --only-hash      bool    - Only chunk and hash - do not write to disk.
  -w,      --wrap-with-directory bool  - Wrap files with a directory object.
  -H,      --hidden         bool    - Include files that are hidden. Only takes effect on recursive add.
  -s,      --chunker        string   - Chunking algorithm to use.
  --pin        bool    - Pin this object when adding. Default: true.
  --raw-leaves bool    - Use raw blocks for leaf nodes. (experimental).
  --nocopy     bool    - Add the file using filestore. (experimental).
  --fscache    bool    - Check the filestore for pre-existing blocks. (experimental).

DESCRIPTION

  Adds contents of <path> to ipfs. Use -r to add directories.
  Note that directories are added recursively, to form the ipfs
  MerkleDAG.

  The wrap option, '-w', wraps the file (or files, if using the
  recursive option) in a directory. This directory contains only
  the files which have been added, and means that the file retains
  its filename. For example:

  > ipfs add example.jpg
  added QmbFMke1KXqnYgBBWxB74N4c5SBnJMVaiMNRcGu6x1AwQH example.jpg
  > ipfs add example.jpg -w
  added QmbFMke1KXqnYgBBWxB74N4c5SBnJMVaiMNRcGu6x1AwQH example.jpg
  added QmaG4FuMqEBnQnN3C8XJ5bpW8kLs7zq2ZxgHptJHbKDDVx

  You can now refer to the added file in a gateway, like so:

  /ipfs/QmaG4FuMqEBnQnN3C8XJ5bpW8kLs7zq2ZxgHptJHbKDDVx/example.jpg
```

Figura B.28: comanda de adăugare (Go CLI)

```
xsky@inf test_ipfs $ tree my_data
my_data
├── test1_dir
│   └── test2.txt
└── test1.txt

1 directory, 2 files
xsky@inf test_ipfs $ ipfs add -r my_data
added Qm2mVZyYZNXZU5WLf4K29SdGeonauK1PQXHXphs9rzNjZ6 my_data/test1.txt
added QmcrNnf4yULS8ET2pogs6xjeuBxM5qZzxbtap8KyyghvUm my_data/test1_dir/test2.txt
added Qmbzpp7fBREH4bw2mws5UaL3UWMtthSRsTbEdzYvf68H8n my_data/test1_dir
added QmcZ115VAhyee7emaYPLHRAZcmreVi3KjWjT7xZQMqEC3ja my_data
xsky@inf test_ipfs $
```

Figura B.29: adăugare recursivă a unui folder (Go CLI)


```

ipfs.object.new([template][, callback])
ipfs.object.put(obj, [options, callback])
ipfs.object.get(multihash, [options, callback])
ipfs.object.data(multihash, [options, callback])
ipfs.object.links(multihash, [options, callback])
ipfs.object.stat(multihash, [options, callback])
ipfs.object.patch.addLink(multihash, DAGLink, [options, callback])
ipfs.object.patch.rmLink(multihash, DAGLink, [options, callback])
ipfs.object.patch.appendData(multihash, data, [options, callback])
ipfs.object.patch.setData(multihash, data, [options, callback])

```

Figura B.30: funcționalități pentru lucrul cu obiecte din interfața principală (Javascript)

Adăugarea entităților de date în IPFS se poate realiza cu ajutorul comenzii `ipfs add` (Figura B.28) (Figura B.29) și (Figura B.30) pentru varianta Javascript [5]. Se pot observa structura directorului dat ca exemplu și referințele fiecărui director și fișier.

În (Figura B.31) este prezentat un exemplu de listare a directorului adăugat din (Figura B.29) în cadrul IPFS-ului dar și afișarea conținutului fișierelor din interiorul acestuia. În cazul afișării de conținut poate observa folosirea diferitelor moduri de referențiere ale aceluiași fișier `test2.txt` în funcție de hash-ul direct, relativ la primul director ascendent `test1_dir` sau la directorul rădăcină `my_data`.

```

xsky@inf test_ipfs $ ipfs ls -v QmcZ115VAhyee7emaYPLHAZcmreVi3KjWJT7xZQMqEC3ja
Hash                               Size Name
QmZmVZyYZNXZU5WLf4K29SdGeonauK1PQXHXphs9rzrNJZ6 28  test1.txt
Qmbzpp7fBREH4bwZmws5UaL3UWMtthSRsTbEdzYvf68H8n 83  test1_dir/
xsky@inf test_ipfs $ ipfs ls -v Qmbzpp7fBREH4bwZmws5UaL3UWMtthSRsTbEdzYvf68H8n
Hash                               Size Name
QmcrNnf4yULS8ET2pogs6xjeuBxM5qZzxzbtap8KyyhvUm 28  test2.txt
xsky@inf test_ipfs $ ipfs cat QmZmVZyYZNXZU5WLf4K29SdGeonauK1PQXHXphs9rzrNJZ6
Hello 1 from IPFS !
xsky@inf test_ipfs $ ipfs cat QmcrNnf4yULS8ET2pogs6xjeuBxM5qZzxzbtap8KyyhvUm
Hello 2 from IPFS !
xsky@inf test_ipfs $ ipfs cat Qmbzpp7fBREH4bwZmws5UaL3UWMtthSRsTbEdzYvf68H8n/test2.txt
Hello 2 from IPFS !
xsky@inf test_ipfs $ ipfs cat QmcZ115VAhyee7emaYPLHAZcmreVi3KjWJT7xZQMqEC3ja/test1_dir/test2.txt
Hello 2 from IPFS !
xsky@inf test_ipfs $ █

```

Figura B.31: listarea directoarelor și afișarea conținutului fișierelor în diverse moduri (Go CLI)

În (Figura B.32) (Figura B.33) și (Figura B.34) sunt afișate la terminal manualele comenzilor `ipfs block <subcmd>`, `ipfs object <subcmd>` și `ipfs pin <subcmd>`. Acestea pot interacționa direct cu blocurile de date și obiectele sau pot fixa anumite date pe care utilizatorul le dorește permanente (merită precizat că la adăugare se face fixare implicită).

```
xsky@inf test_ipfs $ ipfs block
USAGE
  ipfs block - Interact with raw IPFS blocks.

  ipfs block

'ipfs block' is a plumbing command used to manipulate raw IPFS blocks.
Reads from stdin or writes to stdout, and <key> is a base58 encoded
multihash.

SUBCOMMANDS
  ipfs block get <key>      - Get a raw IPFS block.
  ipfs block put <data>     - Store input as an IPFS block.
  ipfs block rm <hash>...   - Remove IPFS block(s).
  ipfs block stat <key>    - Print information of a raw IPFS block.

Use 'ipfs block --help' for more information about this command.
```

Figura B.32: comanda de interacțiune cu blocuri de date (Go CLI)

```
xsky@inf test_ipfs $ ipfs object
USAGE
  ipfs object - Interact with IPFS objects.

  ipfs object

'ipfs object' is a plumbing command used to manipulate DAG objects
directly.

SUBCOMMANDS
  ipfs object data <key>      - Output the raw bytes of an IPFS object.
  ipfs object diff <obj_a> <obj_b> - Display the diff between two ipfs objects.
  ipfs object get <key>      - Get and serialize the DAG node named by <key>.
  ipfs object links <key>    - Output the links pointed to by the specified object.
  ipfs object new [<template>] - Create a new object from an ipfs template.
  ipfs object patch          - Create a new merkledag object based on an existing one.
  ipfs object put <data>     - Store input as a DAG object, print its key.
  ipfs object stat <key>    - Get stats for the DAG node named by <key>.

Use 'ipfs object --help' for more information about this command.
```

Figura B.33: comanda de interacțiune cu obiecte (Go CLI)

```
xsky@inf test_ipfs $ ipfs pin
USAGE
  ipfs pin - Pin (and unpin) objects to local storage.

  ipfs pin

SUBCOMMANDS
  ipfs pin add <ipfs-path>... - Pin objects to local storage.
  ipfs pin ls [<ipfs-path>]... - List objects pinned to local storage.
  ipfs pin rm <ipfs-path>...  - Remove pinned objects from local storage.

Use 'ipfs pin --help' for more information about this command.
```

Figura B.34: comanda de fixare a datelor (Go CLI)

Files (fișierele) reprezintă conceptul de lucru în cadrul IPFS-ului cu entități de date asemănător cu cele din cadrul unui sistem de fișiere Unix. API-ul introduce ideea de director rădăcină virtual unde fișierele și alte directoare pot fi manipulate după bunul plac de către utilizator. Aceste entități sunt foarte asemănătoare obiectelor Git prezentate până acum și la nivel abstract sunt reprezentate de: `block` (bloc de date de dimensiune variabilă), `list`

(colecție de blocuri sau alte liste), `tree` (colecție de blocuri, liste sau alți arbori), `commit` (o anumită versiune din cadrul istoriei arborilor)³⁰ [1].

```
xsky@inf test_ipfs $ ipfs files --help
USAGE
  ipfs files - Interact with unixfs files.

SYNOPSIS
  ipfs files [--flush=false]

OPTIONS
  -f, --flush bool - Flush target and ancestors after write. Default: true.

DESCRIPTION

Files is an API for manipulating IPFS objects as if they were a unix
filesystem.

NOTE:
Most of the subcommands of 'ipfs files' accept the '--flush' flag. It defaults
to true. Use caution when setting this flag to false. It will improve
performance for large numbers of file operations, but it does so at the cost
of consistency guarantees. If the daemon is unexpectedly killed before running
'ipfs files flush' on the files in question, then data may be lost. This also
applies to running 'ipfs repo gc' concurrently with '--flush=false'
operations.

SUBCOMMANDS
ipfs files cp <source> <dest> - Copy files into mfs.
ipfs files flush [<path>] - Flush a given path's data to disk.
ipfs files ls [<path>] - List directories in the local mutable namespace.
ipfs files mkdir <path> - Make directories.
ipfs files mv <source> <dest> - Move files.
ipfs files read <path> - Read a file in a given mfs.
ipfs files rm <path>... - Remove a file.
ipfs files stat <path> - Display file status.
ipfs files write <path> <data> - Write to a mutable file in a given filesystem.

Use 'ipfs files <subcmd> --help' for more information about each command.
```

Figura B.34: comanda `ipfs files` (Go CLI)

În (Figura B.34) este prezentată comanda `ipfs files <subcmd>` ce permite lucrul cu entitățile de date în stilul “unixfs”. (Figura B.36) oferă un exemplu simplu de utilizare a acestuia prin copierea directorului utilizat și mai sus din repo-ul local în sistemul virtual de directoare.

```
xsky@inf test_ipfs $ ipfs files cp /ipfs/QmcZ115VAhyee7emaYPLHAZcmreVi3KjWJT7xZQMqEC3ja /my_data
xsky@inf test_ipfs $ ipfs files ls -l /
my_data QmcZ115VAhyee7emaYPLHAZcmreVi3KjWJT7xZQMqEC3ja 0
xsky@inf test_ipfs $ ipfs files ls -l /my_data
test1.txt      QmZmVZyYZNXZU5WLf4K29SdGeonauK1PQXHXpHs9rzNJZ6 20
test1_dir     Qmbzpp7fBREH4bwZmws5UaL3UWMtthSRsTbEdzYvf68H8n 0
```

Figura B.36: exemplu de utilizare a comenzii `ipfs files` (Go CLI)

³⁰ "How does Files API - 'ipfs files' command work. · Issue #143 · ipfs/faq" Accessed June 18, 2017. <https://github.com/ipfs/faq/issues/143>.

1.5.6. IPNS (Naming and Mutable State)

IPNS (**InterPlanetary Name Space**) este o strategie folosită de IPFS în vederea identificării obiectelor asupra cărora pot surveni schimbări. Problema de bază este faptul ca obiectele sunt **imutabile**, fiind create noi entități la orice schimbare. IPNS oferă un mod de adresare ce rămâne neschimbat pentru obținerea de conținut. Acest lucru este realizat prin “**mutable pointers**” către arborele Merkle. Conceptul se folosește de schema de nume SFS descrisă în subcapitolele anterioare pentru construcția de nume auto-certificate într-un spațiu de nume global asigurat criptografic.

```
NodeId = hash(node.PubKey)
        /ipns/<NodeId>
routing.setValue(NodeId, <ns-object-hash>)
```

Schema de mai sus [1] este o simplificare a conceptului IPNS: este folosită identitatea fiecărui nod ce dorește să publice o referință în IPNS împreună cu prefixul /ipns ce stabilește o diferențiere de căile mutabile și imutabile /ipfs. Procesul de publicare este unul în doi pași: adăugarea datelor dorite în sistem precum în (Figura B.29) și apoi publicarea valorii rezultate a căii rădăcină în IPNS. Astfel, orice schimbare intervine asupra obiectelor respective, identificatorul lor se va schimba dar cel de identificare a nodului nu, lăsând posibilitatea de republicare a noii valori și menținerea referințelor după caz (Anexa 2).

```
xsky@inf test_ipfs $ ipfs name
USAGE
  ipfs name - Publish and resolve IPNS names.

  ipfs name

  IPNS is a PKI namespace, where names are the hashes of public keys, and
  the private key enables publishing new (signed) values. In both publish
  and resolve, the default name used is the node's own PeerID,
  which is the hash of its public key.

SUBCOMMANDS
  ipfs name publish <ipfs-path> - Publish IPNS names.
  ipfs name resolve [<name>]    - Resolve IPNS names.

Use 'ipfs name --help' for more information about this command.
```

Figura B.37: comanda ipfs name (Go CLI)

În (Figura B.37) este prezentat manualul de utilizare a comenzii `ipfs name <subcmd>` prin care se realizează publicare și rezolvarea referințelor publicate în cadrul spațiului de nume IPNS pentru stări mutabile.

