

# **Technology Know-How: IPFS**

## **Final Report**

Michele Mongardi

`michele.mongardi2@studio.unibo.it`

Agosto 2022

# Contents

<b>Introduzione</b>	<b>5</b>
<b>1 IPFS 101</b>	<b>6</b>
1.1 IPFS: che cos'è . . . . .	6
1.2 Content IDentifier . . . . .	6
1.3 Disponibilità . . . . .	7
1.4 Decentralizzazione . . . . .	8
1.5 Nodi . . . . .	8
<b>2 Architettura</b>	<b>9</b>
2.1 Content-addressing . . . . .	9
2.1.1 Hashing . . . . .	9
2.1.2 Multihash . . . . .	12
2.1.3 Identifier Formats . . . . .	14
2.2 Directed Acyclic Graphs (DAGs) . . . . .	15
2.3 Distributed Hash Tables (DHTs) . . . . .	17
2.4 Kademlia . . . . .	19
2.4.1 Routing Tables . . . . .	20
2.4.2 Lookup algorithm . . . . .	22
2.4.3 Peer irraggiungibili . . . . .	23
2.5 Libp2p e Bitswap . . . . .	23
2.5.1 Libp2p . . . . .	24
2.5.2 Bitswap . . . . .	25
<b>3 Install IPFS</b>	<b>29</b>
3.1 IPFS Desktop . . . . .	29
3.1.1 Windows Installation . . . . .	30
3.1.2 macOS Installation . . . . .	30
3.1.3 Ubuntu Installation . . . . .	30
3.2 Command-line . . . . .	31
3.2.1 Windows Installation . . . . .	31
3.2.2 macOS Installation . . . . .	32
3.2.3 Linux Installation . . . . .	32
3.3 IPFS Companion . . . . .	32
3.4 Infrastruttura Server . . . . .	34
3.4.1 Installation . . . . .	34
<b>4 Basics</b>	<b>35</b>
4.1 Desktop app . . . . .	35
4.1.1 Aggiungere un file . . . . .	35
4.1.2 Scaricare un file . . . . .	36
4.1.3 Condividere un file . . . . .	37

4.1.4	Rimuovere un file . . . . .	38
4.1.5	Single e multi-page website su IPFS . . . . .	40
4.2	CLI Operations . . . . .	41
4.2.1	Aggiungere un file . . . . .	41
4.2.2	Recuperare un file . . . . .	42
4.2.3	Visualizzare un file . . . . .	42
4.2.4	Bloccare un file . . . . .	43
4.2.5	Rimuovere un file . . . . .	43
<b>5</b>	<b>Java-ipfs-HTTP-client</b>	<b>45</b>
5.1	Spring . . . . .	45
5.1.1	Spring Initializr . . . . .	46
5.2	Struttura del progetto . . . . .	47
5.2.1	Main Class . . . . .	47
5.2.2	IPFSConfig Class . . . . .	47
5.2.3	IPFSService Class . . . . .	48
5.2.4	IPFSController Class . . . . .	49
5.3	Postman . . . . .	51
5.3.1	I servizi POST e GET . . . . .	52
<b>6</b>	<b>Testing</b>	<b>54</b>
6.1	Metodologia di misurazione . . . . .	54
6.2	Peer irraggiungibili . . . . .	55
6.3	K-bucket incompleti . . . . .	56
6.4	I peer più vicini . . . . .	57
6.5	Considerazioni sui dati . . . . .	58
<b>7</b>	<b>Progetti correlati</b>	<b>59</b>
7.1	IPDL . . . . .	59
7.2	Libp2p . . . . .	59
7.3	IPFS Cluster . . . . .	59
7.4	DNSLink . . . . .	60
7.5	Multiformats . . . . .	60
7.6	ProtoSchool . . . . .	60
<b>8</b>	<b>Conclusioni</b>	<b>61</b>
8.1	Punti di forza e di debolezza della tecnologia . . . . .	61
8.2	Parere personale sulla tecnologia . . . . .	62

## List of Figures

1	Esempio di puntatore . . . . .	7
2	Una funzione crittografica di hash al lavoro . . . . .	10
3	Tabella Multibase . . . . .	11
4	Multihash Format . . . . .	12
5	Una porzione della tabella Multicodec . . . . .	13
6	Un esempio di multihash . . . . .	14
7	Esempio di CIDv1 . . . . .	15
8	Esempio sezioni ridondanti DAG . . . . .	17
9	Hash table . . . . .	17
10	Esempio di una DHT . . . . .	18
11	Bitswap Subsystems . . . . .	27
12	IPFS Destkop . . . . .	29
13	Command-line . . . . .	31
14	IPFS Companion . . . . .	32
15	IPFS Web UI dashboard . . . . .	34
16	Controllo connessione a IPFS . . . . .	35
17	Aggiunta di un file sul proprio nodo . . . . .	36
18	Finestra <i>Import From IPFS</i> . . . . .	36
19	Download di un file da IPFS . . . . .	37
20	Condivisione di un file dal proprio nodo . . . . .	38
21	Rimozione di un file dal proprio nodo . . . . .	38
22	Conferma rimozione risorsa . . . . .	39
23	Esecuzione manuale del Garbage Collector . . . . .	39
24	Importazione di <code>index.html</code> . . . . .	40
25	Recupero URL per visualizzazione tramite Browser . . . . .	40
26	Connessione del nodo IPFS alla rete . . . . .	41
27	Aggiunta di <code>hello-ipfs.txt</code> su IPFS tramite CLI . . . . .	41
28	Recupero di una cartella tramite CLI . . . . .	42
29	Visualizzazione del contenuto di un file tramite CLI . . . . .	42
30	<i>Pinning</i> di una directory tramite CLI . . . . .	43
31	<i>Unpinning</i> di un file tramite CLI per rimozione tramite <i>GC</i> . . . . .	43
32	Servizio <code>POST</code> . . . . .	52
33	Identificatore del file caricato sulla rete IPFS . . . . .	52
34	Servizio <code>GET</code> . . . . .	53
35	Contenuto del file recuperato dalla rete . . . . .	53
36	Visualizzazione/Salvataggio del file recuperato dalla rete IPFS . . . . .	53
37	Peer irraggiungibili all'interno dei bucket . . . . .	55
38	Peer mancanti per bucket . . . . .	56
39	Numero di peer conosciuti tra i 20 più vicini . . . . .	57

## Introduzione

L'**Interplanetary File System**, conosciuto anche come **IPFS**, è un *file system distribuito* che cerca di connettere tutti i dispositivi informatici sotto lo stesso sistema di file.

Se un file system indica un meccanismo con il quale i file sono organizzati su dispositivi informatici utilizzati per l'archiviazione dei dati [1], un file system distribuito è un particolare file system che permette la memorizzazione di risorse su dispositivi di archiviazione distribuiti in una rete informatica.

L'obiettivo perseguito da questa tecnologia è quello di creare un network di portata globale che consenta l'archiviazione delle informazioni in maniera completamente decentralizzata, con elevata scalabilità e con grande resistenza alla censura dell'informazione.

IPFS è quindi un sistema di storage distribuito peer-to-peer (*p2p*): l'accesso al contenuto dei file è possibile grazie ai **peer**, nodi che possono essere localizzati in una qualsiasi parte del globo il cui ruolo è quello di trasmettere e/o memorizzare dati.

Il protocollo sotto esame si compone di tre aspetti fondamentali sui quali soffermarsi:

- L'identificazione univoca dei file grazie al **content-addressing**;
- Il collegamento dei contenuti tramite grafici aciclici diretti (**DAG**);
- La scoperta dei contenuti tramite tabelle hash distribuite (**DHT**).

# 1 IPFS 101

## 1.1 IPFS: che cos'è

IPFS può essere descritto in una frase come “un sistema distribuito per l’archiviazione e l’accesso a file, dati, siti web e applicazioni”. Per comprendere al meglio questa definizione, si supponga che si vogliano ricercare informazioni relative al **Distributed Computing** [2].

In primo luogo, si potrebbe visitare la relativa pagina Wikipedia al seguente indirizzo:

```
https://en.wikipedia.org/wiki/Distributed\_computing
```

Inserendo questo *URL* nella barra degli indirizzi del Browser, viene richiesta ad uno dei calcolatori di Wikipedia (che potrebbe risiedere in qualsiasi parte del pianeta) la pagina relativa al Distributed Computing. Tuttavia, questa non rappresenta l’unica possibilità di recuperare le informazioni di cui si ha bisogno.

Memorizzato su IPFS, infatti, vi è un *mirror* di Wikipedia, ovvero una copia esatta del suo insieme di dati, che può essere utilizzato rispetto al classico metodo di ricerca. Se si fa uso di IPFS, la pagina d’interesse verrà recuperata in questo modo:

```
https://ipfs.io/ipfs/QmXoybizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Distributed\_computing.html
```

## 1.2 Content Identifier

IPFS permette di recuperare le informazioni relative al Distributed Computing tramite il **loro contenuto** (content-addressing), e non in base alla loro posizione. La versione IPFS di queste informazioni è rappresentata dalla stringa di numeri che si trova al centro dell’indirizzo sopracitato. Per comprenderne il funzionamento si introduce il concetto di **CID**.

L’insieme dei caratteri che si trovano dopo */ipfs/* prende il nome di **content identifier**, detto anche **CID**, tramite il quale IPFS può recuperare il contenuto di un documento da luoghi differenti. Di norma un *URL* “classico” identifica un file in base a dove si trova, ovvero alla posizione in cui è allocato in un dispositivo (per esempio, *C:/Users/Michele/Documents/SD.docx*). Tale concetto però non funziona se il file è presente in più luoghi differenti.

IPFS, come già anticipato, non è basato sulla posizione fisica del file, bensì indirizza il documento in base al suo contenuto. Il CID è ottenuto da una **funzione hash crittografica** del contenuto di quell’indirizzo. L’hash è univoco rispetto al contenuto da cui viene generato. Esso permette di verificare che quanto ricevuto sia ciò che è stato effettivamente richiesto.

Un indirizzo IPFS può fare riferimento ai metadata di una singola parte di un file, ad un intero documento, ad una directory, ad un sito web nella sua interezza o a qualsiasi

altro tipo di contenuto.

Siccome l'indirizzo di un file IPFS è generato dal contenuto del documento stesso, i link IPFS non possono essere modificati. Se, per esempio, il testo di una pagina web venisse cambiato, la nuova versione generata otterrebbe un nuovo, diverso, indirizzo IPFS. Per gestire file immutabili in un sistema che deve avere la possibilità di cambiare, è necessario aggiungere un layer al di sopra dei CID relativo ai **puntatori**.

É possibile utilizzare un *puntatore* variabile che fa riferimento al CID della pagina contenente l'ultimo aggiornamento. Così facendo, gli utenti potranno visitare il sito ed essere sempre indirizzati ai contenuti più recenti [3].

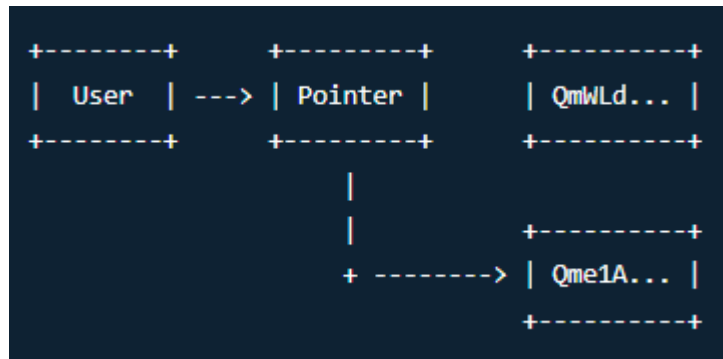


Figure 1: Esempio di puntatore

### 1.3 Disponibilità

Un'altra caratteristica importante che contraddistingue IPFS permette di evitare che il contenuto di un documento sia rimosso dall'architettura finché vi è qualcuno interessato abbastanza da garantirne la disponibilità, che sia esso l'autore originale del file o meno.

Facendo uso di questo protocollo, la richiesta viene inoltrata a tutti i dispositivi sparsi per il pianeta che utilizzano questo sistema, in modo che qualcuno di essi condivida la pagina in suo possesso. É importante notare come le informazioni possano essere recuperate da chiunque le possenga, non esclusivamente da Wikipedia.

Inoltre, nel momento in cui si impiega IPFS, non viene semplicemente scaricato il file richiesto, bensì il richiedente partecipa alla distribuzione del documento stesso: quando un nodo della rete necessita di una certa pagina, vi è la possibilità di ottenerla dall'elaboratore che in precedenza ne ha fatto richiesta, ma ha la medesima probabilità di riceverla da un peer limitrofo o da chiunque altro dispositivo usi IPFS.

Questa tecnologia rende possibile quanto descritto non solo per pagine web, ma anche per qualsiasi altro tipo di file, che si tratti di un documento, un'email, o persino un record di un database.

## 1.4 Decentralizzazione

Il fondamento su cui si basa IPFS è la **decentralizzazione**. Questa rendere possibile il download di un documento da posizioni differenti della rete, le quali **non** vengono gestite da un'entità centralizzata.

Grazie a questa caratteristica, IPFS sostiene una rete resistente agli attacchi informatici e complica notevolmente la censura dei contenuti.

A differenza del World Wide Web dei giorni nostri, il quale si struttura sulla base di **proprietà** e di **accesso**, IPFS si basa sui concetti di **possesso** e **partecipazione**. Per questa ragione, nodi diversi della rete possiedono file tra essi condivisi e partecipano per garantirne la disponibilità. Pertanto, le funzionalità di IPFS sono sfruttate a pieno solo quando essi partecipano attivamente nella rete: è ragionevole archiviare le copie dei documenti negli elaboratori che mantengono la loro esecuzione attiva con un'istanza di IPFS in esecuzione, in modo da rendere quei file disponibili per tutti coloro che ne hanno bisogno.

## 1.5 Nodi

I partecipanti che compongono la rete IPFS prendono il nome di **nodi**. Questi altro non sono che *programma IPFS* in esecuzione su elaboratori in grado di archiviare file e connettersi alla rete. I nodi rappresentano il cuore pulsante di IPFS: senza di essi non esisterebbe alcun network.

Il termine *nodo* è una locuzione molto generica, il cui significato varia a seconda del contesto. Esistono tre categorie principali di nodi: i nodi IPFS, i nodi di dati e i nodi libp2p per le applicazioni.

I **nodi IPFS**, come anticipato, sono programmi in esecuzione sui dispositivi che si vogliono identificare come tali, i quali hanno la capacità di scambiare dati tra loro. A seconda del contesto è possibile far loro riferimento con termini diversi:

- *nodo*: fa riferimento ad un singolo punto della rete, anche se può essere usato in ambiti differenti;
- *peer*: fa riferimento alla relazione (di uguaglianza) che c'è tra due nodi;
- *daemon*: fa riferimento allo stato di un nodo. Quando questo si trova online ed è in esecuzione in background, in attesa di richieste per i suoi dati, viene chiamato *demone*;
- *instance*: fa riferimento alla libreria (o al programma) in esecuzione su un nodo in un determinato momento.

I **nodi di dati** fanno riferimento ai dati effettivi memorizzati su IPFS, come i nodi DAG e i nodi IPLD. Quando si aggiunge un file a IPFS, questo viene suddiviso tra molteplici nodi, ognuno dei quali memorizza una parte del documento.

I **nodi libp2p** fanno riferimento ai nodi libp2p sui quali è possibile creare applicazioni. Nella relativa documentazione, questi vengono tipicamente identificati come *peer* [4].



## 2 Architettura

### 2.1 Content-addressing

IPFS utilizza il **content-addressing** per identificare il contenuto di un documento in base a *ciò che contiene*, piuttosto che al *luogo* in cui è archiviato.

La ricerca di un elemento in base al suo contenuto è una pratica diffusa che viene utilizzata anche nella vita quotidiana. Basti pensare alla ricerca di un libro all'interno di un archivio bibliografico, che tipicamente avviene tramite il **titolo** dell'opera. Se si volesse utilizzare una ricerca basata sulla posizione sarebbe necessario indicare **dove** quel libro è posizionato rispetto all'archivio, e nel caso in cui qualcuno lo spostasse non sarebbe più possibile trovarlo. Quanto appena descritto è il tipo di ricerca che viene utilizzata su Internet e sui device odierni [5].

Al contrario, ogni contenuto che utilizza il protocollo IPFS ha un **content identifier**, detto anche **CID**, che corrisponde al suo hash. L'hash è univoco al contenuto da cui è generato. Il content-addressing, attraverso l'hash, è diventato un mezzo ampiamente utilizzato per connettere i dati nei sistemi distribuiti.

Un CID non è altro che un'etichetta utilizzata per puntare a del materiale in IPFS. Esso non indica dove è archiviato il documento, ma definisce una sorta di indirizzo basato sul suo contenuto. I CID sono brevi, indipendentemente dalle dimensioni del file da cui sono generati: questi, infatti, hanno origine dall'hash crittografico del loro contenuto. Ciò significa che:

- Contenuti differenti produrranno CID differenti;
- Lo stesso contenuto archiviato su due nodi IPFS diversi produrrà lo stesso CID.

L'algoritmo di hashing utilizzato per generare i content identifier è lo **SHA-256**, ma vengono supportate anche molte altre funzioni di hash.