```

xsky@inf test_ipfs $ ipfs add ipns_test.txt
added QmWSdddRBrb4DX8fustqX7M1GxV5Fx48u1ut5w4qvqPQ1X ipns_test.txt
xsky@inf test_ipfs $ ipfs cat QmWSdddRBrb4DX8fustqX7M1GxV5Fx48u1ut5w4qvqPQ1X
Hello from IPFS w/ IPNS !

xsky@inf test_ipfs $ ipfs name publish QmWSdddRBrb4DX8fustqX7M1GxV5Fx48u1ut5w4qvqPQ1X
Published to QmPfbWuqDNQWmJW3TxDtufKmmhwg6z8rChe68S8Kbxq8wN: /ipfs/QmWSdddRBrb4DX8fustqX7M1GxV5Fx48u1ut5w4qvqPQ1X
xsky@inf test_ipfs $ ipfs id -f '<id>\n'
QmPfbWuqDNQWmJW3TxDtufKmmhwg6z8rChe68S8Kbxq8wN
xsky@inf test_ipfs $ ipfs cat /ipns/QmPfbWuqDNQWmJW3TxDtufKmmhwg6z8rChe68S8Kbxq8wN
Hello from IPFS w/ IPNS !

xsky@inf test_ipfs $
xsky@inf test_ipfs $ echo "edit 1" >> ipns_test.txt
xsky@inf test_ipfs $ ipfs cat /ipns/QmPfbWuqDNQWmJW3TxDtufKmmhwg6z8rChe68S8Kbxq8wN
Hello from IPFS w/ IPNS !

xsky@inf test_ipfs $ ipfs cat QmWSdddRBrb4DX8fustqX7M1GxV5Fx48u1ut5w4qvqPQ1X
Hello from IPFS w/ IPNS !

xsky@inf test_ipfs $ ipfs add ipns_test.txt
added QmWEzPwV7Pj3hgJSYR88gX1WUWK33xroeQbQLDxspGmTd ipns_test.txt
xsky@inf test_ipfs $ ipfs name publish /ipfs/QmWEzPwV7Pj3hgJSYR88gX1WUWK33xroeQbQLDxspGmTd
Published to QmPfbWuqDNQWmJW3TxDtufKmmhwg6z8rChe68S8Kbxq8wN: /ipfs/QmWEzPwV7Pj3hgJSYR88gX1WUWK33xroeQbQLDxspGmTd
xsky@inf test_ipfs $ ipfs cat /ipns/QmPfbWuqDNQWmJW3TxDtufKmmhwg6z8rChe68S8Kbxq8wN
Hello from IPFS w/ IPNS !

edit 1
xsky@inf test_ipfs $ ipfs name resolve QmPfbWuqDNQWmJW3TxDtufKmmhwg6z8rChe68S8Kbxq8wN
/ipfs/QmWEzPwV7Pj3hgJSYR88gX1WUWK33xroeQbQLDxspGmTd
xsky@inf test_ipfs $

```

Figura B.38: exemplu de utilizare a comenzii ipfs name (Go CLI)

(Figura B.38) aduce un exemplu relevant utilizării comenzii pentru IPNS. Se poate observa fiecare pas al acestui caz: adăugarea normală a fișierului în arborele Merkle, publicarea cu `ipfs name publish` și afișarea conținutului folosind direct referința rezultată (identificatorul nodului). O schimbare asupra fișierului este produsă după care acesta este readăugat în sistem și republicat la IPNS. Lucrul principal de menționat este faptul că deși fișierul a fost modificat, readăugat și republicat, afișarea conținutului a fost făcută prin **aceeași** referință la IPNS.

Considerând avantajele IPNS-ului, acestuia îi lipsește o caracteristică: ușurința de memorare de către utilizatori a hash-urilor generate. Totuși, prin elasticitatea conceperii sale, IPFS oferă posibilitatea de realizare a unei legături între **înregistrări DNS de tip TXT** și domenii valide de nume de forma `/ipns/<domain>.[1]` Astfel, utilizatorii pot reține domenii “human readable” ce oferă acces la referințe din cadrul IPFS-ului.

2. Javascript IPFS API

Javascript IPFS API [14] (sau pe scurt **js-ipfs-api**) este o librărie client pentru API-ul HTTP expus de către sistemul IPFS. Aceasta implementează interfața principală a IPFS-ului [5] oferind posibilitatea aplicațiilor de instanțiere și conectare la un nod IPFS “la distanță”.

js-ipfs-api este conceput și implementat sub forma unui pachet *npm* [12] și poate fi instalat și folosit în cadrul unei aplicații web ce utilizează Javascript.

```
var ipfsAPI = require('ipfs-api')

// connect to ipfs daemon API server
var ipfs = ipfsAPI('localhost', '5001', {protocol: 'http'})
// leaving out the arguments will default to these values

// or connect with multiaddr
var ipfs = ipfsAPI('/ip4/127.0.0.1/tcp/5001')

// or using options
var ipfs = ipfsAPI({host: 'localhost', port: '5001', protocol: 'http'})
```

Figura B.39: exemplu de instanțiere js-ipfs-api (Javascript)³¹

API-ul necesită configurarea în prealabil a daemon-ului pentru a-i permite conexiunea prin port-ul expus de către instanță. Acesta poate fi utilizat în browser folosind Browserify³², Webpack[13] sau folosind un CDN³³ sursă dar și în cunoscutul mediu de dezvoltare Node.js³⁴.

Disponând de o serie de diverse funcționalități implementate peste interfața principală, API-ul permite utilizarea daemon-ului pentru descoperirea nodurilor, transferul de date sau trimiterea de mesaje prin **PubSub**.

Observație: Aflându-se încă în stadiul de dezvoltare, js-ipfs-api nu dispune de seria totală de funcționalități ale IPFS-ului, unele dintre ele nefuncționând după așteptări. Un exemplu relevant ar fi utilizarea pubsub-ului folosind librăria care în momentul de față are dezactivată această funcționalitate din cauza faptului că închiderea cererilor nu se poate realiza efectiv, acestea fiind doar “ascunse”³⁵. Astfel, o serie de modificări temporare au fost necesare pentru utilizarea pubsub-ului în proiect, lăsând posibilitatea de apariție a viitoarelor rezolvări ale problemei.

³¹ "ipfs/js-ipfs-api - GitHub." <https://github.com/ipfs/js-ipfs-api>

³² "Browserify." <http://browserify.org/>

³³ "Content Delivery Network Explained - GlobalDots." <http://www.globaldots.com/content-delivery-network-explained/>.

³⁴ "Node.js." <https://nodejs.org/>

³⁵ "PubSub does not work in the browser · Issue #518 · ipfs/js-ipfs-api" <https://github.com/ipfs/js-ipfs-api/issues/518>

3. Cordova & Phonegap

3.1. Cordova

Cordova [8] este un framework open-source de dezvoltare pentru dispozitive mobile creat de Nitobi³⁶ ce oferă avantajul de a concepe și implementa o aplicație folosind tehnologiile web arhicunoscute HTML5, CSS3 și Javascript.

Ca principal mod de expunere a aplicației către utilizator, acest framework se folosește de WebView, o tehnologie ce permite dispozitivelor să ruleze aplicații ce se comportă precum în cadrul unor browsere populare (ex. Chrome). Pe lângă aceste caracteristici, Cordova expune un API ce oferă acces la funcționalitățile native ale dispozitivului (accelerometru, cameră, geolocație și altele).

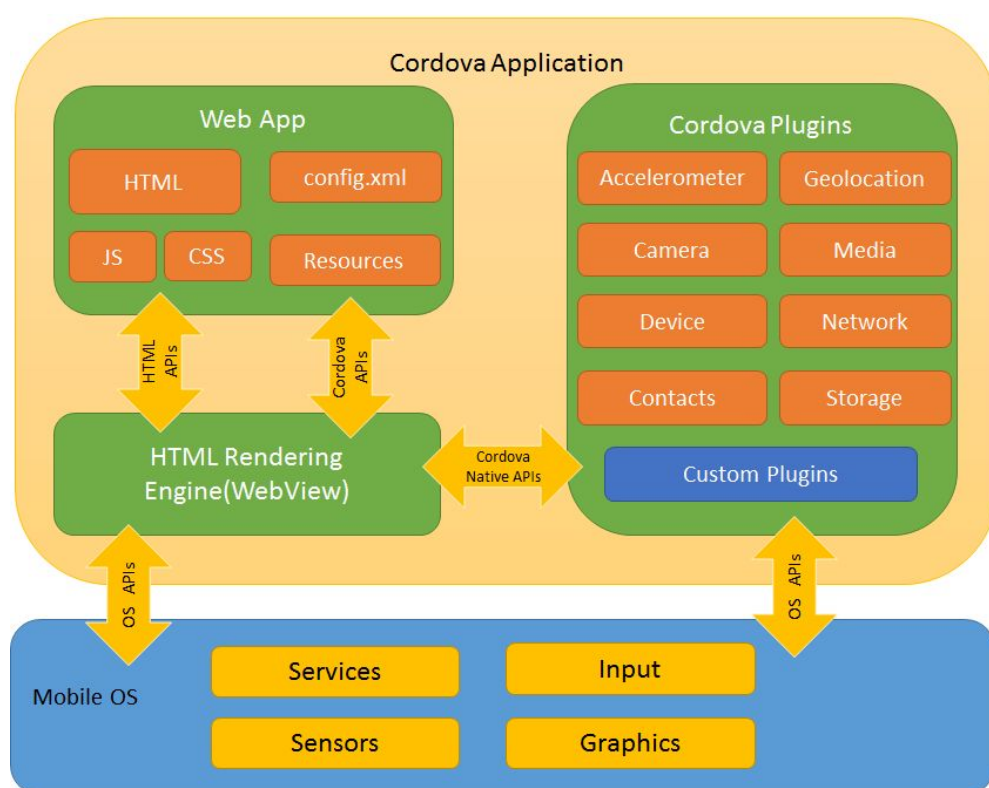


Figura B.40: arhitectura unei aplicații Cordova³⁷

Aplicația mobilă este concepută la bază ca fiind una web care “trăiește” în interiorul unui WebView³⁸ folosind componente native ale device-ului pe care rulează. Toate funcționalitățile de la nivel nativ sunt implementate cu limbajele specifice sistemului de operare ale dispozitivului (ex. Android - Java, iOS - Objective C). Expunerea acestora se face

³⁶ "Nitobi Software | crunchbase." <https://www.crunchbase.com/organization/nitobi-software>

³⁷ "Cordova App Architecture"

<https://cordova.apache.org/static/img/guide/cordovaapparchitecture.png>

³⁸ "WebView | Android Developers."

<https://developer.android.com/reference/android/webkit/WebView.html>

prin API-ul global Cordova sub formă de funcționalități Javascript. Fiecare pagină HTML și stilizare CSS împreună cu script-urile aferente Javascript sunt împachetate de către framework într-o aplicație mobilă specifică platformei pe care se face dezvoltarea.

```
xsky@inf myProj $ cordova
Synopsis

  cordova command [options]

Global Commands
  create ..... Create a project
  help ..... Get help for a command
  telemetry ..... Turn telemetry collection on or off
  config ..... Set, get, delete, edit, and list global cordova options

Project Commands
  info ..... Generate project information
  requirements ..... Checks and print out all the requirements
                    for platforms specified

  platform ..... Manage project platforms
  plugin ..... Manage project plugins

  prepare ..... Copy files into platform(s) for building
  compile ..... Build platform(s)
  clean ..... Cleanup project from build artifacts

  run ..... Run project
              (including prepare && compile)
  serve ..... Run project with a local webserver
              (including prepare)

Learn more about command options using 'cordova help <command>'

Aliases
  build -> cordova prepare && cordova compile
  emulate -> cordova run --emulator

Options
  -v, --version ..... prints out this utility's version
  -d, --verbose ..... debug mode produces verbose log output for all activity,
  --no-update-notifier ..... disables check for CLI updates
  --nohooks ..... suppress executing hooks
                    (taking RegEx hook patterns as parameters)