#### 2.1.1 Hashing

Una **funzione di hash** produce una sequenza di bit, detta *digest*, strettamente correlata con i dati in ingresso. Gli hash crittografici accettano input arbitrari e restituiscono un valore di lunghezza prefissata. L'output dipende dall'algoritmo hash utilizzato (ad esempio SHA-256, SHA-512, BLAKE2b, ecc...) ma un dato algoritmo restituisce sempre lo stesso valore di output per lo stesso valore di input.

La maggior parte delle funzioni di hash crittografiche è quindi progettata per prendere in input una stringa di qualsiasi lunghezza e produrre in output un valore di hash di lunghezza prefissata. Queste funzioni devono essere in grado di resistere a tutti gli attacchi basati sulla *crittoanalisi*, ovvero lo studio dei metodi per ottenere il significato di informazioni cifrate senza però avere accesso all'informazione segreta che è richiesta per effettuare quell'operazione[6]. Per tale scopo, si definisce il livello di sicurezza di una funzione di hash crittografica facendo riferimento alle seguenti proprietà:

- **Resistenza alla preimmagine:** dato un valore di hash  $h$ , deve essere difficile risalire ad un messaggio  $m$  con  $\text{hash}(m) = h$ . Questa proprietà deriva dal concetto di **funzione unidirezionale**.
- **Resistenza alla seconda preimmagine:** dato un messaggio  $m_1$ , deve essere difficile trovare un secondo messaggio  $m_2$  tale che  $\text{hash}(m_1) = \text{hash}(m_2)$ .
- **Resistenza alla collisione:** dati due messaggi  $m_1$  ed  $m_2$ , deve essere difficile che i due messaggi abbiano lo stesso hash, quindi con  $\text{hash}(m_1) = \text{hash}(m_2)$ . Tale coppia è chiamata **collisione di hash crittografica** [7].

Queste funzionalità indicano la possibilità di utilizzare un hash crittografica per identificare qualsiasi dato. L'hash è unico per i dati da cui è stato calcolato e ha una dimensione ridotta, quindi inoltrarlo sulla rete richiede un numero limitato di risorse.

Tutte queste caratteristiche sono fondamentali per un sistema distribuito come IPFS, in cui si vuole essere in grado di archiviare e recuperare dati da luoghi differenti. Senza un identificatore breve e univoco come l'hash, l'indirizzamento dei contenuti non sarebbe possibile.

È importante notare che, poiché IPFS suddivide il contenuto in blocchi e li verifica tramite **grafici aciclici diretti (DAG)**, gli hash dei file **non** corrisponderanno ai CID siccome ogni blocco ha il proprio CID.

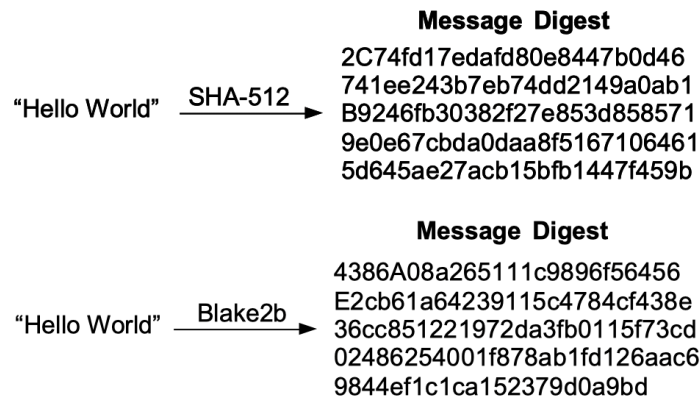


Figure 2: Una funzione crittografica di hash al lavoro

Prendendo come esempio di input la stringa `Hello world`, si otterrebbe come risultato dell'operazione di hashing (utilizzando SHA-256) il seguente digest a 256 bit:

`0x64EC88CA00B268E5BA1A35678A1B5316D212F4F366B2477232534A8AECA37F3C`.

Lo `0x` anteposto indica che l'output è rappresentato in formato esadecimale.

Gli hash possono essere rappresentati su *basi* differenti (base2, base16, base32, ecc...). IPFS utilizza questa caratteristica come parte dei suoi identificatori di contenuto in

modo da supportare diverse rappresentazioni di base contemporaneamente, utilizzando *Multibase*.

Ad esempio, l'hash SHA-256 di `Hello world` può essere rappresentato in base32 come:

`mtwirsqawjuoloq2gvtyug2tc3jbf5htm2zeo4rsknfiv3fdp46a`

**Multibase** è un protocollo che permette di rispondere alla domanda: avendo i dati `d` codificati nel testo `s`, con quale *base* è stata effettuata la codifica?

Questo sistema permette di disambiguare la codifica di binari *codificati in base* che appaiono nel testo. [8]

Di seguito viene riportata l'attuale tabella multibase contenente tutte le codifiche di rilievo.

1	encoding	code	description	status
2	identity	0x00	8-bit binary (encoder and decoder keeps data unmodified)	default
3	base2	0	binary (01010101)	candidate
4	base8	7	octal	draft
5	base10	9	decimal	draft
6	base16	f	hexadecimal	default
7	base16upper	F	hexadecimal	default
8	base32hex	v	rfc4648 case-insensitive - no padding - highest char	candidate
9	base32hexupper	V	rfc4648 case-insensitive - no padding - highest char	candidate
10	base32hexpad	t	rfc4648 case-insensitive - with padding	candidate
11	base32hexpadupper	T	rfc4648 case-insensitive - with padding	candidate
12	base32	b	rfc4648 case-insensitive - no padding	default
13	base32upper	B	rfc4648 case-insensitive - no padding	default
14	base32pad	c	rfc4648 case-insensitive - with padding	candidate
15	base32padupper	C	rfc4648 case-insensitive - with padding	candidate
16	base32z	h	z-base-32 (used by Tahoe-LAFS)	draft
17	base36	k	base36 [0-9a-z] case-insensitive - no padding	draft
18	base36upper	K	base36 [0-9a-z] case-insensitive - no padding	draft
19	base58btc	z	base58 bitcoin	default
20	base58flickr	Z	base58 flickr	candidate
21	base64	m	rfc4648 no padding	default
22	base64pad	M	rfc4648 with padding - MIME encoding	candidate
23	base64url	u	rfc4648 no padding	default
24	base64urlpad	U	rfc4648 with padding	default
25	proquint	p	PRO-QUINT <a href="https://arxiv.org/html/0901.4016">https://arxiv.org/html/0901.4016</a>	draft
26	base256emoji	🏳️‍🌈	base256 with custom alphabet using variable-sized-codepoints	draft

Figure 3: Tabella Multibase

Come si evince dalla colonna “status”, ogni codifica multibase ha uno stato che può assumere i valori di **bozza** (*draft*) nel caso in cui la codifica sia stata proposta ma non sia completamente implementata, **candidata** (*candidate*), se la codifica è matura ma non implementata da tutte le implementazioni, e infine **default** nel caso in cui la codifica sia ampiamente utilizzata e implementata da tutte le implementazioni.

### 2.1.2 Multihash

**Multihash** è un protocollo che consente di distinguere gli output ottenuti dalle funzioni hash consolidate, affrontando considerazioni sulle dimensioni e sul tipo di codifica.

Un multihash non è altro che un hash autodescrittivo che contiene metadati che ne descrivono sia la lunghezza sia l'algoritmo crittografico che lo ha generato.

Il progetto Multihash permette di utilizzare, oltre alla predefinita SHA-256, altri algoritmi di hashing con l'obiettivo di rendere “a prova di futuro” l'uso delle hash da parte delle applicazioni e per consentire la coesistenza di più hash functions nella stessa organizzazione.

Multihash è particolarmente importante in sistemi che dipendono dalle funzioni hash crittografiche, come IPFS. Alcuni tipi di attacchi informatici potrebbero però fare breccia nelle proprietà delle funzioni hash sicure, provocandone quindi la *rottura*.

Queste rotture sono particolarmente dolorose nei grandi ecosistemi di strumenti, dove potrebbero essere state fatte supposizioni sui valori di hash, come la funzione utilizzata e la dimensione del digest. In questi casi, l'aggiornamento è particolarmente problematico in quanto gli strumenti che fanno queste ipotesi dovrebbero essere aggiornati per usare i nuovi valori.

Multihash è stato progettato per semplificare enormemente il processo di aggiornamento infatti, nel momento in cui avviene una rottura, aggiorna automaticamente i valori di hash. Anche se un sistema implementa una singola hash function, l'uso di multihash chiarisce che i valori di hashing potrebbero variare.

In questo modo, strumenti, applicazioni e script possono evitare di fare ipotesi sui valori di hash e leggerli direttamente dal valore multihash.

Così facendo, la stragrande maggioranza degli strumenti non dovrebbe essere aggiornata affatto. [9]

Il formato Multihash segue il modello **TLV** (type-length-value).

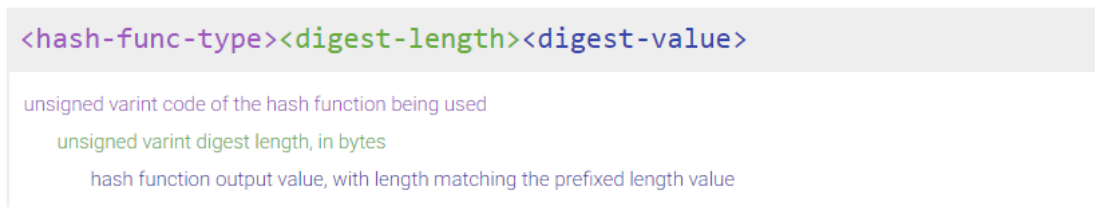


Figure 4: Multihash Format

- Il *type* “hash-func-type” è un **intero variabile senza segno** che identifica la funzione hash utilizzata. La tabella **Multicodec** contiene i valori predefiniti relativi alle funzioni hash che possono essere utilizzate da questo protocollo.

1	name	tag	code	status	description
2	identity	multihash	0x00	permanent	raw binary
3	cidv1	cid	0x01	permanent	CIDv1
4	cidv2	cid	0x02	draft	CIDv2
5	cidv3	cid	0x03	draft	CIDv3
6	ip4	multiaddr	0x04	permanent	
7	tcp	multiaddr	0x06	permanent	
8	sha1	multihash	0x11	permanent	
9	sha2-256	multihash	0x12	permanent	
10	sha2-512	multihash	0x13	permanent	
11	sha3-512	multihash	0x14	permanent	
12	sha3-384	multihash	0x15	permanent	
13	sha3-256	multihash	0x16	permanent	
14	sha3-224	multihash	0x17	permanent	
15	shake-128	multihash	0x18	draft	
16	shake-256	multihash	0x19	draft	
17	keccak-224	multihash	0x1a	draft	keccak has variable output length. The number specifies the core length
18	keccak-256	multihash	0x1b	draft	
19	keccak-384	multihash	0x1c	draft	
20	keccak-512	multihash	0x1d	draft	
21	blake3	multihash	0x1e	draft	BLAKE3 has a default 32 byte output length. The maximum length is (2^64)-1 bytes.
22	sha2-384	multihash	0x20	permanent	aka SHA-384; as specified by FIPS 180-4.
23	dccp	multiaddr	0x21	draft	
24	murmur3-x64-64	multihash	0x22	permanent	The first 64-bits of a murmur3-x64-128 - used for UnixFS directory sharding.

Figure 5: Una porzione della tabella Multicodec

- La *length* “digest-length” è un *intero variabile senza segno* che definisce la lunghezza del digest, in byte
- Il *value* “digest-value” è il digest della funzione hash, con una lunghezza pari a “digest-length” byte.

Il seguente esempio multihash mostra diversi output ottenuti con differenti funzioni hash ma a partire dallo stesso input, ovvero **Merkle-Damgård**.

## sha2-256 - 256 bits (aka sha256)

122041dd7b6443542e75701aa98a0c235951a28a0d851b11564d20022ab11d2589a8

Hashing function encoded as varint: sha2-256 (code in hex: 0x12)

Length: 32 (in hex: 0x20)

Digest: 41dd7b6443542e75701aa98a0c235951a28a0d851b11564d20022ab11d2589a8

## sha2-512 - 256 bits

132052eb4dd19f1ec522859e12d89706156570f8fbab1824870bc6f8c7d235eef5f4

Hashing function encoded as varint: sha2-512 (code in hex: 0x13)

Length: 32 (in hex: 0x20)

Digest: 52eb4dd19f1ec522859e12d89706156570f8fbab1824870bc6f8c7d235eef5f4

Figure 6: Un esempio di multihash

Si noti che, poiché IPFS trova il contenuto tramite multihash, l'uso di un algoritmo di hashing diverso da quello predefinito (SHA-256) limiterà il numero di peer da cui è possibile recuperare il contenuto. Questo perché altri peer che effettivamente possiedono lo stesso documento lo identificheranno con un hash differente.

### 2.1.3 Identifier Formats

In base alla codifica utilizzata o alla versione, i CID possono assumere diverse forme. Molti degli strumenti IPFS generano **CID di versione 0 (CIDv0)**, nonostante file e operazioni sugli oggetti utilizzino i più recenti **CID di versione 1 (CIDv1)**.

Durante la progettazione di IPFS, venne utilizzata la codifica in base58btc di *multihash* per rappresentare gli identificatori di contenuto v0. I CID più recenti sono più flessibili rispetto ai CIDv0, anche se questi sono ancora utilizzati per impostazione predefinita da molte operazioni di IPFS.

Per identificare facilmente un CIDv0 è sufficiente verificare se un CID ha come caratteri iniziali **Qm**.

Il CIDv1, invece, contiene alcuni identificatori principali che chiariscono esattamente quale rappresentazione viene utilizzata, insieme al hash del contenuto stesso. Questi includono:

- **Un prefisso multibase**, che specifica la codifica utilizzata per il resto del CID;
- **Un identificatore di versione CID**, che indica di quale versione di CID si tratta;

- Un **identificatore multicodec**, che indica il formato del contenuto di destinazione: aiuta le persone e il software a capire come interpretare i dati una volta che il contenuto è stato recuperato.

CID
[Docs](#) [Spec](#) [Tutorial](#)

bafybeigdyrzt5sfp7udm7hu76uh7y26nf3efuy1qabf3oc1gtqy55fbzdi

**HUMAN READABLE CID**

base32 - cidv1 - dag-pb - (sha2-256 : 256 : C3C4733EC8AFFD06CF9E9FF50FFC6BCD2EC85A6170004BB709669C31DE94391A)

MULTIBASE - VERSION - MULTICODEC - MULTIHASH (NAME : SIZE : DIGEST IN HEX)

MULTIBASE	MULTICODEC	MULTIHASH
PREFIX: <b>b</b>	CODE: <b>0x70</b>	CODE: <b>0x12</b>
NAME: <b>base32</b>	NAME: <b>dag-pb</b>	NAME: <b>sha2-256</b>
		BITS: <b>256</b>
		DIGEST (HEX): <b>C3C4733EC8AFFD06CF9E9FF50FFC6BCD2EC85A6170004BB709669C31DE94391A</b>

**CIDV1 (BASE32)**

bafybeigdyrzt5sfp7udm7hu76uh7y26nf3efuy1qabf3oc1gtqy55fbzdi

Figure 7: Esempio di CIDv1

Questi identificatori principali forniscono anche compatibilità con le versioni successive, supportando diversi formati da utilizzare nelle versioni future dei CID.

É possibile utilizzare i primi byte del CID per interpretare il resto dell'indirizzo del contenuto e sapere come decodificarlo dopo essere stato recuperato da IPFS.

Il progetto IPFS passerà ai CIDv1 come nuova impostazione predefinita nel prossimo futuro.

## 2.2 Directed Acyclic Graphs (DAGs)

Molti sistemi distribuiti, così come IPFS, impiegano la struttura dati chiamata **Directed Acyclic Graphs**, ovvero i *grafi aciclici diretti*, detti anche **DAGs**.

Un DAG è un **grafo orientato**, ovvero un grafo costituito da un insieme di nodi collegati da archi diretti, che non presenta **cicli**, in modo tale che seguendo le direzioni degli archi non si formi mai un anello chiuso.

Un grafo orientato è un DAG se e solo se può essere ordinato *topologicamente* ovvero se, tramite un ordinamento lineare dei suoi vertici, per ogni arco diretto  $uv$ ,  $u$  antecede  $v$  [10]. Un vertice  $v$  di un grafo orientato si dice raggiungibile da un altro vertice  $u$  quando esiste un percorso che inizia in  $u$  e termina in  $v$ . In un caso speciale, ogni vertice è considerato raggiungibile da se stesso. Se un vertice può raggiungere se stesso tramite un percorso non banale (ovvero tramite un percorso composto da uno o più archi), questo

rappresenta un ciclo. Per questa ragione, un altro modo per definire i DAGs è che sono grafi in cui nessun vertice può raggiungere se stesso tramite un percorso non banale [11].

IPFS utilizza i **Merkle DAGs** per rappresentare file e directory attraverso l'uso dei *CID*.

Un *Merkle DAG* è un DAG in cui ogni nodo del grafo ha un identificatore univoco che altro non è che un hash del suo contenuto.

Questa caratteristica porta ad alcune considerazioni importanti:

- I Merkle DAG vengono costruiti a partire dalle foglie, cioè dai nodi senza figli. I genitori sono aggiunti solo successivamente, siccome gli identificatori dei figli devono essere calcolati in anticipo per poterli collegare ai nodi padre;
- Ogni nodo in un Merkle DAG è la radice di un *subMerkle DAG* contenuto nel DAG padre. Il CID di un nodo radice identifica in modo univoco non solo quel nodo, ma l'intero DAG di cui esso è radice. Di conseguenza, è possibile estendere tutte le garanzie di sicurezza, integrità e permanenza dei CID alla struttura dati stessa;
- I nodi di un Merkle DAG sono immutabili: qualsiasi modifica in un nodo altererebbe il suo CID e di conseguenza influenzerebbe tutti gli ascendenti nel DAG, creando essenzialmente un DAG differente.
- Per condividere una parte di un DAG con qualcuno, è sufficiente inviare all'interessato solo il CID del sotto-grafo d'interesse. Non vi sono, inoltre, vincoli per incorporare il sotto-grafo come parte di un DAG più grande, siccome il CID del DAG (che è quello del nodo radice) dipende dai suoi discendenti e non dai suoi antenati;
- Chiunque possieda un DAG è in grado di proporsi come suo fornitore. Quando si ha la necessità di recuperare dati codificati come DAG, (ad esempi una directory di file), è possibile sfruttare la parallelizzazione per risalire, potenzialmente da diversi provider, ai figli di quel nodo;
- I Merkle DAG forniscono un modo semplice ed efficace per archiviare i dati codificando le sezioni ridondanti come collegamenti.

Entrando nel merito dell'ultima considerazione, si prenda in esempio un DAG relativo a una directory contenente due cartelle: “**fish**” e “**cats**”, ognuna delle quali archivia alcune immagini. Eliminando la cartella “**fish**”, sostituendola con una directory chiamata “**dogs**”, si dà luogo a un nuovo DAG rappresentante lo stato aggiornato della directory root. Tuttavia, i nodi che rappresentano la cartella “**cats**” e i relativi file sono comuni a entrambi i DAGs. Pertanto, è possibile riutilizzarli, come illustrato di seguito, dove i nodi arancioni rappresentano i nodi utilizzati solo nel DAG originale, quelli verdi rappresentano quelli comuni a entrambi, e quelli blu rappresentano i nodi necessari per il nuovo stato.



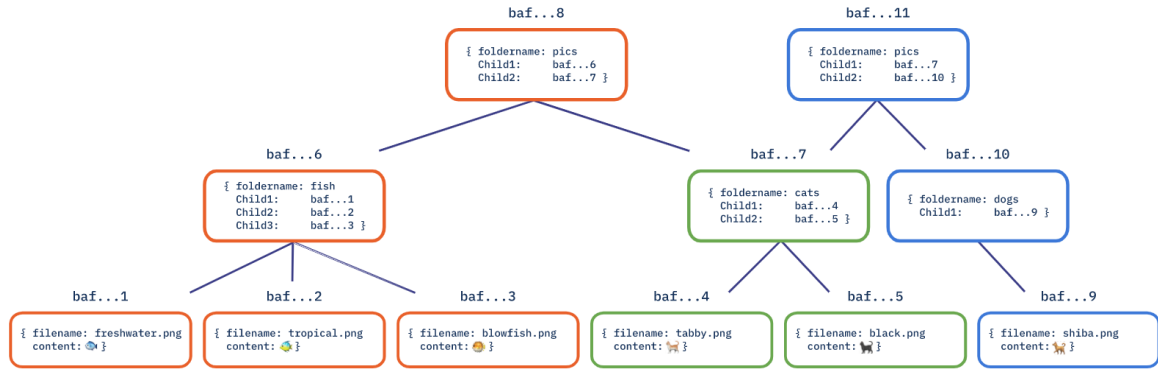


Figure 8: Esempio sezioni ridondanti DAG

Ciò significa che è possibile memorizzare entrambe le versioni della directory root senza occupare il doppio dello spazio necessario per memorizzare una singola versione.

I Merkle DAG offrono, quindi, un modo flessibile per modellare e condividere dati sul web distribuito. Questa loro capacità li rende l'elemento costitutivo di base per molti progetti importanti, tra cui sistemi di *version control* come Git, *blockchain* come Ethereum, *protocolli web decentralizzati* come **IPFS** e *reti di archiviazione distribuite* come Filecoin [12].

### 2.3 Distributed Hash Tables (DHTs)

Per identificare quali peer stanno ospitando il contenuto che si sta cercando, IPFS utilizza una **Distributed Hash Table**, o *DHT*.

Una *hash table*, in italiano *tabella hash*, è una struttura dati utilizzata per relazionare una **chiave** con un determinato **valore**. Queste vengono spesso utilizzate nei metodi di ricerca nominati “hashing”, nei quali si tenta di accedere agli elementi della tabella in modo diretto tramite operazioni aritmetiche che trasformano le chiavi in indirizzi [13].

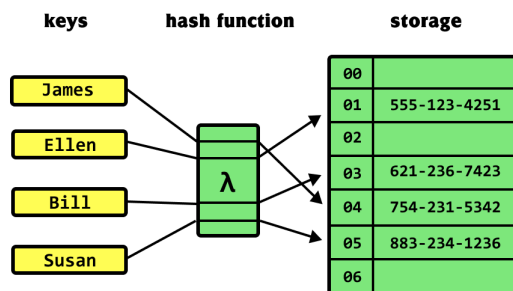


Figure 9: Hash table

Le **Distributed Hash Tables** sono una classe di sistemi distribuiti decentralizzati che partizionano l'appartenenza di un set di chiavi tra i nodi partecipanti nella rete, i quali possono inoltrare in maniera efficiente i messaggi all'unico proprietario di una determinata chiave. Ciascun nodo è l'analogo di un array slot in una hash table.

Una **DHT** rappresenta, quindi, un sistema distribuito che fornisce un servizio di ricerca simile a quello delle tabelle hash: le coppie *chiave-valore* sono archiviate in una DHT e qualsiasi peer della rete può recuperare in modo efficiente il *valore* associato a una determinata *chiave*. Il vantaggio principale di una DHT è che i nodi possono essere aggiunti o rimossi con un lavoro minimo rispetto alla ridistribuzione delle chiavi. Le chiavi sono identificatori univoci che si associano a valori particolari. La responsabilità di mantenere la mappatura dalle chiavi rispetto ai valori è distribuita tra i nodi, in modo che un cambiamento nell'insieme dei partecipanti causi il minimo disagio. Ciò consente a una DHT di scalare a un numero estremamente elevato di nodi e di gestirne continuamente ingressi, uscite e guasti.

Le DHT formano un'infrastruttura che può essere utilizzata per costruire servizi più complessi, come anycast, web caching cooperativo, file system distribuiti, servizi di domain name, messaggistica istantanea, sistemi di condivisione file e distribuzione di contenuti peer-to-peer.

Notevoli reti distribuite che utilizzano le DHT includono il tracker distribuito di BitTorrent, la rete Kad, la botnet Storm, Freenet, il motore di ricerca YaCy e l'**InterPlanetary File System**. [14]

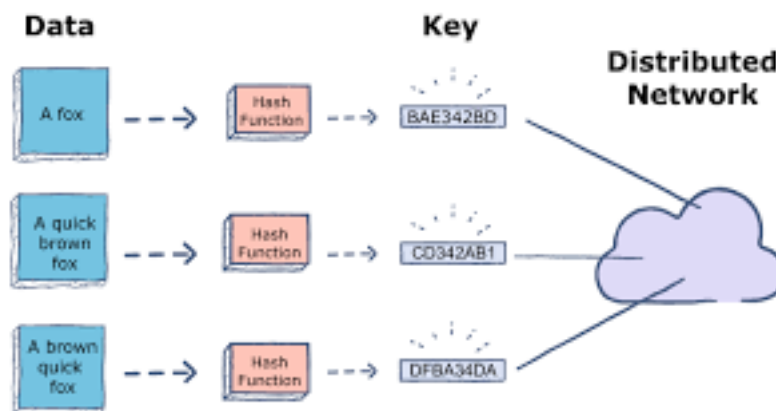


Figure 10: Esempio di una DHT

In IPFS, la DHT è utilizzata come componente fondamentale del **sistema di routing dei contenuti**, agendo come una combinazione tra un catalogo e un sistema di navigazione. Essa mappa ciò che l'utente sta cercando rispetto al peer che sta archiviando il contenuto d'interesse. La si può pensare come un enorme tabella che mantiene memorizzato *chi* ha *quali* dati.

Esistono tre tipi di coppie *chiave-valore* che vengono mappate utilizzando la DHT:

- **Record del provider:** mappa un identificatore di dati (ad esempio un *multihash*) su un peer che ha annunciato di possedere quel contenuto e che è disponibile alla sua condivisione;
- **Record IPNS:** mappa una chiave IPNS (ad esempio l'hash di una chiave pubblica) su un record IPNS (ad esempio un puntatore firmato a un percorso come `/ipfs/bafyxyz...`);
- **Record del peer:** mappa un peerID ad un insieme di indirizzi multipli da cui il peer può essere raggiunto. Questo viene utilizzato da IPFS quando si è a conoscenza del contenuto di un peer ma non dell'indirizzo.

Questi tipi di record hanno una semantica leggermente diversa, ma sono tutti aggiornati e trovati utilizzando lo stesso protocollo DHT. IPFS impiega *Kademlia* per la sua Distributed Hash Table.

## 2.4 Kademlia

**Kademlia** è un protocollo di rete peer-to-peer per un network di nodi decentralizzato. Essa specifica la struttura del network, regola la comunicazione tra i peer e il modo in cui lo scambio di informazioni deve essere effettuato. I nodi Kademlia comunicano tra di loro utilizzando il protocollo di trasporto UDP [15].

L'algoritmo di Kademlia ha come scopo quello di costruire una DHT sulla base di tre parametri di sistema:

- **Uno spazio di indirizzi** per identificare in modo univoco tutti i peer della rete. In IPFS, tutti i numeri interi che vanno da 0 a  $2^{256}-1$ .
- **Una metrica** per ordinare i peer nello *spazio degli indirizzi*, e quindi visualizzare tutti i nodi lungo una retta ordinata, dal più piccolo al più grande. IPFS prende lo  $SHA256(PeerID)$  e lo interpreta come un numero intero compreso tra 0 e  $2^{256}-1$ .
- **Una proiezione** che prende un record chiave e ne calcola la posizione nello *spazio degli indirizzi* in cui dovrebbe essere il peer (o i peer) più adatto alla memorizzazione del record. IPFS usa  $SHA256(Record\ Key)$ .

Grazie alla presenza dello spazio degli indirizzi e della metrica di ordinamento dei peer, è possibile fare ricerche nella rete come se questa fosse un elenco ordinato. Nello specifico, è possibile considerare il sistema come una *skip-list*, ovvero una struttura dati per la memorizzazione di una lista ordinata di elementi, in cui un peer conosce altri peer con distanze crescenti da esso (1,2,4,8...).

A differenza di una *skip-list*, Kademlia è alquanto instabile poiché i peer possono entrare, uscire e rientrare nella rete in qualsiasi momento. Per affrontare la natura instabile del sistema, un peer Kademlia non si limita a mantenere i collegamenti ai peer con distanza  $X$  da esso. Quel che accade è che per ogni nodo  $Z$ , la cui distanza rispetto ad un nodo  $Y$  è multipla di 2, mantiene fino a  $K$  collegamenti. In IPFS,  $K = 20$ . Questo

significa, per esempio, che invece di esserci un peer che mantiene un singolo collegamento con un nodo lontano 128 da esso, mantiene 20 collegamenti che si trovano tra i 65 e i 128 di distanza.

La selezione di parametri a livello di rete, come  $K$ , non è arbitraria. Esso viene determinato in base all'abbandono medio osservato nella rete e alla frequenza con cui il network ripubblica le informazioni. I parametri di sistema vengono calcolati per massimizzare la probabilità che la rete rimanga connessa e che nessun dato venga perso, mantenendo la latenza desiderata per le query e assumendo che l'abbandono medio osservato rimanga costante. Questi parametri guidano le decisioni prese nei due componenti principali di Kademlia: *la tabella di routing*, che tiene traccia di tutti i collegamenti nella rete, e *l'algoritmo di ricerca*, che determina come attraversare quei collegamenti per archiviare e recuperare dati.

### 2.4.1 Routing Tables

Una **tabella di routing** è un insieme di regole utilizzate per decidere dove devono essere direzionati i dati che viaggiano su una rete. Tutti i dispositivi abilitati IP, inclusi router e switch, utilizzano tabelle di routing. Ogni peer IPFS mantiene una tabella di routing con collegamenti ad altri peer della rete.

IPFS si affida a Kademlia per definire cosa dovrebbe e non dovrebbe entrare nella tabella di routing:

1. Quando ci si collega a un peer, controlla se è idoneo per essere aggiunto alla tabella di routing;
2. Se si qualifica come idoneo, determina quanto è vicino il nuovo peer per capire in quale *bucket* dovrebbe essere inserito;
3. Prova a inserire il peer nel *bucket*;
4. In caso di un tentativo di connessione infruttuoso a un peer della tabella di routing, questo viene eliminato da essa.

Nella lista sopracitata vi sono tre proprietà degne di approfondimento: *qualifica di un nodo*, *bucket* e *aggiornamento/eliminazione di peer*, ma prima di scendere nello specifico è necessario fare una precisazione riguardo alla DHT.

Molti nodi IPFS utilizzano la Distributed Hash Table condivisa pubblicamente per scoprire e pubblicizzare i contenuti. Tuttavia, alcuni nodi operano su reti separate, come reti locali o VPN isolate. Per questi utenti, avere una DHT in cui tutti i nodi non raggiungibili pubblicamente sono client è molto problematico poiché nessuno di essi è, appunto, raggiungibile pubblicamente.

Una DHT separata è disponibile per i nodi che non fanno parte della rete pubblica, denominata **LAN DHT**. Questa è completamente separata dalla **WAN DHT** pubblica. Queste due DHT sono separate utilizzando diversi nomi di protocollo DHT:

DHT	Path
WAN	/ipfs/kad/1.0.0
LAN	/ipfs/lan/kad/1.0.0

Le principali differenze tra la DHT WAN e quella LAN sono i criteri di accettazione per i peer: quali nodi sono idonei a far parte di una tabella di routing o di query, e quali no. Il criterio della WAN DHT è *sembri un indirizzo pubblico*, mentre il criterio della LAN DHT è *sembri un indirizzo non pubblico*. Mentre i nodi WAN DHT passano dalla modalità **client** (in cui possono interrogare la DHT ma non rispondere alle query) a quella **server** (in cui possono sia interrogare che rispondere alle query) in base alla possibilità di essere contattati pubblicamente, i nodi LAN DHT sono sempre **server** a meno che non sia stata impostata l'opzione **dhtclient**.

A fronte dell'approfondimento appena trattato, i peer **qualificati** che possono essere aggiunti a una tabella di routing devono soddisfare i criteri seguenti: il peer è un server DHT che pubblicizza l'ID del protocollo DHT, */ipfs/kad/1.0.0* per WAN DHT, e */ipfs/lan/kad/1.0.0* per LAN DHT. Inoltre, il peer ha indirizzi IP che corrispondono all'intervallo atteso, ad esempio un indirizzo IP nell'intervallo pubblico (per i membri della DHT pubblica), che esulano dagli indirizzi quali *192.168.X.Y*.

Un **bucket** è, invece, una raccolta di massimo 20 peer che possiedono indirizzi simili tra loro. Se, ad esempio, un peer ha una distanza compresa tra  $2^7$  e  $2^8$  rispetto a un altro nodo e lo spazio degli indirizzi ha dimensione  $2^{256}$ , allora quel peer dovrà essere inserito all'interno del bucket 256-8. I nodi possono essere aggiunti a un bucket solo se al suo interno vi sono meno di 20 peer. Se il bucket è già al completo, IPFS determina se è possibile eliminare un qualsiasi nodo dal suo interno. In caso contrario, il peer non verrà aggiunto a quel bucket.

In quanto **all'aggiornamento delle tabelle di routing**, IPFS si occupa di controllarne l'accuratezza a cadenza regolare di **10 minuti**. Sebbene questa frequenza sia probabilmente superiore a quella strettamente necessaria, si è deciso di adottarla ugualmente in modo da proteggere al meglio la salute della rete.

Un aggiornamento della tabella di routing avviene come segue:

- Tutti i bucket che contengono almeno un peer al loro interno vengono esaminati. È importante notare come il numero massimo di bucket sia limitato a 15;
  - Per ogni bucket viene selezionato un indirizzo randomico che ha la possibilità di rientrare nel suo spazio Kademlia. Successivamente viene eseguita una ricerca per trovare i K peer a esso più vicini. Così facendo, ogni bucket ha la garanzia di essere riempito.
- Nel caso in cui le dimensioni e la distribuzione della rete siano tali per cui i 15 bucket non siano sufficienti per conoscere i K peer più vicini a un nodo X, viene effettuata una ricerca di se stesso sulla rete.

I peer possono essere eliminati dalla tabella di routing per motivi differenti, in genere perché quel peer è offline o irraggiungibile. Dopo ogni aggiornamento, IPFS controlla la

tabella di routing e tenta di connettersi ai peer che non sono stati interrogati di recente. Se alcuni di questi peer non sono attivi oppure sono offline, vengono eliminati dalla tabella di routing. I peer possono anche essere eliminati se non sono stati utili durante il periodo di tempo nel quale si prevede che, probabilisticamente, siano stati utilizzati in un aggiornamento. Tale valore è  $\text{Log}(1/K) * \text{Log}(1 - \alpha/K) * \text{refreshPeriod}$ , dove  $\alpha$  è il numero di peer raggiungibili che possono essere interrogati contemporaneamente.