Examples
  cordova create myApp org.apache.cordova.myApp myApp
  cordova plugin add cordova-plugin-camera
  cordova platform add android
  cordova plugin add cordova-plugin-camera --nosave
  cordova platform add android --nosave
  cordova requirements android
  cordova build android --verbose
  cordova run android
  cordova build android --release -- --keystore="..\android.keystore" --storePassword=android --alias=mykey
  cordova config ls
  cordova platform add ios --nofetch
  cordova plugin add cordova-plugin-camera --nofetch
```

Figura B.41: API CLI Cordova

Principalele avantaje oferite de Cordova (Figura B.41) sunt crearea de proiecte, adăugarea de diverse platforme, management-ul plugin-urilor, testarea și compilarea aplicațiilor dar și alte configurații.

3.2. Phonegap

Phonegap [9] este versiunea Adobe de Cordova care moștenește toate funcționalitățile acesteia oferind multe altele în plus. Printre acestea se numără Phonegap Build Service³⁹, un serviciu din cadrul unui mediu Cloud pentru împachetarea aplicațiilor și ușurinta configurărilor aferente.

3.3. Plugins

Probabil cel mai important aspect al ecosistemului Cordova, plugin-urile oferă o interfață pentru acesta și componentele native ce comunică între ele. Acestea oferă posibilitatea invocării codului nativ folosind Javascript⁴⁰.

Platforma dispune de o serie de plugin-uri de bază oficiale pe care le menține și le îmbunătățește în mod activ dar și posibilitatea de folosire a plugin-urilor “third-party” concepute de alți dezvoltatori.

În cadrul aplicației H.O.L.D.I.N, o serie de plugin-uri⁴¹ merită prezentate pe scurt:

- **cordova-plugin-file**: plugin ce oferă un API pentru lucrul cu fișiere din cadrul stocării pe un anumit device
- **cordova-plugin-geolocation**: plugin ce pune la dispoziția utilizatorului acces la funcționalitățile de localizare GPS
- **cordova-plugin-network-information**: oferă informații privitoare la conexiunea device-ului la Internet (ex. mobile data, wifi)
- **cordova-plugin-ipfs[6]**: plugin implementat personal pentru oferirea posibilității de instanțiere nativă a unui daemon IPFS
- **storage**: plugin din seria plugin-urilor de bază Cordova; oferă un mediu de stocare de tipul (*key*, *value*) (“Local Storage”)
- **cordova-plugin-local-notifications**⁴²: plugin third-party ce permite utilizatorilor aplicației să primească notificări asemănătoare celor “push” la nivel local

³⁹ "Adobe PhoneGap Build." <https://build.phonegap.com/>

⁴⁰ "Plugin Search - Apache Cordova." <https://cordova.apache.org/plugins/>

⁴¹ "Plugin APIs - Apache Cordova."

<https://cordova.apache.org/docs/en/4.0.0/cordova/plugins/pluginapis.html>

⁴² "GitHub - katzer/cordova-plugin-local-notifications"

<https://github.com/katzer/cordova-plugin-local-notifications>

4. Vue.js

Vue.js [10] este un framework de Javascript “client-sided” de tipul MVC dezvoltat de către Evan You⁴³. Principalul scop al acestuia este de a oferi diverse metode de construire a unei interfețe utilizator în manieră modulară și incrementală. Conceput să fie “lightweight” și ușor de utilizat, Vue.js încearcă să se detașeze de marii “giganți” ai framework-urilor de dezvoltare client Javascript, în același timp împrumutând de la ei concepte și idei, simplificându-le. Printre principalele caracteristici ale acestuia se numără:

- **declarative rendering**: afișarea datelor în DOM folosind un sintaxa de “templating”
- **conditionals & loops**: utilizarea de condiții logice și a structurilor de iterație pentru lucrul cu date direct în pagină
- **components**: posibilitatea de creare și manipulare a componentelor concepute de utilizator
- **reactivity**: oferă o legătură directă între datele propriu-zise și cele afișate (“two-way data binding”)

La modul general, utilizarea Vue.js-ului este una intuitivă și ușoară. Dezvoltatorii declară o instanță Vue “rădăcină” fixată într-un punct rădăcină în pagină. Asupra acesteia se pot adăuga câmpuri de date, metode, componente custom pe care dezvoltatorul le poate manipula pentru a realiza o interacțiune logică.

5. Framework7

Framework7 [11] este un framework HTML creat de către Vladimir Kharlampidi⁴⁴ pentru dezvoltarea aplicațiilor pe mobile hibride și nu numai. Acesta oferă o serie de componente și funcționalități pentru interfața utilizator pe care dezvoltatorii le pot utiliza “out-of-box” sau le pot modifica după bunul plac. Orientat spre design iOS și Google Material, Framework7 poate fi folosit împreună cu alte framework-uri orientate client de Javascript (Vue.js sau React) ori cu Javascript pur.

Fie ca e vorba despre pagini, butoane, tabele, elemente de input sau diverse funcționalități specifice mobile (“infinite scroll”, “swipe” și altele), Framework7 oferă o gamă largă de posibilități de utilizare ușoară a lor în propriile aplicații, reducând timpul acordat conceperii și stilizării unei interfețe utilizator.

⁴³ "Evan You." <http://evanyou.me/>

⁴⁴ "nolimits4web (Vladimir Kharlampidi) · GitHub." <https://github.com/nolimits4web>

6. NPM & Webpack

NPM [12] (Node Package Manager) este o utilitară pentru limbajul Javascript reprezentată de un manager de pachete. Acesta permite instalarea și administrare de pachete ce oferă diverse alte funcționalități în cadrul unui proiect. Conținând o gamă largă de astfel de pachete în propriul registru global, npm ușurează munca unui dezvoltator lăsându-i posibilitatea de a reutiliza diverse soluții concepute de alți dezvoltatori în vederea rezolvării unor probleme comune. În aplicației cadrul H.O.L.D.I.N acesta reprezintă utilitarul principal de administrare a pachetelor proiectului.

WebPack [13] este un utilitar Javascript ce ajută la “împachetarea” modulelor npm dar și a celor personale, parcurgând recursiv listele de dependențe ale pachetelor instalate în cadrul pachetului și împachetându-le într-un număr mic de “pachete” (în practică, de multe ori, unul singur). În aplicației cadrul H.O.L.D.I.N, webpack “adună” toate dependențele altor pachete și framework-uri (js-ipfs-api, Vue, F7) într-un singur “bundle” pe care apoi îl oferă aplicației spre utilizarea sa în cadrul Webview-ului.

C. Arhitectură H.O.L.D.I.N.

1. Considerații preliminare

Plecând de la ideea că aplicația trebuie să lucreze atât în regim “online” dar și “offline” din punct de vedere al lipsei de conexiune cu rețeaua globală de Internet, o serie de detalii merită specificate. Contextul inițial imaginat de utilizare a aplicației este acela în care o zonă geografică nu are acces la Internet din diverse motive dar dispune încă de infrastructuri locale ce rămân interconectate (multiple rețele LAN, zone ale unui oraș sau un oraș întreg). IPFS-ul intervine în asigurarea comunicării între nodurile conectate la astfel de rețele prin natura sa protocolară.

Viziunea de ansamblu inițială a sistemului **H.O.L.D.I.N.** [7] a fost crearea unei aplicații mobile (Android) ușor de utilizat dar destul de puternică pentru a oferi utilizatorilor posibilitatea de localizare a altor utilizatori ce au nevoie de ajutor. Aplicând principiul de “no single point of failure” asupra realității, s-a încercat eliminarea dependenței de autoritățile principale de la care o persoană poate primi ajutor și “distribuirea” a astfel de cereri către alte persoane din imediata vecinătate a punctelor afectate de diverse evenimente.

Din punct de vedere al implementării am efectuat o serie de teste folosind combinații ale tehnologiilor prezentate mai sus. Pentru ușurința implementării am folosit Vue.JS și Framework7 spre a dezvolta o interfață intuitivă, tehnologii care nu necesită un grad mare de efort al învățării și utilizării.

În legătură cu variantele IPFS-ului de implementare, JS-IPFS [16] este cea care ocupa primul loc în utilizare, luând în calcul și faptul că tehnologia Cordova oferă un mediu de dezvoltare pentru aplicații web. Lipsa temporară a unor funcționalități ce se află încă în dezvoltare și testare de către comunitate precum necesitatea folosirii unui server WebRTC Star în lipsa rutării folosind DHT-urile și problemele de descoperire locală observate experimental (lipsa tehnologiei mDNS în browser) au dus fluxul de implementare în altă direcție. Varianta Go-IPFS, deși dispune de cele mai multe caracteristici, nu respectă interfața principală [5] iar limbajul Javascript se pretează mult mai bine pentru dezvoltarea de aplicații web în special pe partea de client decât limbajul Go.

Principala soluție prevăzută luând în calcul aspectele “pro” și “contra” observate experimental este utilizarea implementării Go pentru instanțierea nodului IPFS și conectarea sa la rețea (descoperirea locală și cea globală funcționează corect) și utilizarea API-ului Javascript a IPFS-ului [14] pentru controlul asupra nodului și expunerea funcționalităților.

2. Scheletul general & detaliile implementării

În primă fază a fost necesară conceperea și implementarea unui plugin Cordova, **cordova-plugin-ipfs** [5], ce oferă aplicației posibilitatea de inițializare a unui repository local IPFS și ridicarea unui daemon pentru conexiunea cu rețeaua.

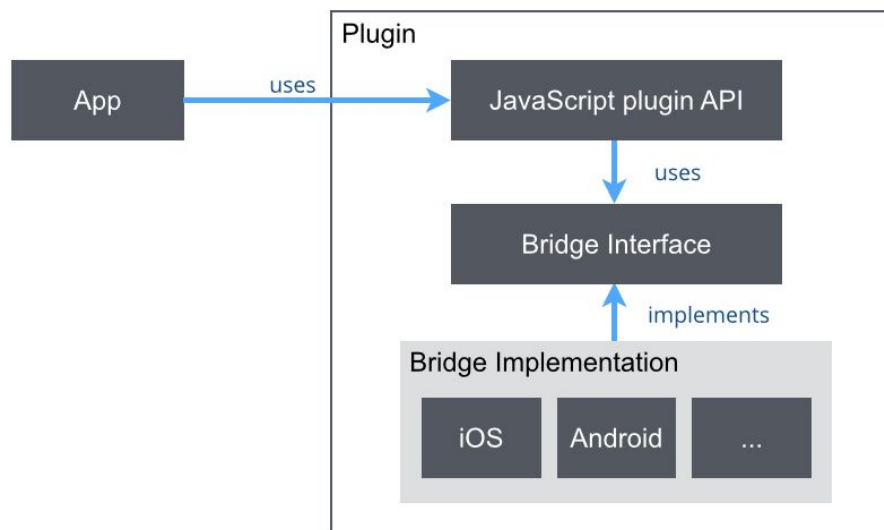


Figura C.1: arhitectura unui plugin Cordova⁴⁵

În (Figura C.1) se poate observa arhitectura generală a unui plugin Cordova, subliniind faptul că acesta implementează diverse funcționalități folosind limbajul de programare și librăriile native (Java în cazul Android) iar mai apoi le expune aplicației printr-un API Javascript.

cordova-plugin-ipfs[5] verifică dacă în spațiul de stocare alocat aplicației există fișierele “binary” ale variantei Go ARM de implementare a IPFS-ului iar în caz contrar le descarcă și le dezarchivează. În continuare plugin-ul verifică existența repo-ului local IPFS, inițializând sistemul dacă acesta nu a fost în prealabil. Se trece mai apoi la configurarea nodului pentru permiterea primirii de conexiuni din partea unui API. Daemon-ul poate fi pornit și oprit apoi folosind interfața plugin-ului. Toate aceste configurații sunt realizate prin execuția apelurilor de comenzi “shell” din Android iar daemon-ul IPFS este rulat într-un fir de execuție separat din “pool-ul” Cordova.

Plugin-ul expune astfel în mediul Cordova inițializarea nodului cu diverși parametri de configurare, pornirea și oprirea daemon-ului IPFS.

⁴⁵ "Ionic: An AngularJS based framework on the rise - codecentric Blog." 28 Nov. 2014, <https://blog.codecentric.de/files/2014/11/plugin.png>

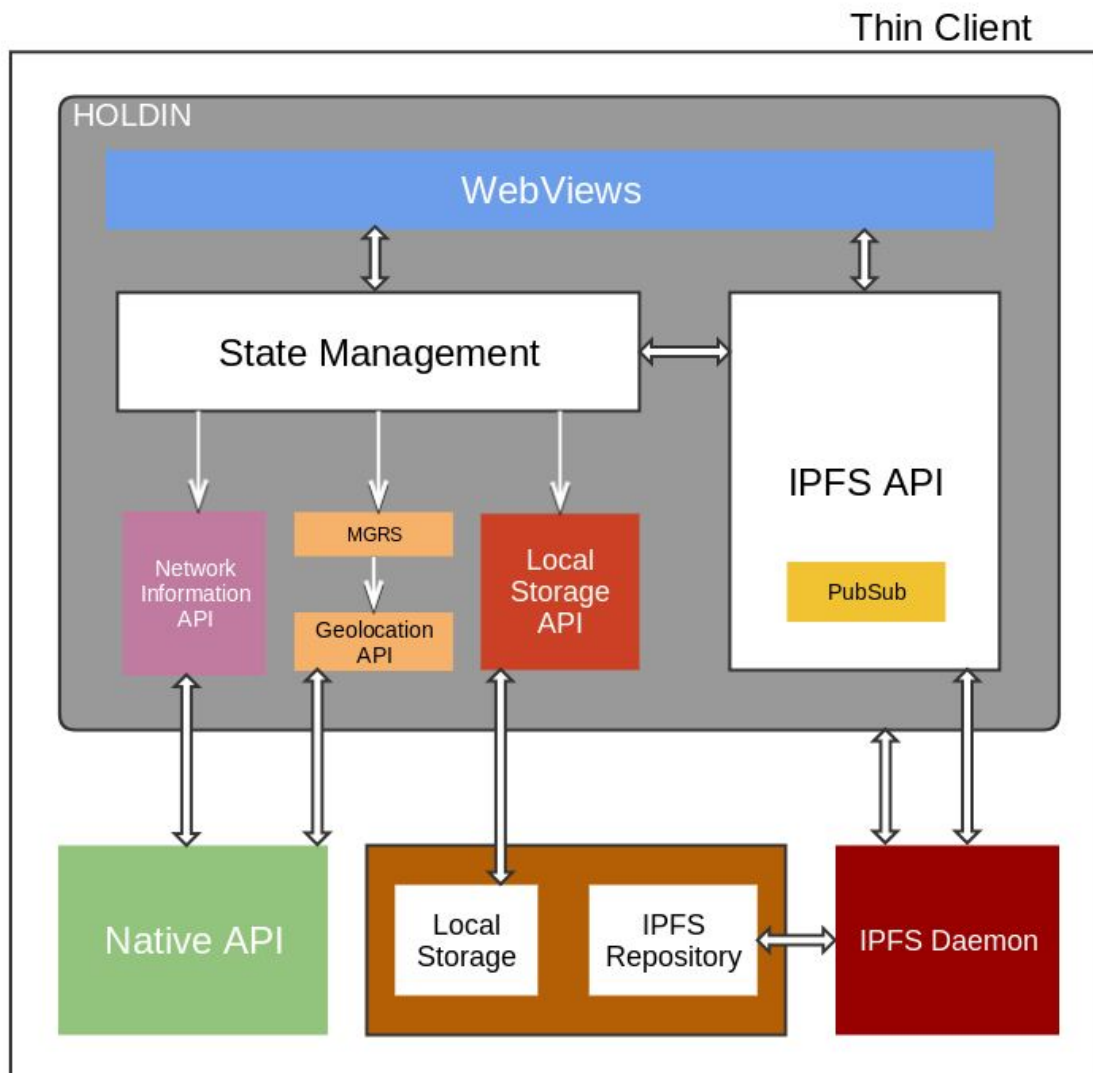


Figura C.2: diagrama de arhitectură H.O.L.D.I.N

Sistemul H.O.L.D.I.N, având ca și fundație mediul de dezvoltare Cordova, dispune de 3 mari componente: interfața utilizator (**WebViews**) , manager-ul de stare (**State Management**) și **API-ul Javascript** al protocolului IPFS. Acestea se folosesc de diverse alte API-uri și funcționalități între care există o continuă comunicare pentru preluarea, procesarea, transferul și prezentarea datelor.

Interfața grafică utilizator implementată cu Vue.js și Framework7 este forma de prezentare principală a aplicației ce respectă principiul Material Design⁴⁶ (Google). Fiecare “pagină” este reprezentată de un fișier sau componentă de tip Vue + Framework7 prin care utilizatorul poate interacționa cu aplicația și caracteristicile sale. Printre acestea se numără:

⁴⁶ "Introduction - Material design - Material design guidelines." <https://material.io/guidelines/>.

- **Login Popup:** o pseudo-pagină ce conține un formular de completare a datelor și preferințelor personale (nume utilizator, durata de viață a mesajelor, timpul de actualizare a zonei etc.)
- **Home:** o pagină descriptivă a aplicației cu informații generale
- **Alerts:** pagina “nucleu” a aplicației; conține o “afișare arborescentă” sub forma unui timeline a mesajelor primite de către utilizatorul respectiv și scurte detalii pentru fiecare mesaj; în cadrul paginii se află și butoanele pentru filtrarea și trimiterea de mesaje
- **Profile:** pagină ce oferă utilizatorului informații privitoare la starea sistemului, datele personale, locația curentă etc.
- **Settings:** pagină pentru realizarea setărilor personale sau de sistem
- **Message View:** oferă informații legate de un anumit mesaj primit
- **Send Message:** conține formularul principal de trimitere a unui mesaj

Fiecare pagină a fost concepută și stilizată spre ușurința utilizării aplicației și oferă destule informații pentru ca utilizatorul să folosească în mod corect sistemul. Principalele elemente prezente în pagini au fost create cu ajutorul componentelor de UI introduse de Framework7 și a iconițelor Google Material Design [17] prin aranjare proprie. La primirea mesajelor este creată o notificare nativă Android iar în cazuri de eroare sunt afișate alerte cu diferite mesaje legate de evenimentul respectiv. Interfața utilizator trimite / primește informații spre / de la sistemul propriu-zis de procesare și administrare a diverselor configurații, mesaje sau subsisteme. Având la bază framework-ul Vue, aceasta dispune de reactivitate: datele modificate în sistemele de stocare sunt afișate imediat în interfață, realizându-se astfel legătura “two-way data binding”.

La baza fiecărui Vue și a componentelor aferente se află un schelet de marcarea HTML (template), metodele ce acționează asupra entității respective și câmpurile ce conțin datele afișate. Este de menționat faptul că pagina “Alerts” dispune de un timeline de afișare a mesajelor implementat cu un “virtual list” oferit de Framework7. Acesta oferă posibilitatea de afișare a unui număr mare de elemente cu performanțe ridicate (ordinul zecilor de mii).

Toate paginile sunt încărcate la pornirea aplicației, navigarea între ele făcându-se prin acțiunea de “left, right swipe” sau prin bara de navigație dispusă în partea inferioară. Este folosit și un procedeu de “rutare” între unele afișări precum aducerea în prim plan a detaliilor unui anumit mesaj primit în funcție de identificatorul unic al acestuia.

State Management (managerul stărilor) este principalul modul ce se ocupă de stocarea, modificarea și administrarea datelor personale și ale sistemului ca întreg. A fost conceput ca o singură entitate pentru asigurarea reactivității și a unui singur “source-of-truth” ce reține totalitatea datelor în momentul execuției.

În cadrul acestui modul se află informațiile vitale necesare aplicației:

- `nodeId`: id-ul nodului IPFS
- `username`: numele de utilizator folosit în mesaje
- `messages`: lista propriu-zisă a mesajelor afișate
- `messagesTTL`: durata de viață a unui mesaj (în zile)
- `locationZone`: obiect ce reține id-ul zonei și hash-ul ei
- `locationExact`: obiect ce reține longitudinea și latitudinea curentă
- `zoneUpdaterDelay`: durata de timp după care se face actualizarea locației
- `statuses` (`Network`, `Location`, `IpfsRepo`, `Daemon`, `PubSub`): starea fiecărui subsistem (`true` = Online, `false` = Offline)

Toate datele personale și de configurare a sistemului sunt salvate cu ajutorul API-ului **Local Storage**. Acesta oferă un mod simplu, sincron de stocare sub forma (`key`, `value`) unde `key` reprezintă un identificator iar `value` o valoare de tip `string` (structurile complexe de date necesitând serializarea). În afară de lista de mesaje “în viață” ce este salvată sub formă de listă de identificatori hash ai IPFS-ului, restul datelor sunt stocate direct folosind acest API. La pornirea sistemului sunt “completate” toate câmpurile cu informații prin aducerea lor din mediile de stocare iar la închiderea aplicației acestea sunt salvate în același punct de origine. Lista de mesaje este iterată și pentru fiecare identificator al unui mesaj se apelează API-ul IPFS pentru aducerea obiectului-mesaj salvat în format JSON în repo-ul local IPFS.