#### 2.4.2 Lookup algorithm

**L'algoritmo di ricerca** risponde alla domanda: quali sono i  $K$  peer più vicini a  $X$ ? L'implementazione IPFS dell'algoritmo di ricerca di Kademlia utilizza i seguenti step per rispondere a questa domanda:

1. Individua i  $K$  peer più vicini a  $X$  tramite la tabella di routing e li carica nella coda delle query;
2. Avendo la possibilità di effettuare fino a 10 interrogazioni simultaneamente, recupera uno di questi nodi dalla *queue* e lo interroga, domandandogli quali sono i  $K$  peer più prossimi a  $X$ ;
3. Una volta terminata l'interrogazione al nodo, aggiunge i risultati ottenuti alla coda delle query;
4. Estrae il peer più vicino successivo nella coda e ripete l'interrogazione;
5. La query termina nel momento in cui i tre peer noti più vicini a  $X$  sono stati interrogati correttamente senza interruzioni o errori;
6. Al termine delle query, i  $K$  peer più vicini che non hanno fallito vengono restituiti.

Sebbene l'algoritmo di ricerca consenta a IPFS di **inserire** e **ottenere** record dalla DHT, il modo in cui queste attività vengono realizzate è leggermente diverso in base al tipo di record trattato.

Per *l'inserimento* di **un blocco con multihash  $H$** , viene eseguita una ricerca per i  $K$  peer più vicini a  $\text{SHA256}(H)$ , per poi aggiungere il record del provider nei  $K$  nodi più prossimi ad esso. Per il suo *recupero*, invece, viene effettuata una ricerca dei  $K$  peer più vicini a  $X = \text{SHA256}(H)$ , si interroga ognuno di essi per individuare i  $K$  nodi più vicini ad  $X$  conosciuti e si richiede l'inoltro del record corrispondente ad  $X$  se qualcuno di essi ne è a conoscenza.

Per *l'inserimento* di **una chiave IPNS dove il multihash della chiave pubblica è  $H$** , viene eseguita una ricerca per i  $K$  peer più vicini a  $\text{SHA256}(/ipns/H)$ , per poi aggiungere il record IPNS nei  $K$  nodi più prossimi ad esso. Per il suo *recupero*, invece, viene effettuata una ricerca per i  $K$  peer più vicini a  $X = \text{SHA256}(/ipns/H)$ , si interroga ognuno di essi per individuare i  $K$  nodi più vicini a  $X$  conosciuti e si richiede l'inoltro del record corrispondente a  $X$  se qualcuno di essi ne è a conoscenza. Se si riceve un record con un numero di sequenza IPNS più alto, si aggiorna quello esistente e si continua fino

al termine della ricerca. Ciò è necessario per assicurarsi che l'utente ottenga il record più recente. Questo perché i record IPNS sono mutevoli e, pertanto, è necessario assicurarsi di indirizzare una richiesta all'ultima versione del contenuto. Una volta completata la ricerca, i  $K$  peer più vicini a  $X$  si occuperanno di inviargli il record più recente da essi conosciuto.

Per *l'inserimento* di **un record di un peer il cui multihash della chiave pubblica è  $H$** , viene inizialmente effettuato uno scambio automatico delle informazioni contenute sui peer. Successivamente, facendo parte della DHT (come client o come server), il record del peer viene ereditato in seguito ai frequenti contatti con i  $K$  nodi a esso più vicini. Per il suo *recupero*, invece, viene effettuata una ricerca dei  $K$  peer più vicini a  $X = \text{SHA256}(H)$ , si interroga ognuno di essi per individuare i  $K$  nodi conosciuti più vicini a  $X$ , e si richiede l'inoltro del record del nodo corrispondente ad  $H$  se qualcuno di essi ne è a conoscenza.

### 2.4.3 Peer irraggiungibili

Una delle principali proprietà di Kademlia è che i peer possono essere organizzati in ordine crescente, in modo da formare un'ipotetica retta di nodi. Questo è utile perché, per esempio, mentre il **peer 0** “si muove” lungo la linea per trovare il **peer 55**, può sapere che si sta progressivamente avvicinando ad esso. Tuttavia, ciò richiede che tutti i nodi sulla linea possano parlare tra di loro. In caso contrario, il **peer 33** potrebbe mandare il **peer 0** in un vicolo cieco comunicandogli che il contenuto che sta cercando è su un nodo con cui non è in grado di comunicare. Questo può comportare una rete lenta e frammentata, con dati accessibili da alcuni nodi ma non da altri.

Sebbene avere peer che non possono parlare tra loro sembri una stranezza, due cause principali di irraggiungibilità sono i traduttori di indirizzi di rete (**NAT**) e i **firewall**. Avere reti asimmetriche in cui i peer  $X$ ,  $Y$  e  $Z$  possono connettersi ad  $A$ , ma  $A$  non ad essi, è abbastanza consueto. Allo stesso modo, è estremamente comune che i peer  $A$  e  $B$ , entrambi connessi ad un NAT, non possano parlare tra loro. Per far fronte a questo problema, i nodi IPFS ignorano i nodi ritenuti irraggiungibili dal pubblico. Inoltre, i nodi filtrano eventualmente anche se stessi fuori dalla rete se sospettano di non essere raggiungibili dagli altri. Solo quando i peer rilevano di essere contattabili pubblicamente passano dalla modalità **client** alla modalità **server**. Allo stesso modo, se un server scopre che non è più raggiungibile pubblicamente, tornerà alla modalità client [16].

## 2.5 Libp2p e Bitswap

Il progetto **libp2p** descrive la parte relativa all'ecosistema di IPFS che si occupa di fornire la DHT e di gestire la connessione e il dialogo tra peer.

Libp2p è uno “stack di rete” che consente alle applicazioni di utilizzare solo i protocolli strettamente necessari, senza rinunciare all'interoperabilità e alla possibilità di aggiornamento. Il progetto è nato da IPFS, ma è stato realizzato in modo che possa essere utilizzato per molteplici progetti.

Per richiedere e inviare blocchi ad altri peer, invece, viene utilizzato un modulo chiamato **Bitswap**. *Bitswap* permette di connettersi al peer o ai peer che hanno il contenuto desiderato, inviare loro la propria “lista dei desideri” (la quale contiene un elenco di tutti i blocchi d’interesse) per poi ricevere i blocchi richiesti. Una volta che questi saranno ricevuti è possibile verificarli eseguendo l’hashing del loro contenuto, in modo da ottenere i relativi CID e confrontarli con gli identificatori di contenuto richiesti.

### 2.5.1 Libp2p

Libp2p è un sistema modulare di **protocolli**, **specifiche** e **librerie** che consentono lo sviluppo di applicazioni di rete peer-to-peer. A causa del modo in cui libp2p è stato progettato, molte delle esigenze e delle considerazioni su cui è stato costruito il Web 2.0 non sono più applicabili [17].

Alla base di libp2p c’è il **livello di trasporto**, che è responsabile dell’effettiva trasmissione e ricezione dei dati da un peer all’altro. Libp2p fornisce un’interfaccia semplice che può essere adattata per supportare protocolli esistenti e futuri, consentendo alle applicazioni di operare in ambienti di rete differenti e a diversi runtime.

In un mondo con miliardi di dispositivi collegati in rete, sapere con chi si sta comunicando è la chiave per una comunicazione *sicura* e *affidabile*. A tal proposito, libp2p utilizza la **crittografia a chiave pubblica** per l’identificazione dei peer, la quale ricopre due scopi complementari: innanzitutto, assegna a ciascun nodo un *identificativo univoco* a livello globale, sotto forma di **PeerId**. In secondo luogo, il PeerId consente a chiunque di recuperare la chiave pubblica del peer a cui fa riferimento, il che consente una comunicazione sicura tra nodi.

Entrando nel merito del PeerId, questo non è altro che un identificatore univoco e verificabile associato ad un peer che non può essere falsificato a fronte di un banale rilevamento. In libp2p, i peer sono identificati dal loro PeerId, che è univoco a livello globale e che consente ad altri nodi di ottenere la chiave pubblica crittografica del peer a cui esso fa riferimento.

La forma più comune di PeerId è un *multihash* della chiave pubblica di un peer, che può essere utilizzata per recuperare l’intera chiave pubblica dalla DHT per la verifica della crittografia [18].

Un’importante proprietà delle identità crittografiche dei nodi è che sono disaccoppiate dal livello di trasporto, consentendo ai peer di verificare l’identità degli altri partecipanti indipendentemente dalla rete sottostante. Ciò conferisce loro la possibilità di mantenere la conservazione “viva” molto più a lungo rispetto agli identificatori basati sulla posizione (ad esempio, gli indirizzi IP), siccome le identità rimangono stabili durante i cambi di indirizzo.

È di fondamentale importanza essere in grado di inviare e ricevere informazioni tra peer in modo sicuro, il che significa potersi fidare dell’identità del nodo con cui si sta comunicando. Questo implica che nessuna terza parte deve poter leggere o modificare la conversazione tra due nodi durante lo scambio di messaggi. Libp2p supporta “l’aggiornamento” di una connessione fornita tramite trasporto in un canale cifrato sicuro. Il processo è flessibile e supporta più metodi di crittografia della comunicazione,



tra cui TLS 1.3 e Noise.

Quando si vuole inviare un messaggio a un peer, si ha la necessità di conoscere due informazioni chiave: il suo PeerId e come individuarlo sulla rete, in modo da aprire una connessione con esso. In diverse situazioni può verificarsi di essere a conoscenza del solo PeerId del nodo con cui si vuole comunicare. Vi è quindi la necessità di scoprire il suo indirizzo di rete. Il **peer routing** è il processo di scoperta degli indirizzi peer che sfrutta la conoscenza degli altri nodi. In un sistema di peer routing, un peer può fornire l'indirizzo d'interesse al richiedente nel caso in cui questo ne sia a conoscenza, oppure può inviare la richiesta ricevuta a un altro nodo che ha maggiori probabilità di conoscere la risposta. Con il progredire del numero di peer contattati, non solo aumentano le possibilità di trovare il nodo che si sta cercando, ma viene anche definita una visione più completa della rete all'interno delle tabelle di routing del client, il che consente di rispondere alle domande di routing degli altri peer in maniera ottimale. L'attuale implementazione del peer routing di libp2p utilizza una DHT per instradare iterativamente le richieste al PeerId desiderato più vicino, utilizzando l'algoritmo di routing **Kademlia** (discusso al paragrafo 2.4).

Inoltre, libp2p fornisce un'interfaccia di routing dei contenuti per recuperare i dati d'interesse senza la necessità di sapere quale sia il peer che li fornisce. Per questa implementazione viene utilizzata la stessa DHT basata su *Kademlia* utilizzata nel peer routing.

Lo scambio di messaggi tra i nodi è centrale per la maggior parte dei sistemi peer-to-peer. **Pubsub** è un modello utile per inviare un messaggio a gruppi di destinatari interessati a una determinata tematica. Libp2p definisce, infatti, un'interfaccia *pubsub* per l'invio di messaggi a tutti i peer iscritti a un determinato "argomento". L'interfaccia ha attualmente due implementazioni stabili: **floodsub**, che utilizza una strategia di "allagamento" della rete (semplice ma inefficiente), e **gossipsub**, che definisce un protocollo di gossip estensibile. Attualmente è in corso lo sviluppo di **episub**, un *gossipsub* ottimizzato per multicast a sorgente singola con un numero limitato di sorgenti fisse che trasmettono a un numero elevato di peer interessati ad un determinato argomento [19].

Ciò che rende libp2p particolarmente utile per le connessioni peer-to-peer è il **multiplexing** delle connessioni. Tipicamente, ogni servizio in un sistema apre una connessione diversa per comunicare in remoto con altri servizi dello stesso tipo. Usando IPFS, invece, viene aperta solo una connessione in cui si "*multiplexa*" tutto su di essa. Questo è utile in quanto la configurazione e la realizzazione di connessioni è un'attività solitamente difficile e costosa. Al contrario, con il multiplexing è possibile eseguire tutto ciò di cui si ha necessità una volta che la connessione è stata stabilita.

### 2.5.2 Bitswap

**Bitswap** è il protocollo basato su messaggi utilizzato da IPFS per lo scambio dei blocchi tra peer. Esso si occupa di gestire le richieste effettuate da un essere umano o da un'applicazione in merito al recupero di blocchi di dati dalla rete. Tutti i messaggi inviati sul network contengono una *lista dei desideri* o un set di blocchi. Al momento della ricezione di una lista dei desideri, un nodo IPFS si dovrebbe occupare dell'inoltro

al client dei blocchi richiesti, nel caso in cui ne sia in possesso. Successivamente alla loro ricezione, il nodo client dovrebbe inviare una notifica chiamata “*Cancel*” a significare che non necessita più di determinati blocchi, in quanto ottenuti.

Bitswap si occupa di gestire due flussi principali: *la richiesta* di blocchi da parte dei nodi e *l'inoltro* di blocchi ai peer. Le richieste di blocchi sono principalmente mediate dal **want-manager**, il quale comunica ai peer la richiesta di un blocco da parte di un nodo. Per un blocco che è stato richiesto, identificato tramite CID, viene invocato il metodo `Bitswap.GetBlock(cid)`. Questo permette di richiedere un CID sulla rete e, se viene ricevuto il blocco corrispondente, questo viene restituito. Più concretamente, `Bitswap.GetBlock(cid)` aggiunge un CID alla lista dei desideri di un nodo. Il want-manager aggiorna tutti i peer aggiungendo una nuova *entry* alla loro **coda dei messaggi**, i quali potrebbero o meno rispondere con il blocco desiderato.

Ogni peer attivo ha una *coda di messaggi* associata. La coda dei messaggi contiene l'informazione successiva da inviare ad un determinato nodo. Le code dei messaggi vengono aggiornate da due sottosistemi:

- **Gestore della lista dei desideri:** quando un CID viene aggiunto o rimosso da una lista dei desideri, è necessario aggiornare i relativi peer. Questi aggiornamenti vengono inviati a tutte le code di messaggi dei peer pertinenti.
- **Motore decisionale:** nel momento in cui si è in possesso di un blocco che un peer desidera e il motore decisionale ne determina l'inoltro, esso viene propagato alla coda di messaggi di quel peer.

I task worker controllano le code dei messaggi, eliminano dalla coda un messaggio in attesa e lo inviano al destinatario.

I fornitori di blocchi vengono gestiti principalmente dal **motore decisionale**, il quale definisce come le risorse devono essere allocate tra i peer. Quando viene ricevuto un messaggio contenente una lista dei desideri di un nodo, questo viene inviato al motore decisionale. Per ogni CID all'interno della lista di cui si possiede il blocco corrispondente, viene aggiunta *un'attività* alla *TaskQueue* del peer. **Un'attività** si considera completata nel momento in cui il blocco corrispondente è stato inviato alla coda dei messaggi per quel nodo.

La struttura dati principale del motore decisionale è la *coda di richieste dei peer* (**PRQ**). La PRQ aggiunge i peer a una *coda round-robin ponderata*, in cui i pesi si basano su una o più metriche specifiche del nodo. Attualmente, una strategia di Bitswap è una funzione il cui input è il *Ledger* di un peer e l'output è un peso per quel peer. Ai peer vengono quindi forniti i Task nelle rispettive TaskQueues. La quantità di dati che ad ogni peer viene fornita in un dato round del round-robin è determinata dal relativo peso nella coda.

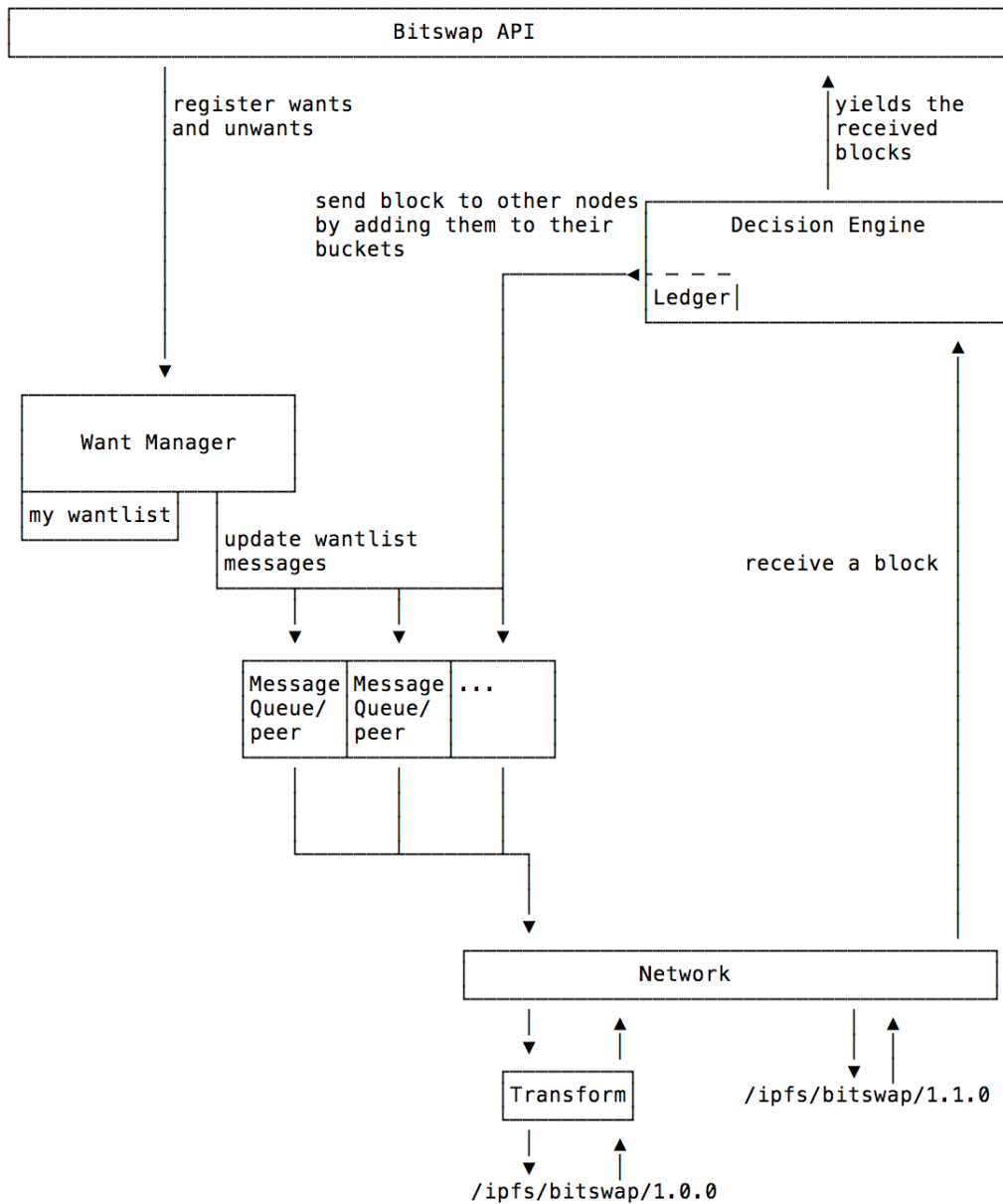


Figure 11: Bitswap Subsystems

I tipi seguenti vengono utilizzati nelle descrizioni dei sottosistemi Bitwap:

- CID: discussi al paragrafo 1.2;
- Peer: un'istanza Bitwap a cui si è connessi;
- Blocco: un blob binario;
- Messaggio: un messaggio Bitwap;

- Entry: una voce della lista dei desideri che può essere inclusa in un Messaggio quando si vuole aggiunge/rimuove un particolare CID da essa. Contiene:
  - Il **CID** riferito a un determinato blocco;
  - La **priorità** con cui l'utente desidera il CID (rilevante solo se *Cancel* è falso);
  - **Cancel**, un booleano che rappresenta se l'entry deve rimuovere o meno il CID dalla lista dei desideri.
- Ledger: un record dei dati aggregati scambiati tra due peer. Ogni peer memorizza un Ledger per ciascuno dei suoi peer.

Un singolo messaggio Bitswap può contenere la lista dei desideri del mittente oppure dei blocchi di dati. La lista inoltrata dal client può essere completa, ma allo stesso tempo può essere anche parziale, ovvero relativa alle sole modifiche che il destinatario deve conoscere. Per quanto riguarda i blocchi di dati, invece, questi sono intesi come i blocchi che il destinatario necessita (cioè, i blocchi che sono nella sua lista dei desideri al momento dell'invio da parte del mittente) [20].

Il formato wire di Bitswap è semplicemente uno stream di messaggi Bitswap. Il seguente *protobuf* descrive la forma di questi messaggi:

```
message Message {
  message Wantlist {
    message Entry {
      optional bytes block = 1; //the block key
      optional int32 priority = 2; //the priority (
        normalized). default to 1
      optional bool cancel = 3; //whether it revokes an
        entry
    }
    repeated Entry entries = 1; //a list of wantlist entries
    optional bool full = 2; //whether it is the full
      wantlist. default = false
  }
  message Block {
    bytes prefix = 1; //CID prefix (cid version, multicodec
      and multihash prefix (type + length)
    bytes data = 2;
  }
  Wantlist wantlist = 1;
  optional repeated bytes blocks = 2; //used to send Blocks
    (bitswap 1.0.0)
  repeated Block payload = 3; //used to send Blocks (bitswap
    1.1.0)
}
```

### 3 Install IPFS

IPFS è una raccolta di protocolli, pacchetti e specifiche che consentono ai dispositivi sui quali è in esecuzione di inviare e ricevere dati. Pertanto, gli utenti possono scegliere quali pacchetti installare quando utilizzano IPFS, siccome non esiste una soluzione adatta a chiunque: uno sviluppatore che realizza applicazioni di rete deciderà di installare un set di strumenti diverso da colui che desidera semplicemente archiviare file su IPFS [21].

#### 3.1 IPFS Desktop

Chiunque può utilizzare IPFS per **archiviare file** in modo *decentralizzato*. Il metodo più semplice per farlo è attraverso l'installazione dell'applicazione **IPFS Desktop**. Quest'app è stata realizzata utilizzando il framework *Electron* [22] e consente di interagire con la rete attraverso una semplice interfaccia utente.

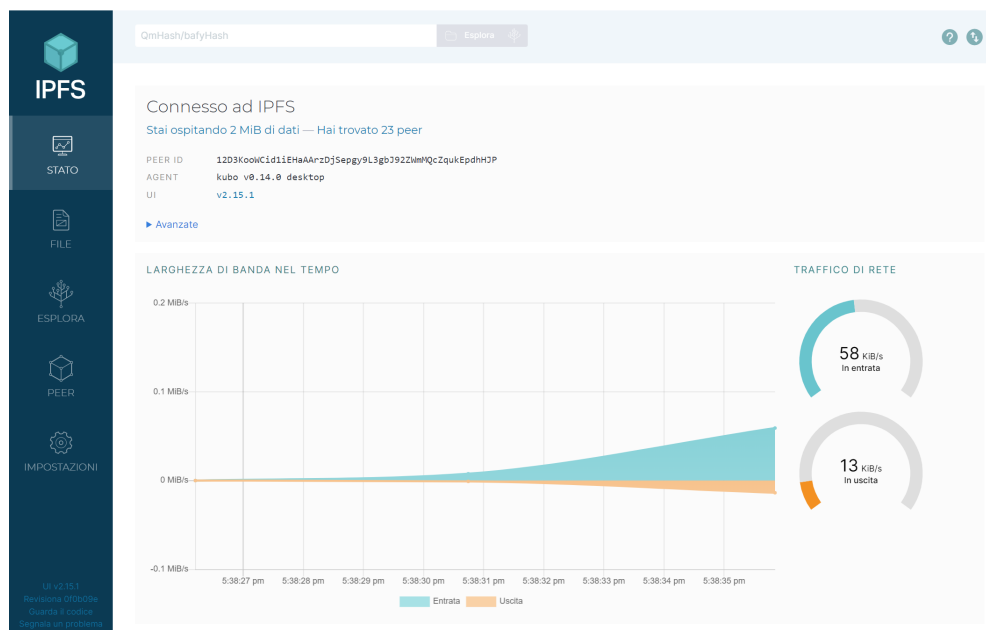


Figure 12: IPFS Destkop

*IPFS Desktop* riunisce sotto un'unica applicazione un nodo IPFS, un file manager, un peer manager e un esploratore di contenuti.

Nel caso in cui si sia già in possesso di un nodo IPFS sul proprio elaboratore, IPFS Desktop fungerà da pannello di controllo per quel nodo, nonché da browser di file. In caso contrario, si occuperà di installarne uno. In entrambe le casistiche, IPFS Desktop verificherà autonomamente la disponibilità di aggiornamenti [23].

IPFS Desktop mette, quindi, a disposizione diverse funzionalità, tra cui:

- L'avvio del nodo al booting del sistema (Mac/Windows);

- La gestione del nodo tramite menubar/taskbar del sistema operativo;
- L'importazione semplice e veloce di file e cartelle su IPFS (ad esempio, tramite trascinamento o click destro sul documento o sulla directory da importare);
- Un'agevole gestione dei contenuti del proprio nodo tramite un browser che offre diverse scorciatoie per rinominare e spostare file e cartelle, visualizzare in anteprima diversi formati di file direttamente in IPFS Desktop, copiare CID negli appunti e molto altro ancora;
- Un celere download per CID, percorsi IPFS e percorsi IPNS;
- La visualizzazione dei peer IPFS sparsi per il globo tramite una mappa che illustra a quali nodi si è connessi, dove questi si trovano, le connessioni in uso e molto altro ancora;
- L'esplorazione della “Foresta Merkle” dei file IPFS con un visualizzatore che consente di vedere in prima persona come i set di dati archiviati su IPFS sono suddivisi in parti.

### 3.1.1 Windows Installation

L'installazione di IPFS Desktop è differente a seconda del sistema operativo sul quale si intende eseguire il programma. Nel caso di **Windows**, è necessario recarsi al seguente indirizzo web, individuare il file con estensione **.exe**, fare click su di esso per avviarne il download e, una volta terminato, *lanciare* l'eseguibile per avviarne l'installazione. Per portare a completamento la procedura con successo sarà necessario indicare il percorso d'installazione dell'applicazione e per quali utenti si intende installare l'app. Una volta terminata la procedura è possibile utilizzare IPFS Desktop.

### 3.1.2 macOS Installation

Per quanto riguarda **macOS**, è necessario recarsi al seguente indirizzo web, individuare il file con estensione **.dmg**, fare click su di esso per avviarne il download e, una volta terminato, aprire il file e trascinare l'icona dell'app all'interno della cartella **Applications**. A questo punto sarà sufficiente lanciare IPFS Desktop.

### 3.1.3 Ubuntu Installation

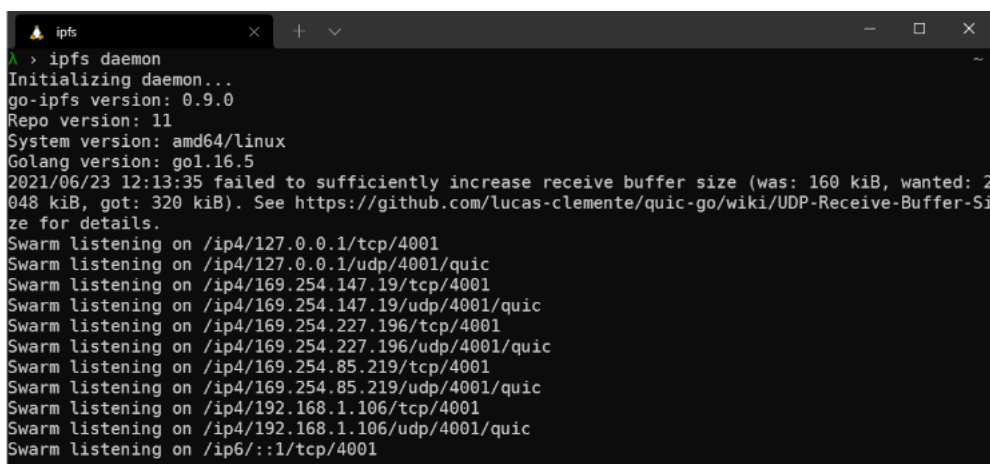
In quanto a **Ubuntu**, è necessario recarsi al seguente indirizzo web, individuare il file con estensione **.deb**, fare click su di esso per avviarne il download e, una volta terminato, lanciare con un doppio click l'eseguibile per avviarne l'installazione tramite l'*Ubuntu Software*. É possibile effettuare l'installazione dell'applicazione anche tramite linea di comando, spostandosi all'interno della directory in cui si è salvato l'eseguibile e digitando il comando `sudo dpkg -i ./ipfs-desktop-[version]-amd64.deb`, dove *version* va sostituito con il numero di versione del pacchetto IPFS appena scaricato.

Nonostante queste indicazioni su come effettuare l'installazione siano specifiche per *Ubuntu*, è probabile che siano compatibili anche con le altre distribuzioni **Linux**. In caso contrario, si consiglia di consultare la seguente pagina web.

## 3.2 Command-line

L'installazione di IPFS tramite riga di comando è vantaggiosa nel caso in cui si intenda creare applicazioni o servizi su un nodo IPFS. Inoltre, questa metodologia è utile anche nell'eventualità in cui si debba configurare un nodo sprovvisto di un'interfaccia utente, ad esempio server remoti o macchine virtuali. L'uso di IPFS tramite *CLI* consente di realizzare tutto ciò che permette di fare IPFS Desktop, ma a un livello più granulare.

Il team IPFS gestisce il sito **dist.ipfs.io** per consentire agli utenti di trovare rapidamente l'ultima versione di ogni pacchetto IPFS di loro interesse. Nel momento in cui una nuova versione di un pacchetto viene rilasciata, questa è automaticamente inserita all'interno della pagina web sopracitata [24].



```
> ipfs daemon
Initializing daemon...
go-ipfs version: 0.9.0
Repo version: 11
System version: amd64/linux
Golang version: go1.16.5
2021/06/23 12:13:35 failed to sufficiently increase receive buffer size (was: 160 kiB, wanted: 2048 kiB, got: 320 kiB). See https://github.com/lucas-clemente/quic-go/wiki/UDP-Receive-Buffer-Size-for-details.
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/127.0.0.1/udp/4001/quic
Swarm listening on /ip4/169.254.147.19/tcp/4001
Swarm listening on /ip4/169.254.147.19/udp/4001/quic
Swarm listening on /ip4/169.254.227.196/tcp/4001
Swarm listening on /ip4/169.254.227.196/udp/4001/quic
Swarm listening on /ip4/169.254.85.219/tcp/4001
Swarm listening on /ip4/169.254.85.219/udp/4001/quic
Swarm listening on /ip4/192.168.1.106/tcp/4001
Swarm listening on /ip4/192.168.1.106/udp/4001/quic
Swarm listening on /ip6:::1/tcp/4001
```

Figure 13: Command-line

### 3.2.1 Windows Installation

L'installazione tramite command-line di IPFS per il sistema operativo di *casa Microsoft* consiste nel download del binario di **Kubo** (la più recente e diffusa implementazione di IPFS) dal seguente sito. Successivamente è necessario estrarre i file contenuti nella cartella compressa appena scaricata e spostarli nella posizione desiderata all'interno del file system, muoversi all'interno della cartella contenente i file estratti e verificare se il comando `ipfs.exe --version` restituisce la versione di IPFS attualmente installata nell'elaboratore. A seguire, è necessario salvare l'attuale directory di lavoro in una variabile temporanea, creare un profilo PowerShell e aggiungere la posizione del Kubo daemon all'interno del path del PowerShell. Per finire, è sufficiente caricare il profilo appena creato e verificare che il percorso IPFS sia stato impostato correttamente, controllando dalla cartella *HOME* che venga restituita la versione di IPFS installata.

### 3.2.2 macOS Installation

L'installazione tramite command-line di IPFS per il sistema operativo **macOS** consiste nel download del binario di *Kubo* dal seguente sito. In seguito, è necessario estrarre i file/directory contenuti nella cartella compressa appena scaricata, muoversi all'interno del folder *kubo* appena ottenuto ed eseguire lo script di installazione, ovvero `sudo bash install.sh`. Infine, è sufficiente verificare che l'operazione sia stata completata con successo controllando la versione corrente installata di IPFS tramite il comando `ipfs --version`.

### 3.2.3 Linux Installation

L'installazione tramite command-line di IPFS per **Linux** prevede la stessa procedura da utilizzare per il sistema operativo di *casa Apple*. L'unica differenza riguarda il file binario da scaricare e installare, che sarà relativo al sistema operativo d'interesse.

## 3.3 IPFS Companion

**IPFS Companion** è un *add-on Browser* che consente di interagire con il proprio nodo e con la rete IPFS tramite il proprio motore di ricerca. Gli *add-on*, chiamati anche *componenti aggiuntivi*, sono piccoli programmi che ampliano la funzionalità di un browser. IPFS Companion è disponibile per Brave, Chrome, Edge, Firefox e Opera. Esso abilita il supporto agli indirizzi `ipfs://`, carica automaticamente siti web e percorsi di file da un **gateway IPFS**, consente di importare e condividere facilmente file con IPFS e molto altro ancora [25].

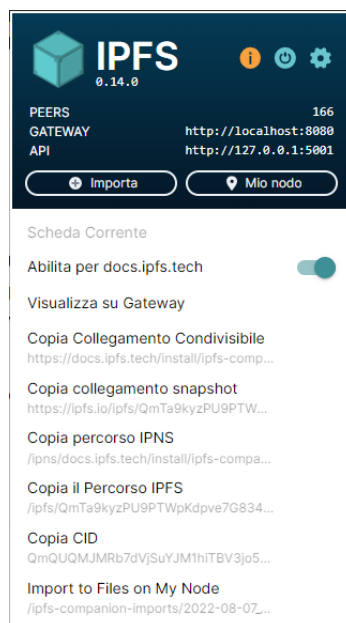


Figure 14: IPFS Companion



La distribuzione IPFS cerca di includere il supporto nativo a se stesso in tutti i browser e gli strumenti più diffusi. I *gateway* forniscono soluzioni alternative alle applicazioni che non supportano IPFS in modo nativo e che potrebbero incorrere in errori. Strumenti come **IPFS Companion** risolvono questi errori di accesso ai contenuti. I *gateway IPFS* forniscono un servizio basato su HTTP che consente a browser e strumenti che ignorano IPFS di accedere al suo contenuto. Qualsiasi gateway IPFS risolve l'accesso a un qualunque CID richiesto. Nel caso in cui si sia installato sulla propria macchina IPFS Desktop o una qualsiasi altra forma di un nodo IPFS, si dispone di un servizio gateway locale [26].

IPFS Companion funziona in simbiosi al nodo IPFS in esecuzione sul proprio computer locale, perciò è necessario assicurarsi di avere un nodo installato prima di utilizzare questo componente aggiuntivo.