Pentru actualizarea locației este folosit **API-ul Geolocation** care este pornit odată cu aplicația în mod implicit. Actualizatorul geolocației va modifica obiectul ce conține longitudinea și latitudinea odată ce apare o schimbare în locația reală. Un alt actualizator de zonă este implementat în aplicație pentru modificarea “zonei” determinată de **MGRS** (Anexa 1) la un interval de X minute (număr specificat de utilizator). Atunci când este apelată funcția de actualizare a zonei, se execută transformarea coordonatelor exacte într-o zonă MGRS asupra căreia se aplică funcția hash **SHA1** pentru generarea unui identificator unic, aceste două date fiind salvate în obiectul zonei de locație. Dacă serviciul de localizare nu este pornit pe dispozitiv, va fi afișată o alertă pentru notificarea utilizatorului.

Mesajele vehiculate între utilizatorii aplicației sunt trimise cu ajutorul conceptului **PubSub** folosind API-ul Javascript IPFS care controlează daemon-ul IPFS. Acestea pot fi de 3 tipuri: “alert” (un mesaj urgent), “warning” (mesaj de avertizare) și “info” (mesaj de informare). Ele au ca scop cererea de ajutor, avertizarea în cazul apropierii unei calamități sau a unor evenimente dezastruoase sau trimiterea de informații generale. Publicarea lor se face pe “canale” PubSub determinate de zonele în care se află fiecare utilizator. Aceste zone sunt descrise de conceptul MGRS și reprezintă “fâșii” geografice cu aria maximă de 100 KM pătrați. Când un utilizator iese din zona sa și intră în altă zonă, sistemul actualizează zona curentă și hash-ul ei, schimbând astfel canalul spre care se face trimiterea de mesaje sau de la care se primesc acestea. Mesajele conțin:

- **id:** identificator unic, inițial `null`, determinat de hash-ul sub care IPFS salvează mesajul
- **from:** id-ul nodului IPFS ce a publicat mesajul
- **username:** numele utilizatorului care a trimis mesajul
- **locationZone:** zona determinată de MGRS
- **locationExact:** coordonatele ultimei locații exacte (longitudine, latitudine)
- **time:** timpul trimiterii mesajului (oră, minute, secunde)
- **day:** ziua trimiterii mesajului (1-31)
- **month:** luna trimiterii mesajului (1-12)
- **year:** anul trimiterii mesajului
- **type:** tipul mesajului (“alert”, “warn” sau “info”)
- **details:** detalii oferite de utilizatorul ce trimite mesajul

Trimiterea și primirea mesajelor prin PubSub se realizează prin vehicularea doar a hash-ului adresare-conținut identificator de obiecte IPFS și nu prin obiectul propriu-zis. Obiectele sunt adăugate în repo-urile locale iar sistemul fiecărui utilizator folosește hash-urile mai sus menționate pentru cererea obiectului respectiv de la nodurile din rețea, neîncărcându-se astfel canalul PubSub și lăsând găsirea și transportul obiectului în seama IPFS-ului, DHT-urilor și a protocolului de transport. La primirea obiectului-mesaj, acesta este introdus în lista de mesaje locală a aplicației și mai apoi afișat după determinarea poziției elementului în listă prin *căutare binară* folosind data și timpul mesajului ca și câmpuri comparative. La fiecare 1 minut (număr fix) se va itera lista locală de mesaje și vor fi determinate ce mesaje nu mai sunt “în viață” folosind numărul de zile specificat de utilizator.

Printre alte funcționalități se numără și starea rețelei, aceasta fiind asigurată de **API-ul Network Information**. În cazul în care nu este pornit serviciu de date mobile sau conexiunea wireless, este adusă o înștiințare utilizatorului sub forma unei alerte. Sistemul va afișa această alertă până când utilizatorul va porni unul dintre cele doua servicii pe dispozitivul respectiv. State Management se ocupă de asemenea și de setarea stărilor fiecărui serviciu.

La pornirea aplicației H.O.L.D.I.N, sistemul va utiliza plugin-ul **cordova-plugin-ipfs[6]** (Anexa 2) care verifică starea fișierelor “binary” și a repo-ului local IPFS. În cazul în care acestea nu există, aplicația va fi pusă în așteptare până la descărcarea și dezarhivarea protocolului dar și până la inițializarea cu succes și configurarea sistemului. Odată inițializate, un **daemon IPFS** va fi pornit într-un fir de execuție și proces “shell” separat de aplicație. La pornirea cu succes a acestuia, **API-ul IPFS[14]** va încerca o conectare cu el. Dacă este realizată cu succes conectarea la daemon, se vor porni actualizatorii de locație și de zonă și se va face “subscribe” la zona determinată MGRS pentru primirea efectivă a mesajelor. La trimiterea unui mesaj, acesta este serializat și adăugat cu ajutorul API-ului sub formă de obiect IPFS urmând mai apoi să i se facă “publish” la identificatorul unic spre canal. La schimbarea unei zone, sistemul va face “unsubscribe” de la canalul respectiv și va face “subscribe” la zona nouă.

Sistemul folosește de asemenea plugin-ul “third-party” **cordova-plugin-background-mode**⁴⁷ pentru a oferi posibilitatea aplicației să ruleze în fundal atât timp cât utilizatorul nu interacționează cu aceasta.

D. Cazuri de utilizare

În cadrul aplicației H.O.L.D.I.N utilizatorul are o interacțiune cu rețeaua IPFS mediată de către subsistemele prezente în aceasta. În continuare putem analiza câteva cazuri de utilizare împreună cu diverse observații ce merită precizate.

1. Pornire și “înregistrare”

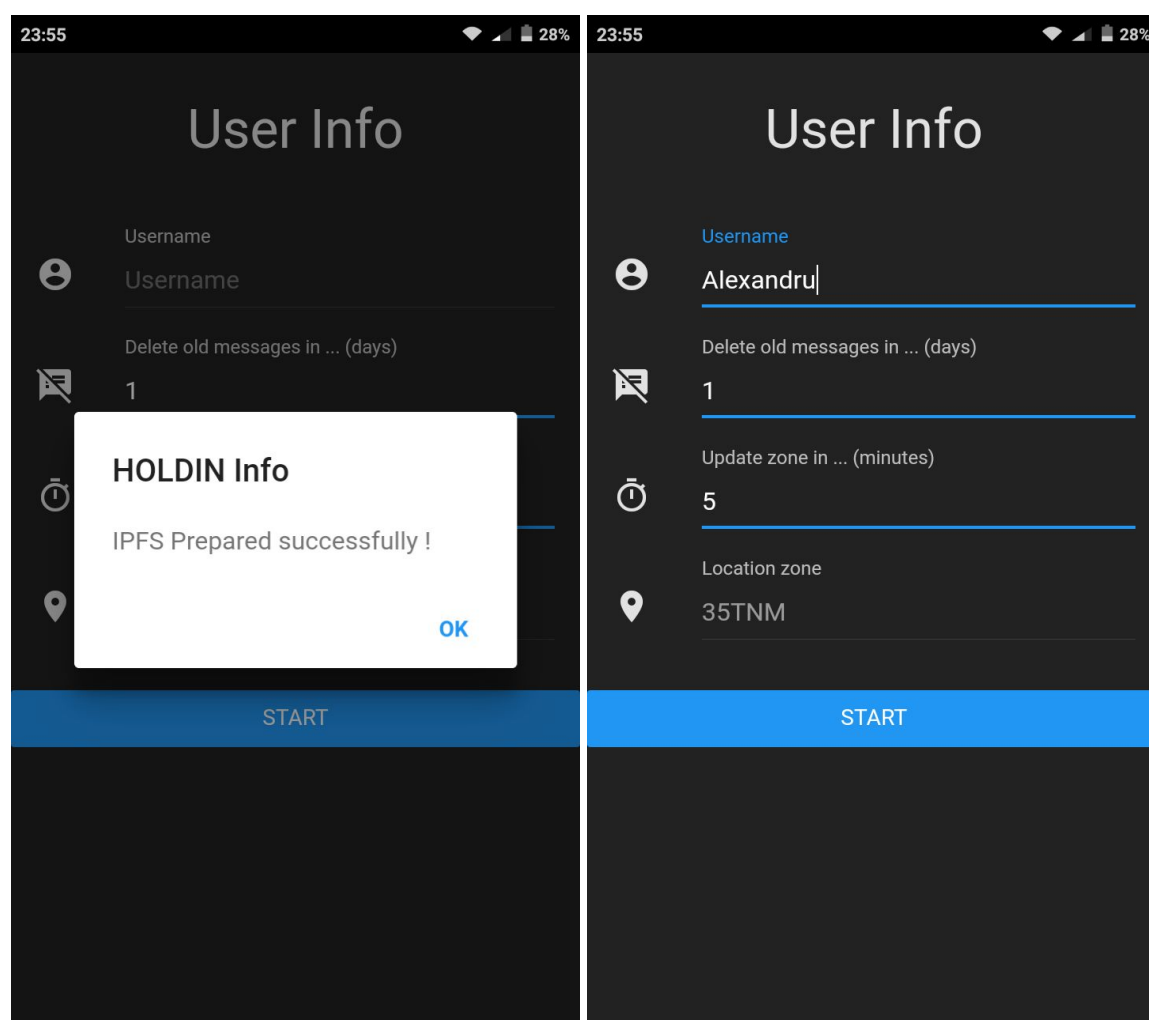


Figura D.1, D.2: inițializare și detalii utilizator

La pornirea aplicației H.O.L.D.I.N se va verifica existența protocolului și a repo-ului local. Utilizatorul va fi pus în așteptare dacă este necesară pregătirea și configurarea lor. După finalizarea acestor aspecte, se va afișa mesajul din (Figura D.1) după care se va afișa

⁴⁷ "katzer/cordova-plugin-background-mode"
<https://github.com/katzer/cordova-plugin-background-mode>.

formularul pentru configurațiile personale (Figura D.2). La apăsarea butonului “Start” se vor salva aceste informații și se va ascunde popup-ul formularului. Detaliile personale și de stare a sistemului se pot observa în (Figura D.3).

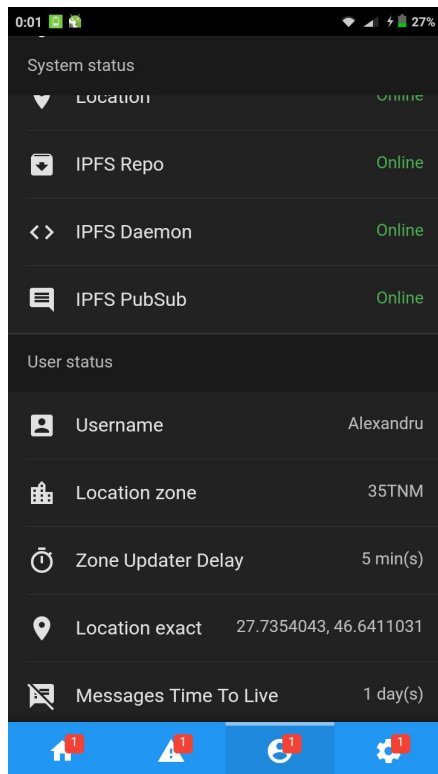


Figura D.3: starea sistemului

În aceasta se pot observa diversele stări ale sistemului (rețea, locație, repo, daemon și pubsub IPFS) împreună cu numele de utilizator, zona activă de vehiculare a mesajelor, timpul de actualizare a zonei, locația exactă în longitudine și latitudine dar și “timpul de viață” a mesajelor. În partea inferioară se află bara de navigație pentru accesarea paginilor și notificările aferente lor.

2. Lipsa rețelei

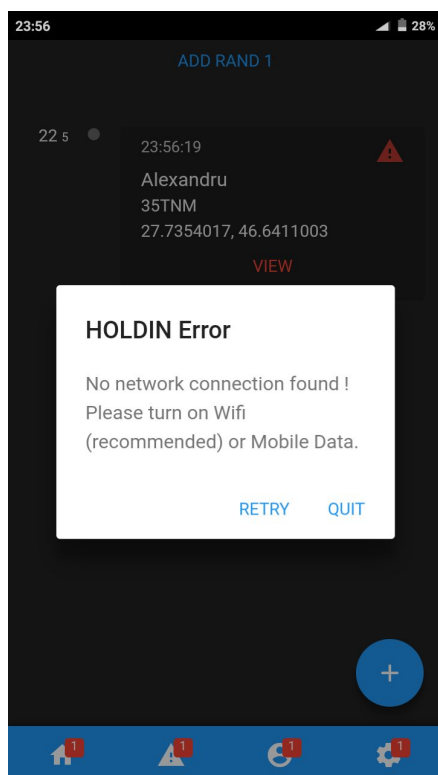


Figura D.4: eroare la conexiune

Dacă la pornirea sistemului sau în timpul utilizării nu există o conexiune activă (wireless sau date) se va afișa un mesaj de eroare (Figura D.4). Odată existentă o conexiune, se poate apăsa butonul “Retry” pentru revenirea la utilizarea normală. Daemon-ul IPFS are avantajul de a se reconecta automat la rețea atunci când apar astfel de cazuri, nefiind necesare alte acțiuni asupra sistemului pentru permiterea trimiterii și primirii de mesaje.

3. Trimiterea și afișarea de mesaje

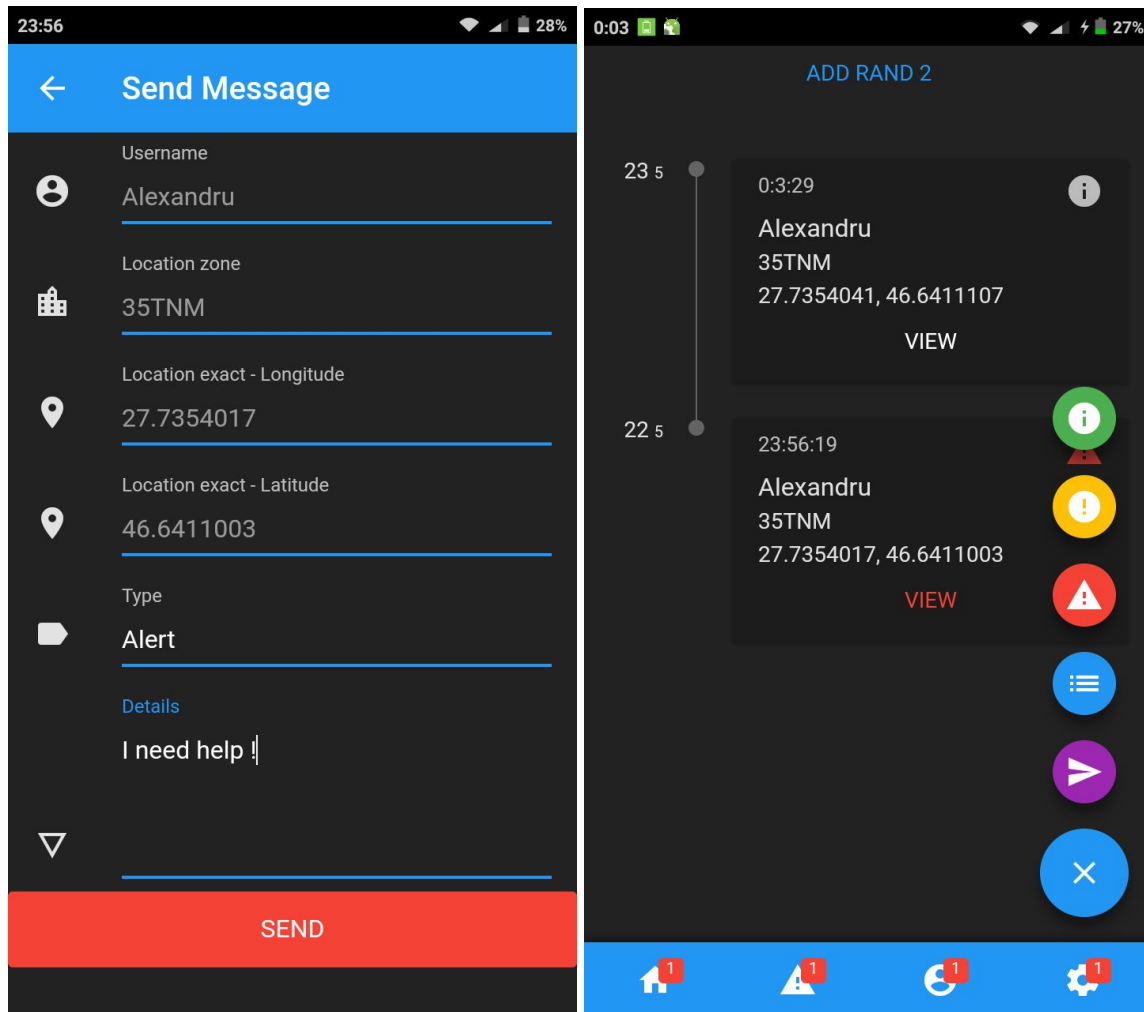


Figura D.5, D.6: trimiterea și afișarea mesajelor

Afișarea formularului de trimitere a mesajelor (Figura D.5) se face prin apăsarea butonului de trimitere din lista de butoane (butonul mov). Primele 5 câmpuri din acesta sunt completate de către sistem în mod automat cu informațiile de bază și nu pot fi modificate la trimitere. Utilizatorul poate alege tipul mesajului și poate specifica detaliile proprii în acesta. După apăsarea butonului “Send” acesta este adăugat în IPFS și trimis prin PubSub folosind API-ul Javascript și daemon-ul IPFS.

În (Figura D.6) se pot analiza mesajele primite și trimise din cadrul paginii de mesaje. Acestea sunt sortate în funcție de data și ora propagării lor în rețea. În cadrul elementelor de prezentare se află și butonul “View” ce afișează pagina cu toate detaliile mesajului respectiv. Pe lângă acestea se pot observa butoanele de filtrare în funcție de tipul mesajului (verde, galben, roșu) și butonul de afișare a mesajelor totale (albastru).

E. Concluzii

1. Direcții viitoare de cercetare & probleme întâlnite

InterPlanetary File System este un protocol modern care ar putea îmbunătăți exponențial Internetul așa cum este perceput de societate. Acesta evoluează în fiecare zi, comunitatea venind cu idei noi și rezolvări de erori, fiind plănuită atât adăugarea de noi funcționalități și tehnologii ce-l pot întregi dar și implementarea în diverse medii și limbaje de programare cunoscute.

Așa cum au fost prezentate, există unele probleme existente în cadrul implementărilor: varianta Go nu respectă interfața de bază, varianta Javascript nu dispune momentan de “discovery” și depinde de un server de semnalizare WebRTC, API-ul Javascript nu permite utilizarea PubSub-ului în medii precum browserele uzuale și necesită modificarea acestuia temporară de către utilizatori. Restul API-urilor concepute în alte limbaje nu implementează interfața de bază și nu dispun de funcționalitățile utilizării PubSub-ului. De asemenea se pot realiza îmbunătățiri asupra sistemelor de rutare și a protocoalelor propriu-zise de transport pentru scurtarea timpului de găsim a fișierelor. Variantele de implementare sub conceptul Web vor primi în viitor o serie de îmbunătățiri fie că e vorba de servere personale de semnalizare, “relay”, rutare sau utilizarea altor protocoale decât WebRTC.

O altă direcție interesantă de dezvoltare ar fi adăugarea suportului pentru protocol în browser-ele populare. Astfel, utilizatorii nu ar trebui să mai depindă de serverul Gateway expus de comunitatea IPFS sau de configurarea și inițializarea manuală a protocolului și ar putea accesa aplicații și fișiere în rețea folosind doar `ipfs:/`

2. Avantajele IPFS-ului

Pornind de la conceptele teoretice create de Juan Benet în legătură cu protocolul, IPFS vrea să aducă inovații și îmbunătățiri în cadrul sistemelor distribuite de fișiere prin utilizarea și combinarea celor mai bune tehnologii din prezent.

Fie că e vorba de permanența datelor, accesul la o submulțime de noduri din rețea în cazul lipsei de conexiune la rețeaua globală, lipsa dependenței de unități centrale de conectare și administrare, IPFS ar putea excela în diverse idei și materializările acestora: folosirea ca și sistem global de fișiere, sistem de versionare, sistem de partajare criptată a datelor, manager de pachete pentru sisteme de operare, sistem de bază de date, CDN sau în cel mai mare caz, un nou Web unde datele persistă între nodurile ce sunt interesate de ele și link-urile nu mor.

F. Bibliografie

- [1] J. Benet. IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3). URL: <https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf>
- [2] J. Benet, David Dias - libp2p specs: requirements. URL: <https://github.com/libp2p/specs/blob/master/3-requirements.md>
- [3] J. Benet, David Dias - libp2p specs: architecture. URL: <https://github.com/libp2p/specs/blob/master/4-architecture.md>
- [4] Protocol Labs. Go-Ipfs: BitSwap. URL: <https://github.com/ipfs/go-ipfs/tree/master/exchange/bitswap>
- [5] Protocol Labs. IPFS Core Interface. URL: <https://github.com/ipfs/interface-ipfs-core>
- [6] **Cristea Alexandru-Gabriel**. cordova-plugin-ipfs. URL: <https://github.com/xSkyripper/cordova-plugin-ipfs>
- [7] **Cristea Alexandru-Gabriel**. H.O.L.D.I.N URL: <https://github.com/xSkyripper/holdin>
- [8] The Apache Software Foundation. Apache Cordova. URL: <https://cordova.apache.org/>
- [9] Adobe Systems Inc. Phonegap. URL: <https://phonegap.com/>
- [10] Evan You. VueJS. URL: <https://vuejs.org/>
- [11] Vladimir Kharlampidi. Framework7. URL: <https://framework7.io/>
- [12] npm, Inc. npm. URL: <https://www.npmjs.com/>
- [13] Tobias Koppers and other contributors. WebPack. URL: <https://webpack.js.org/>
- [14] Protocol Labs. Javascript IPFS API. URL: <https://github.com/ipfs/js-ipfs-api>
- [15] Protocol Labs. Go IPFS. URL: <https://github.com/ipfs/go-ipfs>
- [16] Protocol Labs. JS IPFS. URL: <https://github.com/ipfs/js-ipfs>
- [17] Google. Google Material Design Icons. URL: <https://material.io/icons/>
- [18] Military Grid Reference System. URL: https://www.maptools.com/tutorials/mgrs/quick_guide

G. Anexe

Anexa 1

MGRS[18] (Military Grid Reference System) este standardul utilizat de NATO pentru localizarea punctelor pe pământ. Fiind derivat din alte sistemele de coordonate (UTM, UPS) acesta dispune de avantajele localizării zonelor spațiale cu diferite precizii.

35T NN 4449 2283

Figura G.1: exemplu pentru formatul MGRS

În (Figura G.1) se poate observa un exemplu de coordonată determinată de sistemul MGRS. Primele două cifre și litera specifică (35T) reprezintă identificatorul zonei de gridă, acesta fiind determinat de fâșii UTM de 6° lățime numerotate de la 1 - 60 ce intersectează fâșii de 8° înălțime specificate prin literele de la C - X (excluzând "I" și "O" pentru a scăpa de confuzia cu 1 și 0). Următoarele două caractere (NN) reprezintă identificatorul unei zone de 100 KM² format dintr-o literă de la A - Z (excluzând "I" și "O") urmat de o altă literă de la A - V (excluzând "I" și "O").

Șirul numeric format din 8 cifre determină locația în zona de 100 KM² mai sus descrisă și are formatul de $n + n$ cifre, cu n de la 1 la 5. În funcție de numărul de cifre folosit, precizia variază astfel: 1 (precizie de 10 KM²), 2 (precizie de 1 KM²), 3 (precizie de 100 M²), 4 (precizie de 10 M²) și 5 (precizie de 1 M²). Zona descrisă în figură are o precizie de 10 M². Determinarea zonelor cu diferite precizii se face prin trunchiere și nu prin rotunjire (ex: 35T NN 444 228 pentru exemplul de mai sus reprezintă o precizie de 100 M² iar 35T NN este zona de 100 KM²).

Un alt aspect de merită menționat este cazul special al polilor. Aici zonele de gridă (primul identificator) sunt reprezentate de jumătăți de cerc notate cu A și B (polul sud) și literele Y, Z (polul nord). Zona de 100 KM² este dată de literele de la A - Z (primul caracter; se exclud "I" și "O") și literele de la A - Z (al doilea caracter; se exclud "I", "O", "D", "E", "M", "N", "V", "W").

Anexa 2 (Evoluția înțelegerii protocolului & Experimente)

InterPlanetary File System este suma tuturor tehnologiile după care a fost conceput de către Juan Benet și comunitatea ce s-a strâns în decursul timpului din pasiune și din dorința de creare a unui protocol modern, mai bun, ce ar putea avea implicații majore asupra Web-ului și Internet-ului în viitor. Îmbinând o număr mare de alte protocoale, paradigme și idei, nu este un lucru ușor de realizat înțelegerea în totalitate a tuturor detaliilor ce îl definesc. În special curba învățării acestuia este una tot mai abruptă ținând cont și de faptul că protocolul nu este complet în momentul de față, diferența dintre variantele implementării Go și Javascript accentuând acest lucru.

De la descoperirea IPFS-ului și până în prezent am efectuat o serie de teste și experimente în diferite medii și sub diverși factori pentru a scoate în evidență atât finele caracteristici ale sale dar și cazurile în care protocolul poate fi folosit pentru aducerea de beneficii în ideile de dezvoltare a aplicațiilor și ecosistemelor distribuite P2P. De asemenea am analizat și alte sisteme de fișiere distribuite precum AFS, NFS și GFS cu scop comparativ.

La o primă accesare a siteului <https://ipfs.io> pot fi analizate multe dintre problemele pe care IPFS încearcă să le rezolve în cadrul Internet-ului așa cum îl știm noi în prezent. Probabil cele mai importante avantaje ale protocolului care cad sub lumina inovației de la primul “impact” cu acesta sunt: distribuirea datelor în mod eficient, asigurarea permanenței lor, descentralizarea completă și posibilitatea de lucru în mediul offline.

Prima interacțiune personală cu protocolul a fost prin implementarea de referință Go. Aceasta poate fi obținută prin descărcarea sub formă de arhivă sau prin clonarea Git a repository-ului oficial [15] și compilarea și instalarea sa locală. Cea din urmă poate avea atât beneficii cât și dezavantaje datorită / din cauza faptului că dispune de ultimele schimbări asupra protocolului dar poate avea și diverse bug-uri nerezolvate temporar. Pentru ușurință am utilizat inițial varianta arhivată stabilă iar apoi pe cea din repository.

Observație: Fiecare comandă rulată în experimente acceptă parametrul `--help` cu care se pot analiza parametrii și argumentele utilizării ei.

```
xsky@inf ~ $ ipfs init
initializing IPFS node at /home/xsky/.ipfs
generating 2048-bit RSA keypair...done
peer identity: QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi
to get started, enter:

ipfs cat /ipfs/QmVLDahCY3X9P2uRudKARYuQFPM5zqA3Yij1dY8FpGbl7T/readme
```

Figura G.1: inițializare IPFS (Go)

Primii pași în înțelegerea protocolului se pot observa încă din inițializare (Figura G.1). Aici se poate vedea crearea repo-ului local cu rădăcină în directorul `$HOME` al user-ului autentificat pe acea mașină în mod implicit, generarea de perechi de chei PKI și a identității dar și un hash referință la “readme-ul” inițial. (Figura G.2)


```
xsky@inf ~ $ ipfs cat /ipfs/QmVLDAhCY3X9P2uRudKAryuQFPM5zqR3YiJidY8FpGbL7T/readme
Hello and Welcome to IPFS!
```



```
If you're seeing this, you have successfully installed
IPFS and are now interfacing with the ipfs merkle dag!
```

```
-----
| Warning:                                     |
| This is alpha software. Use at your own discretion! |
| Much is missing or lacking polish. There are bugs. |
| Not yet secure. Read the security notes for more. |
|-----
```

```
Check out some of the other files in this directory:
```

```
./about
./help
./quick-start    <-- usage examples
./readme         <-- this file
./security-notes
```

Figura G.2: readme-ul inițial IPFS (Go)

Deși nu este în totalitate explicată faza de inițializare, prin parcurgerea documentației oficiale și a repository-urilor GitHub ale comunității IPFS se pot afla informații precum faptul că directorului îi poate fi specificată o altă cale definită de utilizator iar acesta reține toate informațiile, configurările și blocurile de informații vehiculate între nod și rețea. Acesta poate fi șters manual folosind comanda `rm -rf ~/.ipfs` și sistemul poate fi reinițializat cu aceeași `ipfs init`. La fiecare regenerare (care poate fi efectuată chiar și offline) va fi creată o altă pereche de chei și un alt identificator de nod pentru mașina respectivă, unicitatea fiind asigurată de spațiul de lucru al criptosistemelor și al funcțiilor de hash folosite (RSA 2048-bit, SHA256), fiind extrem de puțin probabilă coliziunea dintre 2 identificatori.

În următoarea fază, și probabil una dintre cele mai importante, am analizat lucrul cu blocurile de date, fișierele și modul cum protocolul interacționează cu acestea dar și cu rețeaua. Ca și entități de test am folosit o imagine mare (~6 MB) și un fișier text (~10 KB). (Figura G.3)

```
xsky@inf anexa2_test $ tree -h
├── [4.0K]  my_data_root
│   ├── [5.5M]  cat.jpg
│   └── [4.0K]  my_dir
│       └── [9.9K]  some_text.txt
2 directories, 2 files
xsky@inf anexa2_test $
```

Figura G.3: date de test

Acestea pot fi introduse în sistem cu ajutorul comenzii `ipfs add` și a diversilor parametri pe care ea îi acceptă (ex. `-r` adaugă recursiv).