Il modo più semplice per ottenere IPFS Companion è attraverso l'add-on store del browser. Di seguito si riportano i link per Mozilla Firefox e Google Chrome.

IPFS Companion mette a disposizione un'ampia gamma di funzionalità. Tra le più importanti è possibile individuare le seguenti:

- **Rileva gli indirizzi con percorsi IPFS**

IPFS Companion individua e verifica le richieste per percorsi IPFS, come `/ipfs/cid` o `/ipns/peerid_or_host-with-dnslink`, su qualsiasi browser sul quale è installato. Se un percorso è un indirizzo IPFS valido viene reindirizzato per essere caricato dal gateway locale, il quale converte i dati da un protocollo all'altro;

- **Rileva gli indirizzi abilitati per il DNSLink**

Il DNSLink è un protocollo semplice che collega contenuto e funzionalità del DNS sfruttandone l'architettura distribuita. IPFS Companion individua le informazioni DNSLink nei record DNS del browser. Se un sito utilizza DNSLink, IPFS Companion reindirizza la richiesta HTTP al gateway locale;

- **Rileva pagine con headers `x-ipfs-path`**

IPFS Companion aggiorna il trasporto a IPFS nel caso in cui trovi `x-ipfs-path` nell'header di una risposta HTTP. Questo funge da alternativa nei casi in cui un percorso IPFS non sia presente nell'URL;

- **Attiva/disattiva i reindirizzamenti a livello globale o per sito**

IPFS Companion permette di disabilitare e riattivare i reindirizzamenti del gateway locale;

- **Consente di accedere, dalla barra del browser, alle azioni IPFS utilizzate più di frequente**

IPFS Companion consente di verificare a quanti peer si è connessi, controllare lo stato del gateway, aggiungere facilmente risorse alla rete, importare/condividere velocemente una scheda del browser, copiare i percorsi o i CID dei documenti IPFS, lanciare la *IPFS Web UI dashboard*, attivare o disattivare i reindirizzamenti del gateway e, più in generale, tutte le sue funzionalità direttamente dal menu principale.

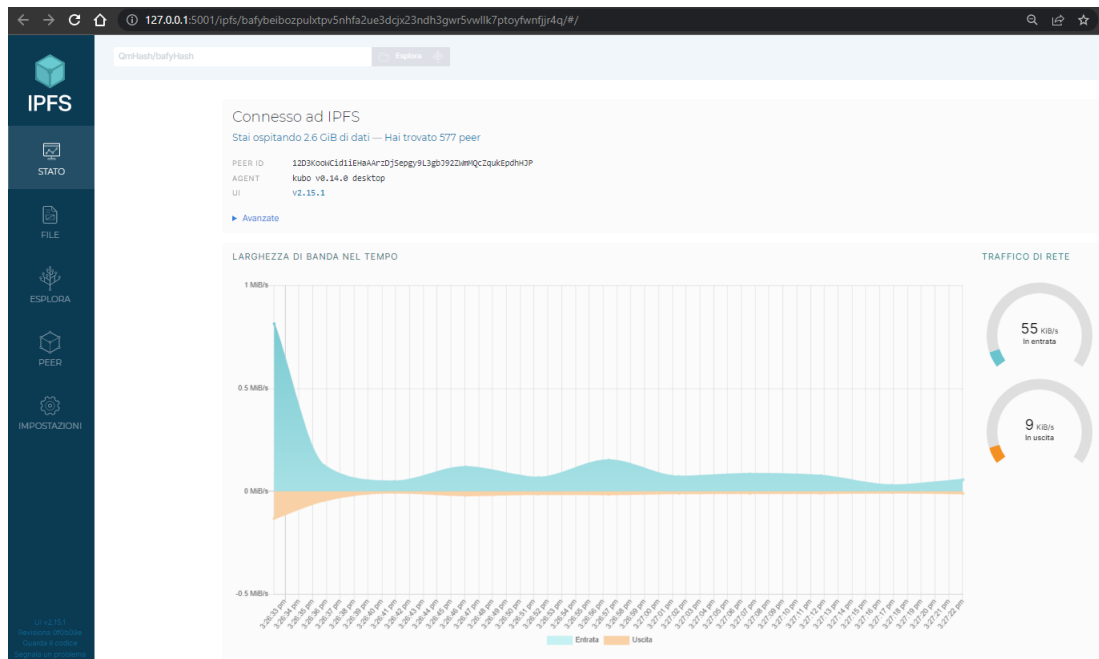


Figure 15: IPFS Web UI dashboard

### 3.4 Infrastruttura Server

Se si vuole installare IPFS in un **ambiente server**, è necessario approfondire la propria conoscenza in merito a *IPFS Cluster*. Il **cluster IPFS** orchestra i dati attraverso uno sciame di daemon IPFS tramite l’allocazione, la replica e il monitoraggio di un set di pin globale distribuito tra più peer. Questo semplifica notevolmente la gestione di più nodi e garantisce che i dati siano disponibili in una rete interna.

#### 3.4.1 Installation

In primis, è necessario verificare che **Docker** e **Docker Compose** siano installati nel sistema, controllandone la versione. Successivamente è necessario scaricare l’ultima versione del pacchetto `ipfs-cluster-ctl` dal sito `dist.ipfs.io`. Una volta terminato il download, si procede con l’estrazione dei file dall’interno della directory appena salva. Per completare l’operazione, è sufficiente scaricare il file `docker-compose.yml`, muoverlo all’interno della cartella `ipfs-cluster-ctl` e avviare il cluster attraverso il comando `docker-compose`.

## 4 Basics

In questo capitolo vengono trattate le operazioni e le configurazioni base di cui ha bisogno un nuovo utente per approcciarsi inizialmente a IPFS. Si è posta particolare attenzione sull'esecuzione dell'app **IPFS Desktop** e sull'interazione con IPFS tramite **riga di comando**.

### 4.1 Desktop app

Questa guida permette di esplorare le basi di IPFS Desktop, descrivendo come aggiungere, rimuovere e scaricare un file utilizzando la rete.

#### 4.1.1 Aggiungere un file

Una volta completata la procedura di installazione di *IPFS Desktop* sul proprio computer (descritta al paragrafo 3.1), è possibile aggiungere file e documenti sulla rete IPFS.

Innanzitutto, è necessario eseguire l'applicazione e assicurarsi di essere connessi a IPFS: per accertarsi di questo è sufficiente spostarsi nella schermata di stato facendo click sulla scheda **Status** sul lato sinistro dell'interfaccia. Nella sezione superiore della pagina sarà possibile verificare il proprio stato attuale.

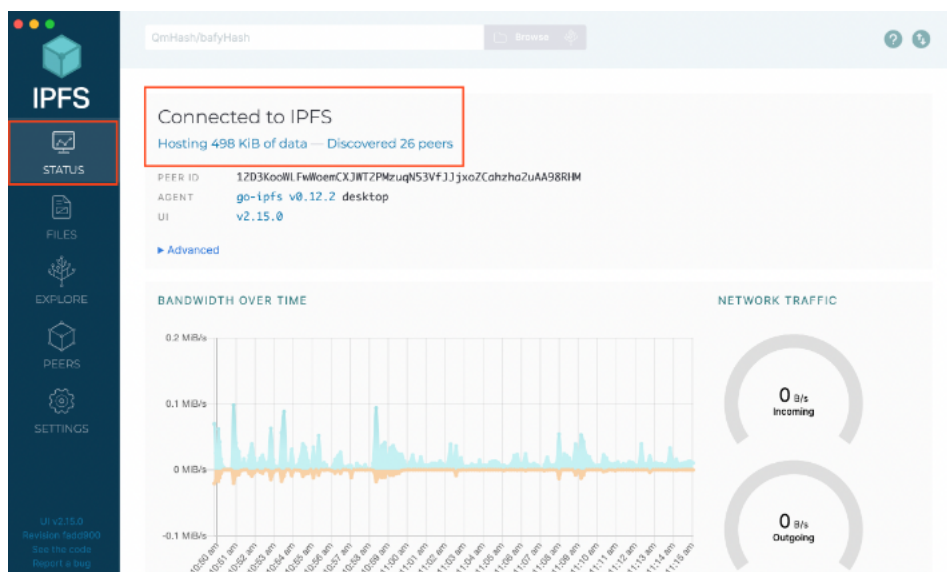


Figure 16: Controllo connessione a IPFS

A questo punto, se si risulta connessi alla rete, è possibile spostarsi nella pagina relativa ai file (scheda **Files**) e procedere con l'importazione di un documento o di un'intera cartella tramite la pressione del pulsante **Import**. Nella finestra di selezione, è ora possibile individuare il file o la directory che si desidera caricare e mantenere su IPFS. Tramite un semplice doppio click, la risorsa verrà importata sul proprio nodo IPFS

locale. Quando un documento viene importato nel nodo, viene calcolato l'identificatore di contenuto (**CID**) di quel file. I CID dei documenti memorizzati sul nodo vengono elencati direttamente sotto il nome del file nella pagina **Files**.

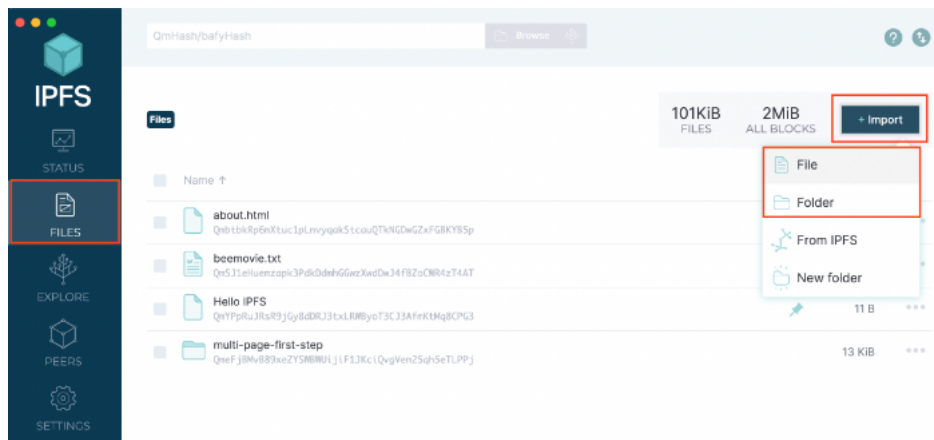


Figure 17: Aggiunta di un file sul proprio nodo

#### 4.1.2 Scaricare un file

Importare e scaricare un file remoto nella memoria locale del proprio dispositivo tramite IPFS è un compito semplice se si utilizza IPFS Desktop. In primo luogo, è necessario ottenere il **CID** del file o della directory che si desidera scaricare, aprire IPFS Desktop e recarsi nella schermata **Files**. A questo punto sarà sufficiente cliccare sul bottone **Import**, selezionare la voce **From IPFS** e inserire il CID della risorsa d'interesse nell'apposito spazio della finestra appena apertosi. Durante questa fase è anche possibile dare un nome al file o alla cartella che si sta scaricando. Per completare l'operazione sarà sufficiente fare click su **Import** in modo da recuperare automaticamente i metadati del file da IPFS.



Figure 18: Finestra *Import From IPFS*

Se la procedura si è conclusa con successo si possiede un **puntatore** al nuovo file sul proprio nodo locale. È importante sapere, però, che ciò non significa che il documento sia stato salvato nella memoria locale del proprio computer. Questa tipologia di importazione prende il nome di *lazy-loading*, il che significa che i dati effettivi vengono recuperati solo al momento della richiesta. Se si desidera scaricare e salvare una copia del file direttamente nella memoria del proprio dispositivo, è necessario cliccare sui tre punti accanto al nome della risorsa d'interesse. Nel menu a tendina che appare, selezionare la voce **Download** e indicare il nome e la destinazione di salvataggio del file. Così facendo si ha una copia della risorsa salvata nella memoria locale del proprio dispositivo e per accedervi non è necessaria alcuna connessione a IPFS.

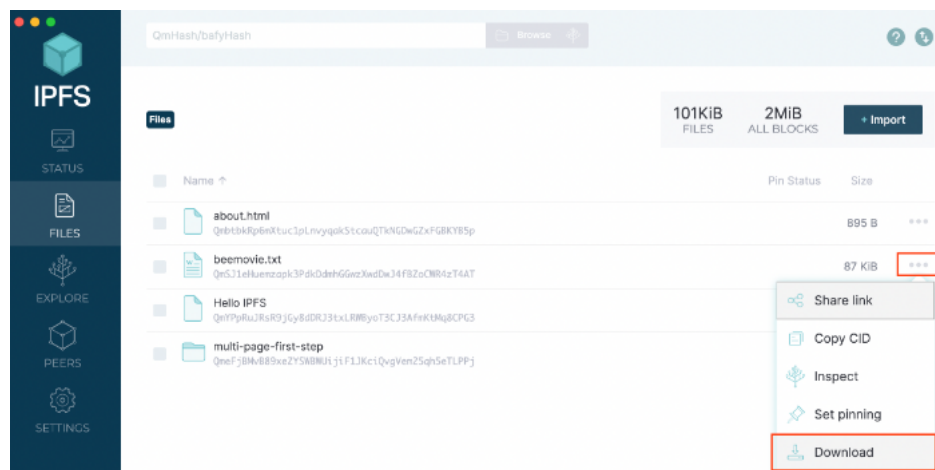


Figure 19: Download di un file da IPFS

### 4.1.3 Condividere un file

La condivisione di un file su IPFS è un'operazione piuttosto semplice. È sufficiente individuare la risorsa che si desidera condividere, copiarne il *CID* e inoltrarlo alla persona con la quale si desidera condividerlo. A questo punto, il ricevente potrà effettuare il download per acquisire il file utilizzando la rete IPFS.

Per prima cosa, è necessario spostarsi nella scheda **Files**, individuare il documento che si vuole condividere, fare click sui tre punti accanto al nome del file e selezionare la voce **Copia CID**, in modo da copiarne l'identificatore di contenuto. A questo punto sarà sufficiente inoltrare alla persona interessata il CID appena copiato.

Il metodo precedentemente discusso richiede che il ricevente sia dotato di IPFS Desktop al fine di poter recuperare la risorsa d'interesse. Per ovviare a questa specifica, è presente anche l'opzione **Share link** che permette di copiare il CID del file insieme a un URL del gateway locale. Così facendo, è possibile condividere con chiunque la risorsa, anche se il destinatario non è in possesso di IPFS Desktop.

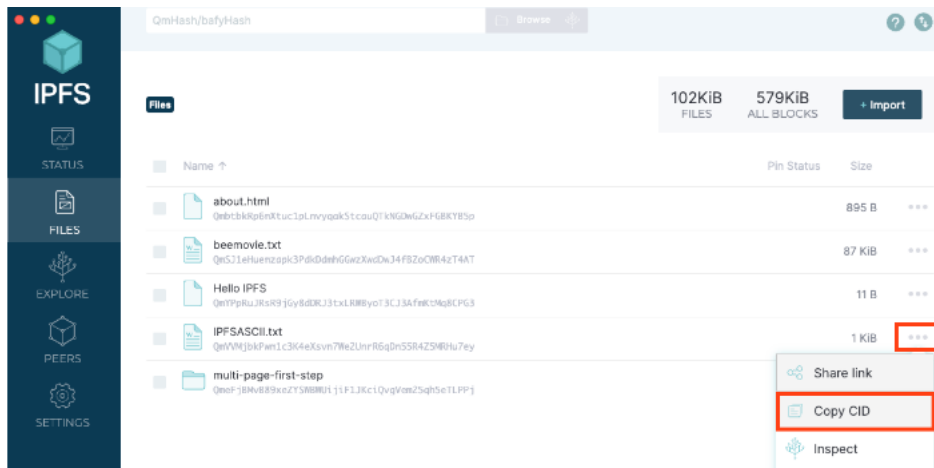


Figure 20: Condivisione di un file dal proprio nodo

#### 4.1.4 Rimuovere un file

La rimozione di un file dal nodo locale non rimuoverà necessariamente la risorsa dalla rete IPFS. Nel caso in cui il proprio file sia stato scaricato da alcuni peer presenti sul network, questi potrebbero continuare a renderlo disponibile per l'importazione e il download ad altri. Proprio come per il caricamento di un file su un normale server web, non c'è modo di garantire che non ci siano sue copie sulla rete.

La procedura di rimozione ha inizio con l'avvio dell'applicazione IPFS Desktop. É poi necessario spostarsi nella scheda dei **Files** e aprire il menù a scomparsa cliccando sui tre punti a fianco della risorsa che si intende rimuovere. Per completare l'iter, sarà sufficiente selezionare la voce **Remove** e confermarne la rimozione dalla schermata di conferma che verrà aperta.

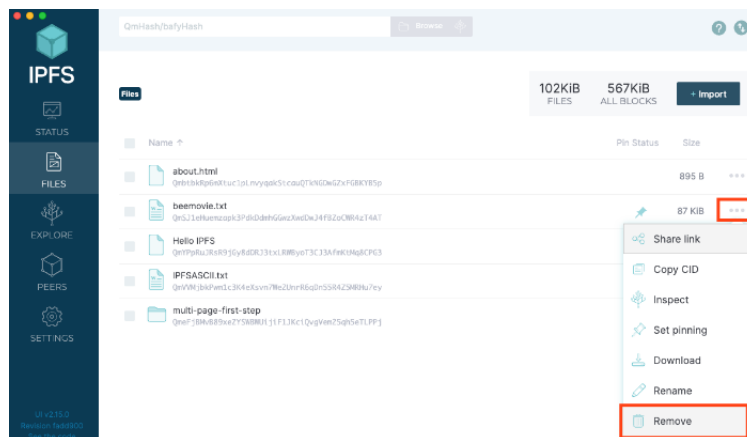


Figure 21: Rimozione di un file dal proprio nodo

Il **Garbage Collector (GC)** integrato in IPFS Desktop permette di recuperare la memoria occupata da oggetti che non sono più in uso. IPFS utilizza un Garbage Collector per liberare spazio su disco nel nodo IPFS eliminando i dati che ritiene non siano più necessari o che sono stati contrassegnati come tali [27].