```
xsky@inf my_data_root $ ipfs add cat.jpg
added QmW1oDNGfhZsYwQxKmgVJ1CJ2S7jVxfrH8EBJPT1bSXKbJ cat.jpg
xsky@inf my_data_root $ ipfs add -r .
added QmW1oDNGfhZsYwQxKmgVJ1CJ2S7jVxfrH8EBJPT1bSXKbJ my_data_root/cat.jpg
added QmPic9FwaPzBe6wjeJcsC5EcPvgFFe9cC1zQVCR4o3Hx7f my_data_root/my_dir/some_text.txt
added QmTdjuXPLt62J6NGKFs2kCgDP5qzRAkaIKNCoib2tuUbgP my_data_root/my_dir
added QmTTaHUUiwA9ZQBjZsmXmvZyW6yM13F4mTNmMUT8qrwQLo my_data_root
xsky@inf my_data_root $ ipfs cat QmPic9FwaPzBe6wjeJcsC5EcPvgFFe9cC1zQVCR4o3Hx7f | head -n 3
Hello world !

This is a Lord of The Rings (J.R.R Tolkien) text in a test file that can "live" in IPFS.
xsky@inf my_data_root $ ipfs cat QmTdjuXPLt62J6NGKFs2kCgDP5qzRAkaIKNCoib2tuUbgP/some_text.txt | head -n 3
Hello world !

This is a Lord of The Rings (J.R.R Tolkien) text in a test file that can "live" in IPFS.
xsky@inf my_data_root $ ipfs cat QmTTaHUUiwA9ZQBjZsmXmvZyW6yM13F4mTNmMUT8qrwQLo/my_dir/some_text.txt | head -n 3
Hello world !

This is a Lord of The Rings (J.R.R Tolkien) text in a test file that can "live" in IPFS.
xsky@inf my_data_root $
```

Figura G.4: adăugarea de date în IPFS (Go)

În (Figura G.4) am adăugat recursiv arborele de fișiere precizat mai sus (Figura G.3). La prima vedere există câteva observații ce merită aduse în prim plan: readăugarea aceluiași fișier “cat.jpg” va determina exact același identificator hash atât timp cât entitatea nu se modifică (**deduplicare**), accesarea datelor se poate face prin diferite referențieri (direct hash-ul entității, hash-ul directorului părinte și numele entității ori hash-ul directorului rădăcină urmat de directorul copil și entitatea propriu-zisă) în stilul sistemelor de fișiere `unixfs`. Defapt toate aceste aspecte sunt determinate de **arborele Merkle**, “inima” IPFS-ului. Nodurile acestuia rețin datele entităților dar și referințe către entitățile copii din propriile descendențe.

În afară de interfața CLI a implementării, putem utiliza și browser-ul pentru accesarea fișierelor. Acest lucru necesită **ridicarea daemon-ului IPFS** (Figura G.5) aspect ce merită pus în lumină înainte de evidențierea interacțiunii cu rețeaua în sine.

```
xsky@inf ~ $ ipfs daemon --enable-pubsub-experiment
Initializing daemon...
Adjusting current ulimit to 2048...
Successfully raised file descriptor limit to 2048.
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/172.17.0.1/tcp/4001
Swarm listening on /ip4/172.18.0.1/tcp/4001
Swarm listening on /ip4/172.19.0.1/tcp/4001
Swarm listening on /ip4/172.20.0.1/tcp/4001
Swarm listening on /ip4/192.168.100.3/tcp/4001
Swarm listening on /ip4/92.86.209.159/tcp/4001
Swarm listening on /ip6:::1/tcp/4001
API server listening on /ip4/127.0.0.1/tcp/5001
Gateway (readonly) server listening on /ip4/127.0.0.1/tcp/8080
Daemon is ready
18:01:14.952 ERROR floodsub: already have connection to peer: <peer.ID SoLer2> floodsub.go:112
```

Figura G.5: inițializarea daemon-ului IPFS (Go)

Ridicarea procesului daemon IPFS se poate face într-un terminal separat (Figura G.5) folosind comanda `ipfs daemon`. Acesta asigură conexiunea la rețeaua distribuită creată de protocol, expunând o serie de porturi (4001, 5001 și 8080 pentru `swarm`, API și respectiv pentru `Gateway`). Pe ultima linie se poate observa o eroare care nu afectează rularea procesului. Deși nu este imediat observabil, fiecare port aduce o serie de funcționalități protocolului: 4001 asigură conexiunea cu rețeaua și transferul de date, 5001 oferă un punct de conexiune pentru diverse implementări HTTP API iar 8080 poate fi folosit pentru interacțiunea cu fișierele locale în regim “read-only”.

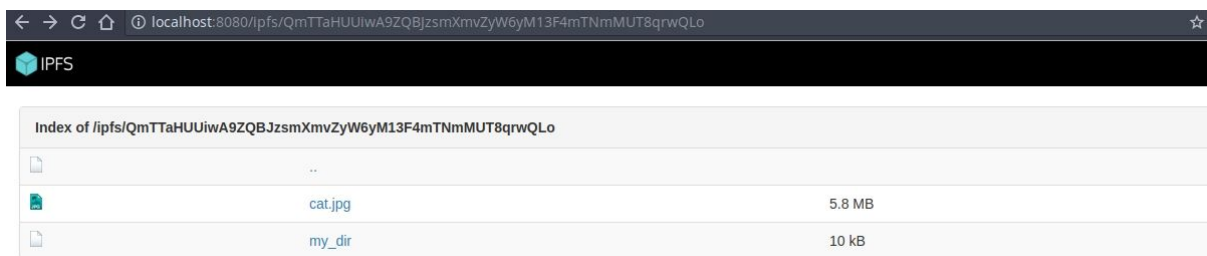


Figura G.6: listarea directoarelor în browser

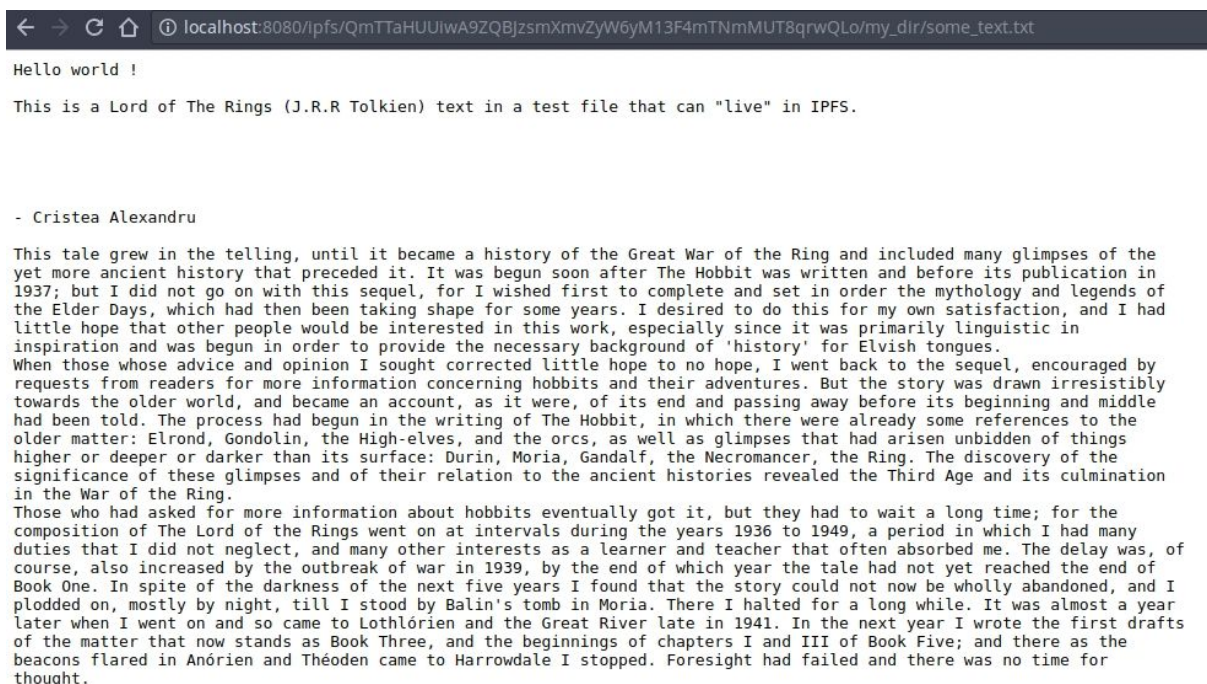


Figura G.7: exemplu de vizualizare a datelor (text) în browser

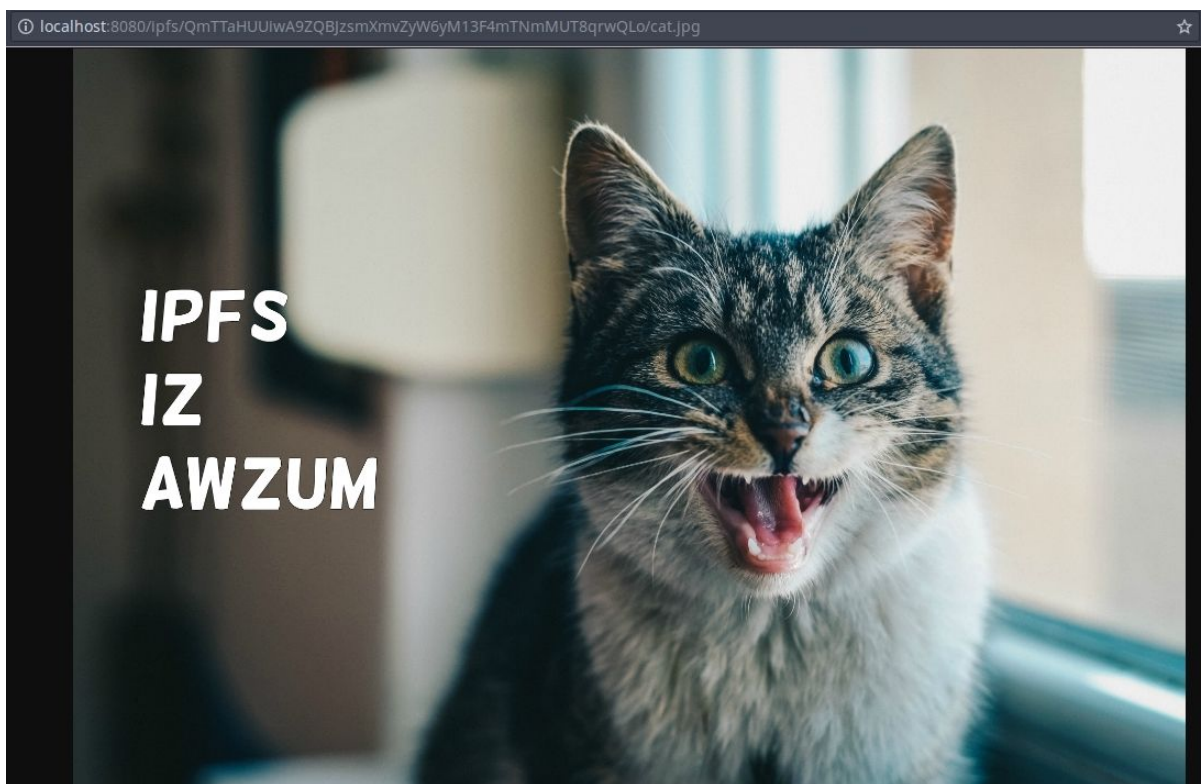


Figura G.8: exemplu de vizualizare a datelor (image) în browser

În (Figura G.6) (Figura G.7) și (Figura G.8) se poate observa utilizarea port-ului de Gateway pentru vizualizarea datelor în browser folosind `http://localhost:8080/ipfs/<hash>`. Acest mecanism funcționează și pentru entități adăugate de alte noduri din cadrul rețelei, atât timp cât daemon-ul este activ. Pentru utilizatorii ce nu doresc să instaleze protocolul și să aibă o interacțiune directă cu toate aceste detalii, comunitatea IPFS ține active servere de gateway la adresa <https://ipfs.io> prin care fișierele adăugate în rețea se pot accesa precum la (Figura G.9), în exemplul nostru rezultând afișarea imaginii de la (Figura G.8).



Figura G.9: link de gateway IPFS oficial


```

xsky@inf anexa2_test $ ipfs object links QmPic9FwaPzBe6wjeJcsC5EcPxpGFFe9cC1zQVCR4o3Hx7f
xsky@inf anexa2_test $ ipfs object links QmW1oDNGfhZsYwQxKmgVJ1CJ2S7jVxfrH8EBJPT1b5XKbJ
QmbkJZSV4NPMThVvU48iWardJmdXHXnF4urXmcNtBCRWpB 262158
QmeTh2XwiF1BLiK3sHw12vFg8wAQ1vMpxnefNLRfmtFxsR 262158
QmXimunM9vHME35f4ExYjVdWCVRMvXgasMhiJaVPFcUPo 262158
QmR8R28EeL36iB8aJqNLeqcJR8BFoctHnnwYmwseay14wX 262158
QmU2JdzNEDcPUnc2NRnm8ZcTuPe4qV3G2AW3vSbCgoFddF 262158
QmdFzYvVj7ECgHrJd2Jkru1cGH6ga3nBjvv4hhKVxpLhgs 262158
QmNoMEfWxewEXL5uhA91N3zwmgaJyBN5pZveAE23W3hiQC 262158
QmYjFip1Y1A77eBhJ8mvzCb88HfDvH3bik92ip8XnPhVrc 262158
QmPyVnVmrQqeevUqYoA6phjyZMsWR261uAzVx5fdbgtFYn 262158
QmS9cNDhpjDk3xWTFhJLEAGY3iwugDnpfdPFMAzfHESLR9 262158
QmeTTuDtn28VvHkhpnofLKwvj7oRLrkZwifxMK4mpqx11 262158
QmbUjL1fiMLz8jDEFbvp1qkdSYRvpFmS8SBbRf7hDTooQT 262158
QmS956Ni4Z5kutuvcaAKJ1VqcQzq7roQhWGRN26R7i7wAD 262158
QmVVG5Y5JH8toevEX1dyr6hZtQFPKupaRXkvJBnSCEfbu 262158
QmaoLhHi5H6FN74Wbcgry3d9iZzQeKY1m2tfnE7bJtmWo 262158
QmXgBH4Wb67cYN6VUSe2Y9Yx5QpEN6w3K9ndMB1tXnaqa 262158
QmY6NvZCZV8vr8S9rPvtDrvWCFskZZWFTWLkQBubJauUHi 262158
QmQNUCETCVR1mcj3qpvt1CrFdUCksxP6rsfFczShnYeA89 262158
QmSL3xd2Eb1yPKKaY6MCmTf4Ft6p7ep46RP9Lr8m813vN 262158
QmCKgS5rp7GsrkME8W82SvBUCtwngwaUKqAM9TJXP773dX 262158
QmS1uierE3KbjoLU9UymH4w8Yxf5fExDpT3YCdVwAvmxYb 262158
QmWq8vynMPZXJo6mgHwPRf34h7iEE5aqsKG4GQEIAbwUpXf 262158
QmRyJQ2Pn7B1NF8U3PMeeu34mE4e32h6fFTAHFm4cbQ8oZ 27493
xsky@inf anexa2_test $

```

Figura G.10: referințe din arborele Merkle pentru obiecte (Go)

Obiectele adăugate în repo-ul local, care trăiesc practic în rețea fiind stocate la fiecare nod, sunt “dezmembrate” în blocuri de date de maxim 256 KB . Fiecare entitate (fișier, director etc.) menține atât datele proprii cât și referințele către copii săi. În exemplul de la (Figura G.10) se poate observa prin utilizarea comenzii `ipfs object links <hash>` cum fișierul text adăugat mai devreme de aproximativ 10 KB este reprezentat doar de un singur bloc (nu se afișează nimic, fișierul este în cadrul unui singur bloc) în timp ce imaginea de aproximativ 6 MB adăugată menține ca și entitate mai multe referințe la blocurile copii din care este alcătuită. Comanda `ipfs object <subcmd>` oferă la modul general o posibilitate de interacțiune cu obiectele IPFS: afișarea datelor identificate de un hash, crearea de noi obiecte și adăugarea de date în interiorul lor etc. Aceeași funcționalitate ca cea descrisă mai sus (afișarea referințelor) este posibilă și cu ajutorul comenzii `ipfs refs`.

Un alt mod de lucru cu datele în cadrul IPFS-ului este prin utilizarea comenzii `ipfs files <subcmd>` ce permite crearea, modificarea și ștergerea entităților de date care sunt interpretate precum într-un sistem de fișiere asemănător `unixfs` . În practică nu am folosit această funcționalitate deoarece interacțiunea cu obiectele IPFS identificate prin hash și vizualizarea acestora ca și noduri în arborele Merkle a fost mai relevantă înțelegerii conceptelor protocolului.

Ca și funcționalitate experimentală, există comanda `ipfs dag <subcmd>` ce permite lucrul cu nodurile Merkle DAG-ului. Momentan aceasta permite afișarea informațiilor dintr-un nod cât și referințele acestuia și introducerea unui nou nod.

O altă observație ce merită specificată este **analiza propriu-zisă a rețelei**, lucru ce poate fi realizat cu ajutorul unui set de comenzi precum: `ipfs id` (Figura G.11), `ipfs swarm <subcmd>` (Figura G.12), `ipfs bootstrap` (Figura G.13), `ipfs dht <subcmd>` (Figura G.14), `ipfs ping` sau `ipfs diag`. Deși nu este evident în primă fază, odată ridicat

daemon-ul IPFS, acesta folosește diverse concepte și mecanisme pentru descoperirea de noi noduri: **Bootstrap-List** “hardcoded” pentru conectarea la un set de noduri “de încredere” care lasă posibilitatea de creare de conexiuni aferente și cu alte noduri, **mdNS** ca și tehnologie de localizare a nodurilor prin difuzarea mesajelor de descoperire locală, **Random Walk** pentru traversarea aleatorie a rețelei și conectarea la nodurile întâlnite pe parcurs.