Un nodo IPFS può proteggere i dati dal GC andando ad aggiungere un **pin locale** alla risorsa che si vuole preservare. Questa tecnica funziona con ogni tipo di dato e può essere messa in pratica sia manualmente che automaticamente (nel secondo caso, nel momento in cui si aggiunge un file utilizzando il comando **ipfs add** tramite CLI).

Nella finestra di conferma della rimozione è quindi importante assicurarsi di selezionare anche la voce **Also remove local pin** nel caso in cui il file sia *pinnato*.

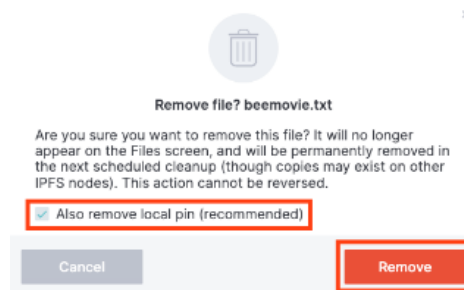


Figure 22: Conferma rimozione risorsa

Il GC viene eseguito automaticamente ogni ora per impostazione predefinita, rimuovendo tutti i file contrassegnati per l'eliminazione. Tuttavia, è possibile eseguirlo anche manualmente selezionando dalla taskbar del proprio sistema operativo l'icona di IPFS e selezionando **Run Garbage Collector** dalla sezione **Advanced**.

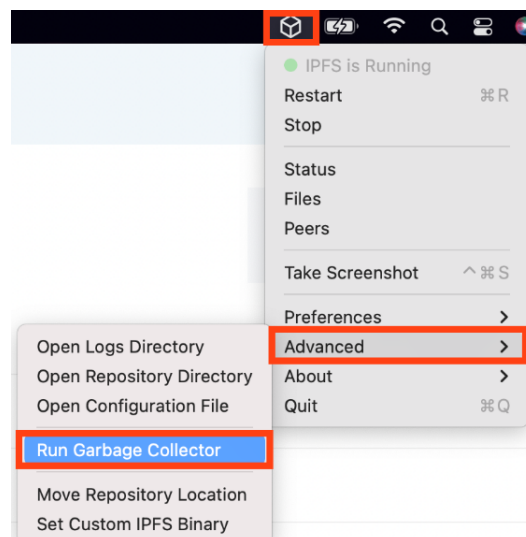


Figure 23: Esecuzione manuale del Garbage Collector

#### 4.1.5 Single e multi-page website su IPFS

L'infrastruttura IPFS può essere utilizzata per ospitare pagine web all'interno della sua rete distribuita. Di seguito viene descritta la procedura da seguire per rendere disponibile, tramite IPFS, un semplice sito web composto da un'unica pagina.

Completata l'installazione di *IPFS Desktop* sul proprio elaboratore, è necessario importare sulla rete il file relativo alla propria webpage. A tal proposito, è necessario eseguire l'applicazione IPFS Desktop e recarsi nella sezione **Files**, selezionando successivamente la voce *Add File*. Verrà aperta una nuova finestra in cui sarà possibile navigare tra le risorse del proprio dispositivo, individuando il file d'interesse (in questo caso `index.html`). Per portare a termine l'operazione d'importazione, fare click su *Open* una volta selezionato il documento.

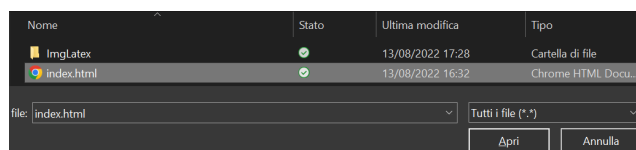


Figure 24: Importazione di `index.html`

Quando la risorsa sarà stata caritata su IPFS, sarà possibile recuperarne l'indirizzo per la visualizzazione sul proprio browser: tramite un semplice click sui tre puntini che seguono il nome del file all'interno della UI, selezionare la voce **Share link**. A questo punto non resta che aprire il proprio motore di ricerca e incollare nella barra degli indirizzi l'**URL** appena recuperato. Il browser dovrebbe caricare il sito web in pochi istanti, anche se il primo caricamento potrebbe richiedere alcuni minuti [28] (questo può dipendere dalla velocità della propria connessione Internet e della complessità del sito web).

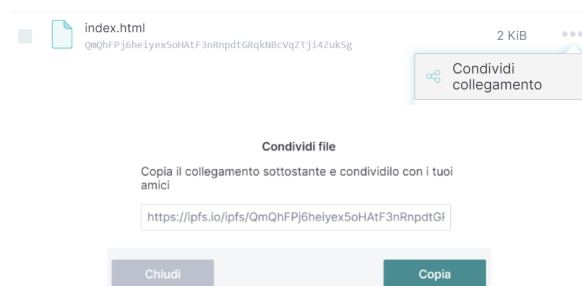


Figure 25: Recupero URL per visualizzazione tramite Browser

Allo stesso modo, IPFS permette di ospitare un sito composto da più pagine web: al posto d'importare sulla rete i file singolarmente, è possibile aggiungere **l'intera cartella del progetto** tramite l'app IPFS Desktop. Seguendo gli step precedentemente descritti, sarà possibile navigare tra le pagine del proprio sito web senza alcuna difficoltà.



## 4.2 CLI Operations

In questa sezione si introducono le basi per l'utilizzo di IPFS tramite **Kubo CLI**, soffermandosi sull'aggiunta, sul recupero, sulla lettura e sulla rimozione di risorse tramite linea di comando. Come anticipato in precedenza, *Kubo* è solo una delle molteplici implementazioni IPFS (la più diffusa). Al seguente link è possibile esplorare tutte le principali implementazioni disponibili in rete.

Tutte le istruzioni e gli esempi mostrati di seguito sono stati eseguiti e testati su un **Windows**. Tuttavia, i comandi IPFS sono gli stessi anche sui sistemi operativi Linux e macOS.

### 4.2.1 Aggiungere un file

Terminata la procedura d'installazione di *Kubo CLI* sul proprio dispositivo (descritta al paragrafo 3.2), è essenziale rendere operativo il proprio nodo. Nel caso in cui non fosse mai stato eseguito Kubo prima d'ora, è necessario inizializzare i file di configurazione IPFS tramite il comando `ipfs init`. A questo punto è possibile avviare il *demone IPFS* tramite il comando `ipfs daemon` per far sì che il proprio nodo si connetta alla rete. Questa istruzione rimarrà in esecuzione fino a quando non verrà specificato diversamente. È di fondamentale importanza non chiudere la CLI che si è utilizzata per inizializzare il demone in quanto la sua interruzione provocherebbe la disconnessione del peer dal network IPFS. Per proseguire con la procedura è quindi sufficiente aprire un secondo terminale.

```
ipfs daemon

> Initializing daemon...
> API server listening on /ip4/127.0.0.1/tcp/5001
> Gateway server listening on /ip4/127.0.0.1/tcp/8080
```

Figure 26: Connessione del nodo IPFS alla rete

Ora che il nodo è attivo e funzionante, è possibile aggiungere risorse a IPFS. Nella seconda CLI, muoversi all'interno della directory contenente il file o la cartella che si desidera memorizzare e condividere sul nodo. Nell'esempio sotto esame, ci si sposta all'interno del folder **Documents**, dov'è possibile individuare il file che si vuole importare sulla rete, ovvero **hello-ipfs.txt**. Il comando `ipfs add` permette di aggiungere il file a IPFS. Durante questa operazione, è necessario assicurarsi di aggiungere l'estensione del file alla fine del nome del documento da importare: `ipfs add hello-ipfs.txt`. Al termine dell'operazione, nel caso in cui la procedura si sia conclusa con successo, verrà restituito un messaggio di avvenuto caricamento.

```
added QmRgR7Bpa9xDMUNGiKaARvFL9MmnoFyd86rF817EzyfdGE hello-ipfs.txt
6 B / 6 B [=====] 100.00
```

Figure 27: Aggiunta di **hello-ipfs.txt** su IPFS tramite CLI

Recuperare file remoti da IPFS e salvarli sul proprio computer non è un'operazione complessa. In questo paragrafo si definisce come recupereremo una cartella al cui interno è contenuto un file di testo.

In primis, è necessario spostarsi tramite linea di comando nell'area di memoria in cui si desidera salvare la directory, in questo caso **Documents**. Successivamente, come si è anticipato nei capitoli precedenti, vi è la necessità di conoscere e specificare il CID relativo alla risorsa che si vuole recuperare: per far ciò, è necessario indicare al *demone ipfs* il CID d'interesse (**bafybeif2ewg3nqa33mjokpxii36jj2ywfqjpy3urdh7v6vqyfjoocvgy3a**). Tramite il comando **ipfs get**, abbinato all'identificatore di contenuto sopracitato, è possibile recuperare la cartella a cui quel CID fa riferimento. Al termine dell'operazione, nel caso in cui la procedura si sia conclusa con successo, l'output ottenuto sarà il seguente:

Figure 28: Recupero di una cartella tramite CLI

Per visualizzare il contenuto di un file tramite la *command-line* è sufficiente utilizzare il comando `ipfs cat`. È importante prestare attenzione al fatto che eseguendo da CLI `ipfs cat bafybeif2ewg3nqa33mjokpxii36jj2ywfqjpy3urdh7v6vqyfjoocvgy3a` verrà restituito un messaggio di errore: questo perché il CID citato in precedenza fa riferimento ad una **directory**, e non a un **file**. Nel caso in cui si voglia visualizzare il contenuto di una cartella, è necessario utilizzare il comando `ipfs refs <CID>`. L'istruzione `refs` restituisce il CID che punta al file che si trova all'intero della directory. In questo caso, infatti, l'output ottenuto è `bafkreig24ijzqxj3cxdp6yh6ia2ysxzvxsfn6rzahxxjv6ofcuix52wtq`. A questo punto è possibile esplorare il contenuto del file tramite il comando `ipfs cat` seguito dal CID appena estratto, ottenendo il risultato riportato di seguito.

[illegible]

Figure 29: Visualizzazione del contenuto di un file tramite CLI

#### 4.2.4 Bloccare un file

Come anticipato nel paragrafo 4.1.4, è possibile *pinnare* (ovvero, bloccare) sul proprio nodo IPFS i dati per i quali si vuole avere garanzia di disponibilità, in modo da assicurarsi che questi non vengano rimossi dal *Garbage Collector*. A tal proposito, è sufficiente utilizzare il comando `ipfs pin add <CID>` rispetto alla risorsa che si vuole preservare. Di default, aggiungere un file a IPFS tramite il comando `ipfs add` ne comporta il *pinning* automatico da parte del proprio nodo. Il comando `pin`, utilizzato congiuntamente a `ipfs add`, fa riferimento a una variabile booleana che assume come valore predefinito `true` [29]. Nonostante ciò, in questa sezione si vuole enfatizzare il *pinning* e come utilizzarlo in contesti differenti.

```
pinned bafybeif2ewg3nqa33mjokpxii36jj2ywfqjpy3urdh7v6vqyfjoocvgy3a recursively
```

Figure 30: *Pinning* di una directory tramite CLI

Come si evince dall'output ottenuto, il pin aggiunto è **ricorsivo** in quanto l'identificatore di contenuto specificato fa riferimento a una directory, il che significa che si vuole proteggere sia la cartella che il suo contenuto. Per impostazione predefinita, le risorse recuperate su IPFS non vengono pinnate al proprio nodo. Se si desidera evitare che i file vengano eliminati dal Garbage Collector, è necessario bloccarli seguendo la procedura appena descritta.

#### 4.2.5 Rimuovere un file

Nell'eventualità in cui si decida di non voler più rendere disponibile sulla rete IPFS un determinato file, è sufficiente rimuovere il *pin* da esso.

Imanzitutto, è necessario recuperare il CID della risorsa che si vuole rimuovere. Per visualizzare l'elenco dei contenuti presenti sul proprio nodo sotto forma di CID è necessario eseguire il comando `ipfs pin ls`.

Se si è a conoscenza del CID associato al file che si intende rimuovere, è possibile procedere con il suo *unpinning* tramite il comando `ipfs pin rm <CID>`, in modo da rendere possibile la sua eliminazione attraverso l'uso del Garbage Collector. In caso contrario, può essere utilizzato nuovamente il comando `ipfs add`, rispetto al file o alla cartella che si desidera rimuovere, per venirne a conoscenza. Nell'esempio in essere, si procede alla rimozione del pin dal file `hello-ipfs.txt`, aggiunto al nodo in precedenza. Il CID facente riferimento ad esso è `QmRgR7Bpa9xDMUNGiKaARvFL9MmnoFyd86rF817EZYfdGE`, ottenendo dalla procedura appena descritta quanto segue:

```
ipfs pin rm QmRgR7Bpa9xDMUNGiKaARvFL9MmnoFyd86rF817EZYfdGE
unpinned QmRgR7Bpa9xDMUNGiKaARvFL9MmnoFyd86rF817EZYfdGE
```

Figure 31: *Unpinning* di un file tramite CLI per rimozione tramite *GC*

A questo punto il file è stato sbloccato, ma è ancora presente sul nodo. Per rimuoverlo completamente, è sufficiente eseguire il Garbage Collector tramite il comando `ipfs repo gc`: così facendo, tutte le risorse sprovviste di pin verranno rimosse al fine di liberare spazio sul nodo.

Se una copia della risorsa appena rimossa era stata salva anche nella memoria locale del proprio dispositivo, questa non subirà alcun effetto dalla procedura appena descritta. Se si desidera eliminarla anche dal proprio supporto di memorizzazione, è necessario individuare il luogo in cui risiede file e procedere con la sua cancellazione tramite il classico iter implementato dal proprio sistema operativo.

## 5 Java-ipfs-HTTP-client

Come anticipato nei capitoli precedenti, non esiste un'implementazione canonica di IPFS. Il progetto assume un'ampiezza tale da poter disporre di una pletora di implementazioni con cui esplorare e sperimentare vari casi d'uso.

In questo capitolo si approfondisce l'API **java-ipfs-HTTP-client**, una libreria minore che consente di creare un *client Java* con cui inviare richieste HTTP per aggiungere, scaricare e recuperare file da IPFS. A tal proposito, viene realizzata una semplice applicazione sfruttando il framework **Spring Boot**.

### 5.1 Spring

Riprendendo la definizione data da *Wikipedia* [30], **Spring** è un framework open source per lo sviluppo di applicazioni su piattaforma Java.

Il framework nasce nel 2002, ad opera di Rod Johnson, con l'obiettivo di proporre un'alternativa leggera e flessibile allo standard EJB (Enterprise JavaBeans) per la realizzazione di applicazioni di livello enterprise. Nel corso degli anni, il progetto è stato affiancato da altri progetti satellite che hanno aggiunto utilità e funzionalità al framework, tra cui *Spring Security*, *Spring Data*, *Spring Cloud*, ecc...

Con l'evolversi del progetto, padroneggiare i molteplici aspetti del framework è diventato sempre più complesso. Questo ha spinto il team di Spring a pensare a un nuovo approccio che ne riducesse la curva di apprendimento e ne facilitasse l'adozione, dando così origine a **Spring Boot**.

La prima milestone release di *Spring Boot* risale al 2013. Spring Boot non rappresenta un nuovo framework per la creazione di applicazioni, bensì una sorta di "sovrastuttura" che permette di sfruttare a pieno gli strumenti forniti da Spring con il minimo dello sforzo da parte dello sviluppatore. In particolare, le difficoltà nel configurare un nuovo progetto, tipiche di Spring, vengono mitigate grazie a una serie di configurazioni di default basate su un approccio *opinionated*: Spring Boot, infatti, si occupa di configurare automaticamente tutto ciò che tipicamente viene utilizzato in un'applicazione basata su Spring, mentre per qualunque personalizzazione che esca dallo standard possono essere implementate logiche custom dallo sviluppatore.

Le applicazioni basate su Spring Boot sono applicazioni **stand-alone**, eseguibili tramite il metodo `main`. La *build* genera un pacchetto eseguibile che include una versione embedded del web container (di default, **Tomcat**). Questo toglie al developer l'incombenza di dover installare e configurare un web server sia sulla macchina di sviluppo che su quella di produzione [31].

### 5.1.1 Spring Initializr

La creazione dell'applicativo parte da **Spring Initializr**, un tool che permette di generare un progetto *Spring Boot* di partenza, con dipendenze e configurazioni di base. La procedura ha inizio con la selezione dello strumento di build automation tra quelli disponibili (in questo caso **Maven**), il linguaggio da utilizzare nel progetto (Java) e la versione di Spring Boot tra quelle proposte (v2.7.2). Successivamente si procede con la definizione del Group ID e dell'Artifact ID, per poi aggiungere le dipendenze dell'applicazione tramite l'apposito campo che ne consente la ricerca: per questo elaborato è utile aggiungere il pacchetto **Web** (corrispondente a `spring-boot-starter-web`) in quanto si vuole realizzare un semplice *REST web service* utilizzando gli strumenti messi a disposizione da Spring. Una volta definiti tutti i parametri di configurazione è possibile scaricare l'archivio *zip* contenente i file del progetto cliccando su *Generate Project*. Recuperato il progetto generato da Spring Initializr, è necessario estrarre il contenuto dell'archivio appena salvato.