```
xsky@inf my_data_root $ ipfs id
{
  "ID": "QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi",
  "PublicKey": "CAASpgIwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCxXYqJ4aHxHB4rRchTn+R63CcXx
44Q/YXLMnddGct6VM75vVoMQU0orwRE9EtYNy0xgPHvm959tT5xHsc3V5XhBaJvBiNhb1ohx4cJyhCR1e6fLtlWuPbHwNs0DSBy
FXXXx+m4tW5s5uptGsy9m8BSLUkEuVGB0gz2LyTDJaYYzLoUvIYx1HnrUPA j7vY+v7N170w0Ao jeb4aYZ17oFbAff150KKnycp6
Ry/1Z0Zgj7/%JS10SuhsupLY2kZry54izB5rZwwAtaKJBa9y8GI f8uQa6m3GHamCf8m2C9dUUw3176tv9ABpYMadSPFM1HNLKbp
c1g7msPXW9i1ih7KddAgMBARE=",
  "Addresses": [
    "/ip4/127.0.0.1/tcp/4001/ipfs/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi",
    "/ip4/192.168.100.3/tcp/4001/ipfs/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi",
    "/ip4/172.17.0.1/tcp/4001/ipfs/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi",
    "/ip4/172.19.0.1/tcp/4001/ipfs/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi",
    "/ip4/172.18.0.1/tcp/4001/ipfs/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi",
    "/ip4/172.20.0.1/tcp/4001/ipfs/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi",
    "/ip6:::1/tcp/4001/ipfs/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi",
    "/ip4/92.86.209.159/tcp/55187/ipfs/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi",
    "/ip4/92.86.209.159/tcp/55187/ipfs/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et9ZyQi"
  ],
  "AgentVersion": "go-ipfs/0.4.9-dev/1c50ec0c",
  "ProtocolVersion": "ipfs/0.1.0"
}
xsky@inf my_data_root $
```

Figura G.11: identificarea nodului curent (Go)

Pentru afișarea informațiilor nodului ce rulează la mașina curentă se poate folosi comanda `ipfs id` (Figura G.11) care afișează id-ul nodului, cheia publică, adresele de identificare, versiunea agentului și a protocolului. În funcție de regenerare, aceste date se pot schimba.

```
xsky@inf my_data_root $ ipfs swarm peers | head -n 10
/ip4/101.160.40.195/tcp/4001/ipfs/QmWtt7RdkSQoMbmnc6UA28RDFAXDcagHqoscbUu9SSnw8k
/ip4/101.50.1.236/tcp/4001/ipfs/QmaPxBifji8QA3taH7msbdcDnXn4wQa92yrKe4VB5iomH
/ip4/101.78.229.4/tcp/4001/ipfs/QmPn8tnUvTqvTKZBtume649GqCrjbg1ukLZwDyWjmlwioC
/ip4/103.18.139.230/tcp/53722/ipfs/QmRm1hBYtkTJxbVMDbou3RCxhRUHRan68nviUypVWzBETg
/ip4/104.130.138.117/tcp/4001/ipfs/QmSrmMHXNRpMcWu6vCXg8Q9aJtFgxQhcqETZjmf4RRmkD1
/ip4/104.131.131.82/tcp/4001/ipfs/QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbUtfsmvsqQLuvuJ
/ip4/104.131.20.170/tcp/4001/ipfs/QmQ68e6xjqbwK7BCURayW7X6YahbCognPuAeqsVpnnVifS
/ip4/104.133.2.68/tcp/34830/ipfs/QmTAmvzNBsicnaJpLTUnVqcPankP3pNDogHpAtUNKK2rU7
/ip4/104.153.108.98/tcp/4001/ipfs/QmWPKUoiPBPDCvmCsUZDLzw9nLyCUUp51YV6x4MJU7hNP
/ip4/104.154.113.126/tcp/4001/ipfs/QmTeTpJm6LhaUYN6jpVi3zYBdzFjUMQvywpZrA9wRNrtxe
xsky@inf my_data_root $ ipfs swarm peers | wc -l
667
xsky@inf my_data_root $
```

Figura G.12: numărul de noduri & parte din lista de noduri cu care există o conexiune activă (Go)

Listarea nodurilor cu care există o conexiune activă se poate face precum în (Figura G.12). Comanda `ipfs swarm <subcmd>` permite de asemenea conectarea / deconectarea manuală la / de la un nod specific atât timp cât îi este cunoscut adresa determinată de `multiaddr`, și atât timp cât este posibilă realizarea unei astfel de conexiuni. Acest format

este vizibil în figura mai sus menționată, adresele conținând protocolul de rețea, adresa acestuia, port-ul, protocolul de comunicare și identificatorul unic.

```
xsky@inf my_data_root $ ipfs bootstrap list
/ip4/104.131.131.82/tcp/4001/ipfs/QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbUtfsmvsgQLUvuj
/ip4/104.236.151.122/tcp/4001/ipfs/QmSoLju6m7xTh3DuokvT3886QRYqxAbz1kShaanJgW36yx
/ip4/104.236.176.52/tcp/4001/ipfs/QmSoLnS6ccFuZQJzRadHn95W2CrSFmZuTdDWP8HxaHca9z
/ip4/104.236.179.241/tcp/4001/ipfs/QmSoLppuBtQSGwKDZT2M73ULpJvfd3a26ha4oFGL1KrGM
/ip4/104.236.76.40/tcp/4001/ipfs/QmSoLV4Bbm51jM9C4gDY2Q9Cy3U6aXMDAbzgu2fzaDs64
/ip4/128.199.219.111/tcp/4001/ipfs/QmSoLSafTMBsPKadTEgaXctDQVcqN88CNLHXMkTNwMKPnu
/ip4/162.243.248.213/tcp/4001/ipfs/QmSoLueR4xBelUy9WZ9xGUUxunbKwcrNFTDAadQJmocrNm
/ip4/178.62.158.247/tcp/4001/ipfs/QmSoLer265NRgSp2LA3dPaeykiS1J6DifTC88f5uVQKNAd
/ip4/178.62.61.185/tcp/4001/ipfs/QmSoLMeWqB7YGVLJN3pNLQpmMEk35v6wYtsMGLzSr5QBU3
/ip6/2400:6180:0:d0::151:6001/tcp/4001/ipfs/QmSoLSafTMBsPKadTEgaXctDQVcqN88CNLHXMkTNwMKPnu
/ip6/2604:a880:0:1010::23:d001/tcp/4001/ipfs/QmSoLueR4xBelUy9WZ9xGUUxunbKwcrNFTDAadQJmocrNm
/ip6/2604:a880:1:20::1d9:6001/tcp/4001/ipfs/QmSoLju6m7xTh3DuokvT3886QRYqxAbz1kShaanJgW36yx
/ip6/2604:a880:1:20::1f9:9001/tcp/4001/ipfs/QmSoLnS6ccFuZQJzRadHn95W2CrSFmZuTdDWP8HxaHca9z
/ip6/2604:a880:1:20::203:d001/tcp/4001/ipfs/QmSoLppuBtQSGwKDZT2M73ULpJvfd3a26ha4oFGL1KrGM
/ip6/2604:a880:800:10::4a:5001/tcp/4001/ipfs/QmSoLV4Bbm51jM9C4gDY2Q9Cy3U6aXMDAbzgu2fzaDs64
/ip6/2a03:b0c0:0:1010::23:1001/tcp/4001/ipfs/QmSoLer265NRgSp2LA3dPaeykiS1J6DifTC88f5uVQKNAd
/ip6/2a03:b0c0:1:d0::e7:1/tcp/4001/ipfs/QmSoLMeWqB7YGVLJN3pNLQpmMEk35v6wYtsMGLzSr5QBU3
xsky@inf my_data_root $
```

Figura G.13: lista de noduri bootstrap (Go)

Lista de “noduri de încredere” Bootstrap-List poate fi vizualizată și modificată utilizând comanda `ipfs bootstrap <subcmd>` (Figura G.13). Aceste noduri sunt folosite de către daemon la inițializare pentru conectare și descoperirea de alte noduri.

```
xsky@inf my_data_root $ time ipfs dht findprovs QmTTaHUUiwA9ZQBjzsmXmvZyW6yM13F4mTHmMUT8qrwQLo
QmNo7cDPUY29mZvpDN2LH2xo3FKAc6E9TzFhCY6et9ZyQi

real    0m10.700s
user    0m0.133s
sys     0m0.013s
xsky@inf my_data_root $
```

Figura G.14: găsirea nodurilor ce pot oferi o anumită entitate de date (Go)

Lucrul direct cu tabelele hash distribuite se poate realiza utilizând comanda `ipfs dht <subcmd>` (Figura G.14) ce permite găsirea nodurilor care pot oferi o anumită entitate de date, găsirea unui nod dar și permiterea înregistrării în DHT a unei perechi (`key`, `value`) ce respectă anumite condiții. Caracteristica ultim menționată este folosită la un nivel mai înalt de spațiul de nume IPNS pentru mutabilitate.

Problema imutabilității a venit pe parcursul cercetării sub forma găsirii aceleiași entități de date ce se poate modifica în timp. Acest aspect este rezolvat de spațiul de nume **IPNS** care permite publicarea unui set de date folosind `/ipns/<NodeId>` (practic, identificatorul de nod al mașinii curente). Un exemplu relevant se regăsește în (Figura G.15) prin utilizarea comenzii `ipfs name <subcmd>`.


```

xsky@inf anexa2_test $ time ipfs name publish QmTTaHUUiuA9ZQBjZsmXmvZyW6yM13F4mTNmMUT8qrwQLo
Published to QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et92yQi: /ipfs/QmTTaHUUiuA9ZQBjZsmXmvZyW6yM13F4mTNmMUT8qrwQLo

real    0m22.696s
user    0m0.030s
sys     0m0.013s
xsky@inf anexa2_test $ ipfs ls /ipns/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et92yQi
QmW1oDNGfhZsYwQxKmgVJ1CJ2S7jVxfrH8EBJPT1bSXKbJ 5796082 cat.jpg
QmTdjuXPLt62J6NGKFs2kCgDP5qzRAka1KNCoib2tuUbgP 10210 my_dir/
xsky@inf anexa2_test $ ipfs cat /ipns/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et92yQi/my_dir/some_text.txt | head -n 4
Hello world !

This is a Lord of The Rings (J.R.R Tolkien) text in a test file that can "live" in IPFS.

xsky@inf anexa2_test $ nano my_data_root/my_dir/some_text.txt
xsky@inf anexa2_test $ cat my_data_root/my_dir/some_text.txt | head -n 4
Hello world !

This is a Lord of The Rings (J.R.R Tolkien) text in a test file that can "live" in IPFS.
edit 1 !!!!!
xsky@inf anexa2_test $ ipfs add -r my_data_root/
added QmW1oDNGfhZsYwQxKmgVJ1CJ2S7jVxfrH8EBJPT1bSXKbJ my_data_root/cat.jpg
added QmZotsPZeKxvk6Dp9fntRitaMhCZHT2Y2zm3H5fDCDaHXC my_data_root/my_dir/some_text.txt
added QmTR7yMEjm9LtdJf8U4QuR9jxM4ptcZsaSKEudnqCkFDmW my_data_root/my_dir
added QmYDXLqJJzE5o5rirwhKaEHRtdDsE7BkxpNk846AEyx1ju my_data_root
xsky@inf anexa2_test $ ipfs name publish QmYDXLqJJzE5o5rirwhKaEHRtdDsE7BkxpNk846AEyx1ju
Published to QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et92yQi: /ipfs/QmYDXLqJJzE5o5rirwhKaEHRtdDsE7BkxpNk846AEyx1ju
xsky@inf anexa2_test $ ipfs cat /ipns/QmNo7cDPUY29mZvpDN2LH2Xo3FKAc6E9TzFhCY6et92yQi/my_dir/some_text.txt | head -n 4
Hello world !

This is a Lord of The Rings (J.R.R Tolkien) text in a test file that can "live" in IPFS.
edit 1 !!!!!
xsky@inf anexa2_test $ █

```

Figura G.15: utilizarea IPNS-ului

În primă fază am publicat întregul director peste propriul identificator de nod și am utilizat listarea și afișarea de date direct peste spațiul de nume `/ipns/QmNo7...`. După adăugarea unei noi linii de text în fișierul “some_text.txt”, a fost realizată readăugarea întregului director (a se observa modificarea referințelor fișierului, directorului părinte și a celui rădăcină) și republicarea acestuia în IPNS. Rezultatul este modificarea datelor și persistența identificatorului spațiului de nume. Vizualizarea datelor este posibilă și în browser (Figura G.16).

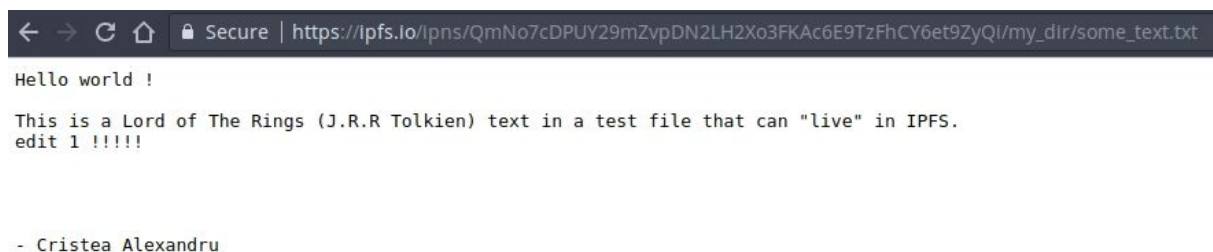


Figura G.16: vizualizarea datelor folosind IPNS în browser

Experimentele descrise până acum au scos în evidență funcționalitățile principale ale protocolului implementat în Go ce pot fi utilizate de către dezvoltatori. O altă comandă ce merită specificată este `ipfs config` care permite realizarea de diverse configurări locale (adrese, permisiuni de conectare etc.). Pe lângă aceasta mai există și comenzile `ipfs mount`, `ipfs resolve`, `ipfs dns`, `ipfs pin`, `ipfs repo`, `ipfs stats`, `ipfs filestore`, `ipfs version`, `ipfs update`, `ipfs commands`, fiecare dintre ele regăsindu-și utilitatea în funcție de caz.

Lucrul offline în lipsa de conexiune cu rețeaua globală de Internet a fost următorul pas ce a fost necesar dovezii că IPFS este un protocol distribuit. Am realizat o serie de experimente cu ajutorul motorului de virtualizare bazat pe containere **Docker**. După traversarea unui set de cunoștințe de bază ale acestei tehnologii, am conceput și construit imagini ce au putut fi instanțiate în containere separate (Figura G.17) (Figura G.18). Utilizând varianta Go o serie de containere au fost ridicate și puse sub test pentru dovedirea descoperirii locale.

```
FROM ubuntu
MAINTAINER xSkyripper <alexandru.cristea604@gmail.com>

# Build argument for the URL source of the .tar.gz binaries
ARG ipfs_url_src

# Expose the ports needed by IPFS
EXPOSE 4001
EXPOSE 4002/udp
EXPOSE 5001
EXPOSE 8080

# Set the TERM and the WORKDIR
ENV TERM=xterm

WORKDIR /ipfs
VOLUME /ipfs/ipfs_data

# Install utilities
RUN apt-get update && \
    apt-get install -y apt-utils \
    nano wget inetutils-ping

# Get the archive and install IPFS
RUN wget $ipfs_url_src
RUN tar fxvz go-ipfs*
RUN rm go-ipfs*.gz
RUN cd go-ipfs && bash install.sh && cd /ipfs && rm -rf go-ipfs

# Colorize prompt
RUN sed s/#force_color_prompt=yes/force_color_prompt=yes/ ~/.bashrc > ~/.bashrc2
RUN rm ~/.bashrc
RUN mv ~/.bashrc2 ~/.bashrc

# Custom made bash script for ENTRYPOINT
COPY ./ipfs_start.sh ./ipfs_start.sh

# One entrypoint to rule them all
ENTRYPOINT ["/bin/bash", "ipfs_start.sh"]
```

Figura G.17: imagine Docker cu IPFS și diverse utilitare

```
xsky@inf IPFS $ docker build -t xskyripper/docker-ipfs:latest --build-arg \
> ipfs_url_src=https://dist.ipfs.io/go-ipfs/v0.4.9/go-ipfs_v0.4.9_linux-amd64.tar.gz -f \
> $(pwd)/docker-ipfs/Dockerfile .
Sending build context to Docker daemon 4.189MB
Step 1/20 : FROM ubuntu
----> 0ef2e08ed3fa
Step 2/20 : MAINTAINER xSkyripper <alexandru.cristea604@gmail.com>
----> Using cache
----> 4cbf1bb8d5a7
Step 3/20 : ARG ipfs_url_src
----> Using cache
----> b38e4df08f6f
Step 4/20 : EXPOSE 4001
----> Using cache
----> 090a470bfc81
```

Figura G.18: construirea imaginii Docker

După construirea cu succes a acestei imagini Docker, am putut instanția containere ce rulează Go-IPFS în cadrul lor în mod independent. Ca și constrângere am utilizat o rețea internă Docker ce nu permite containerelor să comunice cu “exteriorul” ci doar între ele.

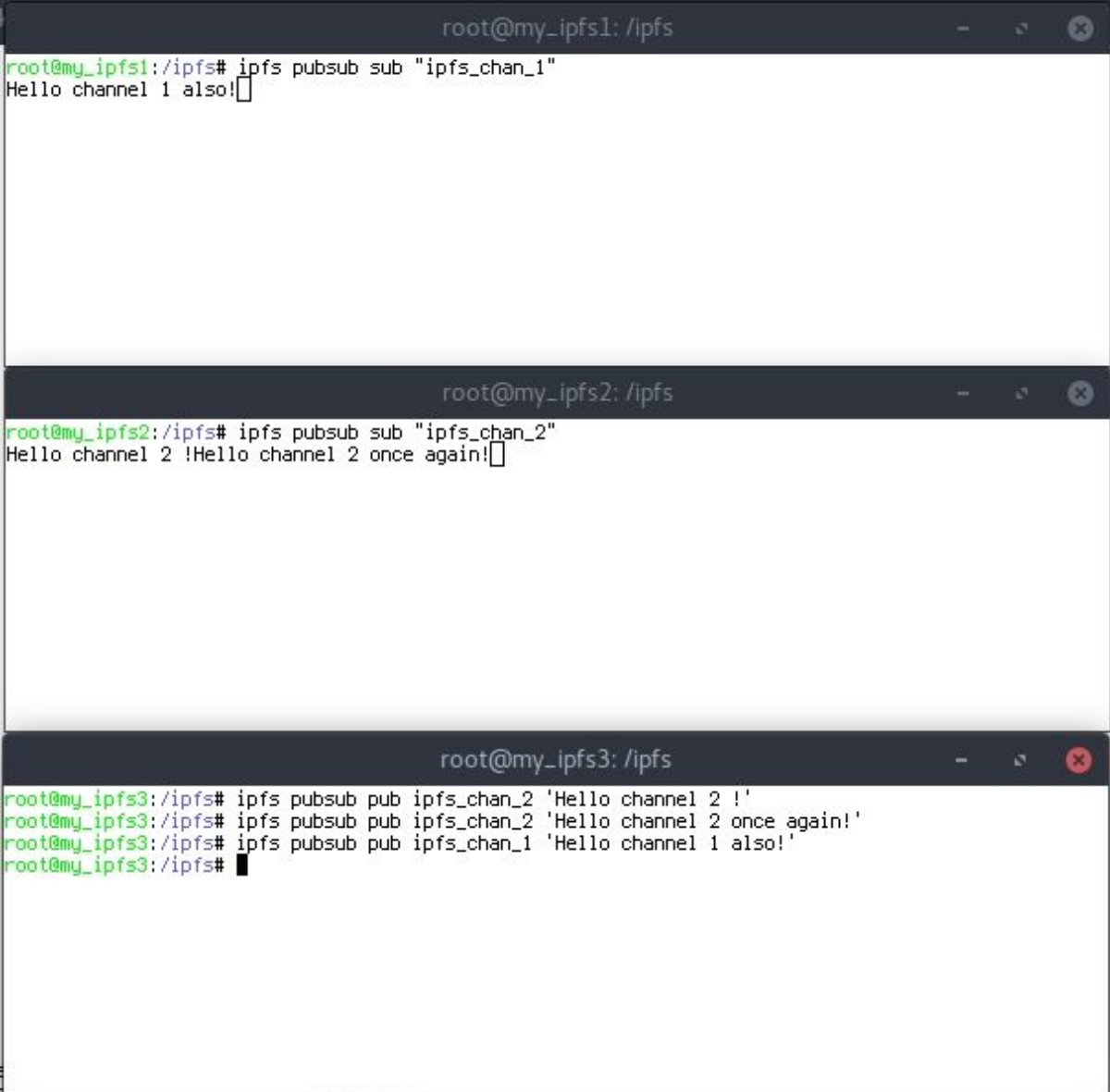
```
docker run -d -h <host> --name <name> \
--network <network> xskyripper/docker-ipfs:latest
```

The image shows three terminal windows stacked vertically, each representing a different Docker container running IPFS.

- Top window (root@my_ipfs1: /ipfs):** Shows the command `ipfs swarm peers` and its output, listing two peers. It then runs `ipfs ping` on one of the peers, showing successful pings with response times around 0.34 ms.
- Middle window (root@my_ipfs2: /ipfs):** Shows the command `ipfs add hello.txt` which successfully adds the file. Then it runs `ip addr` showing the loopback interface `lo` at `127.0.0.1` and the ethernet interface `eth0` at `172.19.0.3`.
- Bottom window (root@my_ipfs3: /ipfs):** Shows the command `ipfs cat` which outputs the content of the added file, "Hello, world IPFS ! My inner Docker container IPv4 is 172.19.0.3". It then runs `ping 8.8.8.8`, showing a successful ping to the external IP address.

Figura G.19: diferite cazuri folosind Go-IPFS și Docker

În (Figura G.19) se pot observa mai multe aspecte ale utilizării protocolului într-o rețea locală precum cea de tip “internal” a Docker-ului. În primul rând în “my_ipfs3” se poate analiza lipsa conexiunii prin executarea unui `ping 8.8.8.8` (server Google DNS ...) și “înghețul” acestuia. Apoi putem observa în “my_ipfs1” faptul ca există o conexiune cu celelalte 2 noduri “my_ipfs2” și “my_ipfs3”, același lucru fiind valabil și pentru cele din urmă prin reciprocitate. Primul nod execută un `ipfs ping` către unul dintre nodurile vecine și se pot observa timpii de răspuns. În al doilea rând, “my_ipfs2” adaugă un fișier la rețea ce conține IPv4-ul său, “my_ipfs3” vizualizându-l.



```
root@my_ipfs1: /ipfs
root@my_ipfs1:/ipfs# ipfs pubsub sub "ipfs_chan_1"
Hello channel 1 also!

root@my_ipfs2: /ipfs
root@my_ipfs2:/ipfs# ipfs pubsub sub "ipfs_chan_2"
Hello channel 2 !Hello channel 2 once again!

root@my_ipfs3: /ipfs
root@my_ipfs3:/ipfs# ipfs pubsub pub ipfs_chan_2 'Hello channel 2 !'
root@my_ipfs3:/ipfs# ipfs pubsub pub ipfs_chan_2 'Hello channel 2 once again!'
root@my_ipfs3:/ipfs# ipfs pubsub pub ipfs_chan_1 'Hello channel 1 also!'
root@my_ipfs3:/ipfs#
```

Figura G.20: utilizarea Go-IPFS pubsub în Docker

Dat fiind contextul problemei de a trimite mesaje în regim “real-time” din cauza diverselor necesități (ex. sincronizări) ce au survenit pe parcurs, conceptul **PubSub** a fost descoperit cu o ușoară dificultate deoarece nu se află încă în manualul interfeței CLI și

momentan este un modul experimental. Utilizarea și analiza aspectelor asupra acestuia a dus direcția de implementare pe un drum mai ușor de traversat spre realizarea unei aplicații. Acest concept oferă o posibilitate de trimitere a unor mesaje pseudo-direcționate spre un anumit canal specific. (Figura G.20)

Apariția conceptuală a aplicației H.O.L.D.I.N. m-a determinat să regândesc diferitele funcționalități ale implementărilor curente (Go [15] și Javascript). Aceasta necesita un mediu mobil (ex. device Android), posibilitatea folosirii geolocației și rularea în background. Atenția mea a căzut pe platformele de dezvoltare mobilă precum Cordova care ofereau un mediu de dezvoltare ușoară a aplicațiilor web (folosind Javascript, HTML și CSS) care pot rula pe diverse device-uri. În acest timp din diversele tutoriale create de comunitatea IPFS pe care le urmăream am observat faptul că varianta de Go nu respectă interfața generică [5]. Împreună cu ideea de rulare a unui proces “shell”, atenția a căzut asupra variantei Javascript ce părea perfectă în implementarea aplicației.

```
const IPFS = require('ipfs')
...
const repoPath = String(Math.random())
const ipfs = new IPFS({
  repo: repoPath,
  init: true,
  start: true,
  EXPERIMENTAL: {
    pubsub: true
  }
});
...
setInterval(() => {
  ipfs.swarm.peers(function(err, peerInfos) {
    if (err) {
      throw err
    }
    console.log('IPFS: ID: ' + nodeID +
'\n IPFS: Swarm Peers: ' + new Date() + '\n')
    peerInfos.forEach((val, key) => {
      console.log(val.addr.toString() +
'/ipfs/' + val.peer.id._idB58String)
    })
    console.log('\n\n')
  })
}, 3000)
```

Figura G.21: inițializare și afișare swarm (Javascript)

După scrierea unor scurte secvențe de cod de test atât pentru browser cât și pentru mediul Node.s (Figura G.21) am observat că discovery-ul era practic inexistent. Lipsa mDNS-ului și a rutării Kademia forța implementările să utilizeze un server WebStar aflat la distanță pentru semnalizare. Acesta oferea posibilitatea descoperirii altor noduri. Cum contextul problemei și ideile aplicației fuseseră stabilite în mediul offline, această variantă de implementare nu respecta condițiile (cel puțin momentan, rutarea și “relay-ul” fiind prevăzute de către comunitate în următoarele actualizări ale protocolului implementat în Javascript).

Problema a fost rezolvată prin luarea deciziei de a folosi varianta de Go împreună cu API-ul Javascript al protocolului. Pentru instanțierea daemon-ului Go a fost nevoie de implementarea unui plugin Cordova folosind Java și Javascript ce permite ridicarea acestuia și menținerea sa în viață într-un proces “shell” și thread separat de cel al aplicației (Figura G.22) dar și configurarea sa astfel încât să poată permite conexiunea unui API.

```
...
final Runnable ipfsDaemonThread = new Runnable() {
    @Override
    public void run() {
        ...
        try {
            ...
            ipfsDaemonProcess = Runtime.getRuntime().exec(
                new String[]{ipfsBinPath, "daemon", "--enable-pubsub-experiment"},
                new String[]{"IPFS_PATH=" + ipfsRepo}
            );
            ...
        }
    }
}
...
ipfsDaemonThreadFuture= cordova.getThreadPool().submit(ipfsDaemonThread);
...
```

Figura G.22: fragment din codul plugin-ului cordova-plugin-ipfs (Java)

Astfel, prin parcurgerea diferitelor cazuri de utilizare ale protocolului la nivel general dar și în funcție de fiecare implementare (protocol sau API) am tras o serie de concluzii și am analizat diverse detalii. Am găsit o soluție pentru realizarea cu succes a implementării sistemului H.O.L.D.I.N. ce a necesitat un drum lung de cercetare.

Protocolul IPFS în implementare, cel puțin momentan, nu este perfect. Dar, așa cum a făcut cunoscut Voltaire acest lucru⁴⁸, “*perfectul este inamicul bunului*”. Nimic nu poate fi perfect și stă în datoria noastră să acceptăm acest aspect și să nu ne oprim din evoluție din cauza piedicilor ce apar pe orice drum am merge.

⁴⁸ "Perfect is the enemy of good - Wikipedia."
https://en.wikipedia.org/wiki/Perfect_is_the_enemy_of_good