**REST** (REpresentational State Transfer) è un modello architetturale, descritto per la prima volta nel 2000 da Roy Fielding, che permette la comunicazione/interoperabilità tra sistemi remoti, tipicamente connessi ad Internet. Il modello REST viene tipicamente implementato tramite il protocollo HTTP, dove i rispettivi metodi vengono utilizzati per definire le azioni da compiere sulle risorse [32].

In prima istanza, è necessario aggiungere all'interno del `pom.xml` (file utilizzato da Maven per reperire le dipendenze del progetto) gli aspetti legati alla compilazione e alla produzione del pacchetto per il deploy dell'applicazione:

```
<repository>
  <id>jitpack.io</id>
  <url>https://jitpack.io</url>
</repository>

<dependency>
  <groupId>com.github.ipfs</groupId>
  <artifactId>java-ipfs-http-client</artifactId>
  <version>$LATEST_VERSION</version>
</dependency>
```

La variabile `$LATEST_VERSION` deve essere sostituita con la versione più recente della libreria sotto esame, in questo caso la `v1.3.3`.

## 5.2 Struttura del progetto

### 5.2.1 Main Class

La prima cosa che si può notare esplorando il metodo `main` auto-generato è l'annotazione `@SpringBootApplication`.

```
@SpringBootApplication
public class ApplicationIpfsDemo {
    public static void main(String[] args) {
        SpringApplication.run(ApplicationIpfsDemo.class, args);
    }
}
```

Questa viene utilizzata per lanciare l'applicazione e ingloba diverse funzionalità importanti:

- Abilita le auto-configurazioni per le componenti Spring del progetto;
- Abilita la definizione di configurazioni all'interno della classe annotata;
- Abilita la scansione e la registrazione delle classi annotate come `@Component` presenti all'interno del package corrente e/o dei suoi figli.

### 5.2.2 IPFSConfig Class

A questo punto è possibile procedere con la creazione di una nuova classe, `IPFSConfig`, nella quale verrà definita l'istanza di IPFS relativa **al proprio nodo locale**. A questa classe viene assegnata l'annotazione `@Component`, definendone anche lo `@Scope` come `Singleton`.

Come anticipato, grazie alla keyword `@Component`, Spring rileva automaticamente i **bean** che sono stati definiti all'interno del progetto, ovvero gli oggetti che costituiscono l'architettura dell'applicazione e che sono gestiti dal container **Spring IoC** (l'ecosistema all'interno del quale le applicazioni Spring vivono). Lo *IoC container* si occupa di creare un'istanza per ognuno di essi e di iniettarvi tutte le dipendenze specificate senza dover scrivere alcun codice specifico.

Nel momento della creazione di un nuovo bean, è come se venisse creata una “ricetta” che consente di generare istanze della classe descritte da quel bean. Quando un bean viene definito come `Singleton`, come in questo caso, verrà gestita solo un'unica sua istanza condivisa. Tutte le richieste con uno o più ID corrispondenti a quella definizione di bean risulteranno nella restituzione di quella specifica istanza da parte del contenitore Spring. Più semplicemente, quando si definisce lo scope di un bean come *Singleton*, il contenitore Spring IoC si occuperà di creare esattamente una singola istanza dell'oggetto in essere. Le istanze create saranno disponibili per essere iniettate in altri bean, in base alle dipendenze che vengono dichiarate.

```

@Component
@Scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)
public class IPFSConfig {

    IPFS ipfs;

    public IPFSConfig(){
        ipfs = new IPFS("/ip4/127.0.0.1/tcp/5001");
    }
}

```

### 5.2.3 IPFSService Class

Definita la classe contenente i parametri di configurazione per il proprio nodo IPFS locale, è possibile proseguire con la creazione della classe **IPFSService**, nella quale viene gestita la logica di business dell'applicazione. Questa *componente* è identificata dall'annotazione **@Service** e ha la funzione di elaborare i dati e di fornirli al **Controller** per essere esposti verso il client. All'interno di questa classe si implementano i due metodi core del web service:

- **saveFile**: grazie a esso è possibile aggiungere un file sulla rete IPFS. La risorsa verrà caricata sul nodo a cui si è connessi e memorizzata nel suo archivio dati locale. Questa operazione restituisce l'identificatore univoco del file che viene caricato sulla rete, ovvero il suo **multihash**. Come anticipato nel paragrafo 2.1.2, Multihash è un hash autodescrittivo utilizzato per identificare in modo univoco gli oggetti e individuarli nel Merkle Tree di IPFS, tipicamente rappresentato in Base58. Quando si aggiunge una risorsa a IPFS, si ottiene come risposta un nuovo ramo del Merkle Tree ad essa dedicato, composto da uno o più oggetti collegati. Il metodo restituisce un elenco di *MerkleNode* che rappresenta il contenuto degli oggetti indirizzabili che sono stati appena aggiunti alla rete IPFS.

```

public String saveFile(MultipartFile file) {
    try {
        InputStream stream = new ByteArrayInputStream(
            file.getBytes());
        NamedStreamable.InputStreamWrapper
            inputStreamWrapper = new NamedStreamable.
                InputStreamWrapper(stream);
        IPFS ipfs = ipfsConfig.ipfs;
        MerkleNode merkleNode = ipfs.add(
            inputStreamWrapper).get(0);
        return merkleNode.hash.toBase58();
    } catch (Exception e) {
        throw new RuntimeException("Error during the
            communication with the IPFS node: ", e); } }

```



- **loadFile**: per recuperare un determinato file sulla rete IPFS, è necessario fornire all'applicativo l'hash dell'oggetto che vogliamo recuperare. IPFS identifica e recupera il file dal peer più vicino che lo ospita (il proprio nodo locale) attraverso la rete peer-to-peer e la tabella hash distribuita (DHT). La procedura più comune per trovare e leggere il contenuto di un determinato hash dalla rete IPFS è mediante il metodo `ipfs.cat()`.

```
public byte[] loadFile(String hash) {
    try{
        IPFS ipfs = ipfsConfig.ipfs;
        Multihash filePointer = Multihash.fromBase58(
            hash);
        return ipfs.cat(filePointer);
    } catch (Exception e) {
        throw new RuntimeException("Error during the
            communication with the IPFS node: ", e);
    }
}
```

Una volta stabilito come dichiarare i bean all'interno del progetto, è necessario comprendere come i singoli componenti possono cooperare nel modo corretto.

Per comunicare a Spring che una certa dipendenza deve essere recuperata dal container e assegnata ad uno specifico oggetto, viene utilizzata l'annotazione `@Autowired`. Questa può essere applicata a diversi elementi di una classe: un campo, un metodo o un costruttore. A prescindere da ciò, il suo compito è quello di indicare a Spring **quali sono le dipendenze richieste da un determinato oggetto**. Queste vengono ricercate tra le istanze presenti nello IoC container e, se presenti, vengono iniettate autonomamente.

```
@Autowired
private IPFSConfig ipfsConfig;
```

In questo modo, l'istanza di `IPFSConfig`, dichiarata all'interno della classe `IPFSService`, può essere utilizzata per soddisfare ulteriori dipendenze.

#### 5.2.4 IPFSController Class

Arrivati questo punto, non rimane altro che definire il **RestController**. In un web service REST le risorse sono identificate da URI; le richieste HTTP dirette verso questi indirizzi permettono di interagire con le risorse esposte e di effettuare su di esse diverse operazioni (ad esempio, **CRUD**, ovvero creazione, lettura, aggiornamento e rimozione). Spring permette di gestire le richieste inviate al server, proprio come per una tipica applicazione web, attraverso degli oggetti di tipo **Controller**. Nel caso specifico di un web service, è possibile sfruttare una tipologia di componente specifica che fornisce un aiuto nell'implementazione dei *resource controller*: questi componenti sono identificati dall'annotazione `@RestController`.

Questa annotazione rappresenta una specializzazione di `@Controller`, la quale permette di annotare tutti gli *handler method* del Controller con `@ResponseBody`, annotazione che permette di effettuare il binding tra il corpo della richiesta e l'argomento di un handler.

Si procede, dunque, con la creazione della classe adibita a questo compito, ovvero `IPFSController`. Al suo interno è possibile individuare i seguenti metodi:

```
@PostMapping(value = "upload")
public String saveFile(@RequestParam("file") MultipartFile
    file){
    return ipfsService.saveFile(file);
}

@GetMapping(value = "file/{hash}")
public ResponseEntity<byte []> loadFile(@PathVariable("hash")
    String hash){
    HttpHeaders httpHeaders = new HttpHeaders();
    httpHeaders.add("Content-type", MediaType.ALL_VALUE);
    byte [] bytes = ipfsService.loadFile(hash);
    return ResponseEntity.status(HttpStatus.OK).headers(
        httpHeaders).body(bytes);
}
```

Il mapping delle richieste HTTP avviene utilizzando l'annotazione `@RequestMapping` (o una delle sue specializzazioni: `@GetMapping`, `@PostMapping`, ecc...), che consente di specificare un **URI template**. Le annotazioni sul singolo metodo indicano al sistema quale codice eseguire a fronte di una *Request* su un determinato indirizzo. L'URI template rappresenta il formato dei *path* che saranno gestiti dal metodo annotato. Il template può contenere dei placeholder che, all'occorrenza, possono essere associati ai parametri dell'handler method, proprio attraverso l'annotazione `@PathVariable`.

Entrando nello specifico, l'annotazione `@PostMapping` indica un mapping ad una richiesta con metodo POST. In questo caso, essendo interessati a recuperare dal sistema i parametri della request, è necessario utilizzare l'annotation `@RequestParam`: questa permette di sfruttare il *data binding* di Spring per associare un parametro della richiesta HTTP ad un corrispondente parametro dell'handler method. Nel metodo sopracitato, si è associato il parametro `file` del metodo al parametro "file" inviato nel *payload* della POST. Questa associazione fa sì che la variabile `file` riceva come argomento il valore del corrispondente parametro della richiesta HTTP.

Il metodo riportato precedentemente, annotato con `@GetMapping`, è adibito alla gestione delle richieste di tipo GET dirette al path con formato `file/{hash}`. Il **placeholder** `{hash}`, consente di recuperare il valore presente nell'URL e di assegnarlo al parametro `hash`. Per generare una response, il controller restituisce un apposito oggetto `ResponseEntity`: questo rappresenta l'intera risposta HTTP (codice di stato, intestazioni e corpo) e, di conseguenza, può essere usato per effettuarne una configurazione completa. È opportuno notare come anche all'interno di questa classe sia possibile uti-

lizzare l'annotazione `@Autowired` in modo tale da permettere all'istanza di `IPFSService` di essere utilizzata per soddisfare ulteriori dipendenze.

```
@Autowired
private IPFSService ipfsService;
```

### 5.3 Postman

Realizzate tutte le componenti che compongono il servizio web, non rimane altro da fare se non testarne l'usabilità. A tal proposito, si ricorre al software *Postman*. **Postman** è una diffusa piattaforma di collaborazione per lo sviluppo di API, utilizzata da oltre 7 milioni di sviluppatori e più di 300.000 aziende in tutto il mondo. *Postman* consente agli utenti di progettare, simulare, eseguire il debug, testare, documentare, monitorare e pubblicare API, tutto da un'unica posizione. Il client invia una chiamata ad un URL specificando un metodo (**GET**, **POST**, **PUT**, **DELETE**) e passando, se necessario, dei dati in JSON nel corpo della richiesta. Il server esegue la funzionalità collegata con l'URL e risponde inoltrando un codice di stato e, se previsto, un oggetto JSON nel corpo della richiesta. Per quanto riguarda la tipologia di risposta, è possibile utilizzare JSON, ma allo stesso modo è ragionevole selezionare altre tipologie di formati di risposta come ad esempio XML, HTML e molto altro ancora, a seconda delle proprie necessità [33].

Attraverso Postman è possibile testare qualsiasi tipo di servizio in maniera semplice e veloce grazie anche alla sua interfaccia abbastanza intuitiva. Tra i vari vantaggi che offre Postman è possibile trovare la cronologia con tutte le chiamate effettuate e con tutte le autenticazioni utilizzate.

Postman può essere scaricato gratuitamente al seguente link per i sistemi operativi Windows, macOS e Linux. Completata la procedura di installazione è possibile eseguire il programma. Questo consentirà di testare un servizio **POST** e un servizio **GET** sull'applicazione realizzata in precedenza.

In primis, è fondamentale verificare che IPFS sia in esecuzione sul proprio nodo locale. Inoltre, è opportuno accertarsi che la porta 8080 non sia già occupata da un altro servizio. Diversamente, il web service realizzato non potrà essere eseguito correttamente: a tal proposito, è essenziale recarsi nella scheda delle **Impostazioni** di IPFS Desktop e individuare la sezione relativa alle **Configurazioni**, in modo da modificare la porta in cui il Gateway locale è in ascolto (nel caso sotto esame, la porta assegnatagli è stata la 8081). Allo stesso modo, è necessario seguire la medesima procedura per IPFS Companion: recarsi quindi nella sezione relativa alle estensioni del proprio Browser, individuare l'icona di IPFS e selezionare al suo interno l'immagine a forma di ingranaggio; questo permetterà di aprire la scheda relativa alle **Impostazioni**, individuare la voce **Gateway locale** e modificarne la porta assegnatagli. Per semplicità, è anche possibile disattivare momentaneamente IPFS Companion tramite l'apposito pulsante all'interno dell'estensione.

### 5.3.1 I servizi POST e GET

Affrontate questa premessa, è ora possibile procedere con il testing dell'applicazione configurando i parametri all'interno di Postman per poter iniziare a testare i servizi creati. Nella *homepage* di Postman è possibile definire due tra i valori essenziali di una qualsiasi chiamata a servizi REST, ovvero l'URL da testare e la tipologia di chiamata che si vuole realizzare. Come si può notare dall'immagine seguente, si vuole testare il servizio POST sull'URL `http://localhost:8080/upload/`: siccome si vuole memorizzare un file all'interno della rete IPFS, è opportuno specificare nel *body* della richiesta la risorsa che deve essere presa in carico dal metodo `saveFile` del `RestController`.

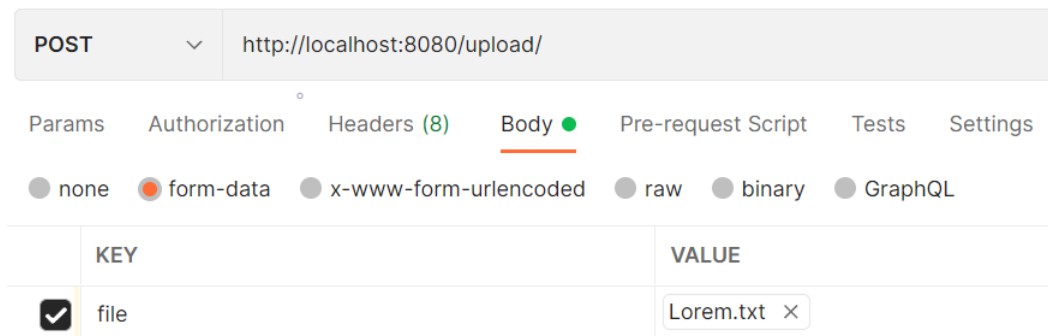


Figure 32: Servizio POST

Nel caso in cui l'operazione abbia esito positivo, verrà visualizzato a schermo **l'hash** che permette di identificare univocamente il file importato sulla rete.

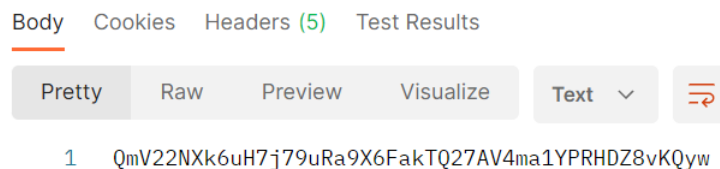


Figure 33: Identificatore del file caricato sulla rete IPFS

Completata con successo la procedura d'importazione di un file sulla rete IPFS, è possibile testare il servizio GET sull'URL `http://localhost:8080/file/{hash}`, in modo da recuperare la risorsa appena caricata. A tal proposito, è necessario sostituire il placeholder `{hash}` con l'identificatore del file che si vuole recuperare dal network, proseguendo poi con l'inoltro della richiesta.

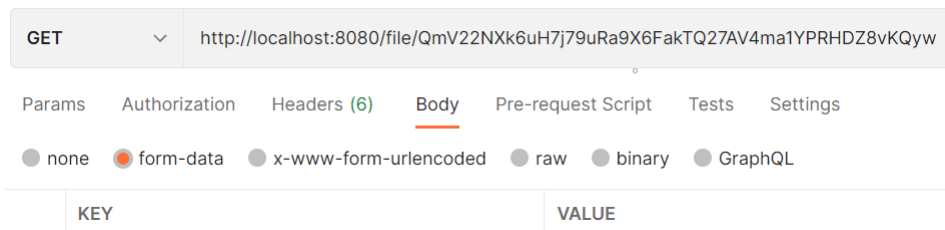


Figure 34: Servizio GET

Nel caso in cui la richiesta vada a buon fine, il contenuto del documento che si vuole recuperare verrà visualizzato sullo schermo.

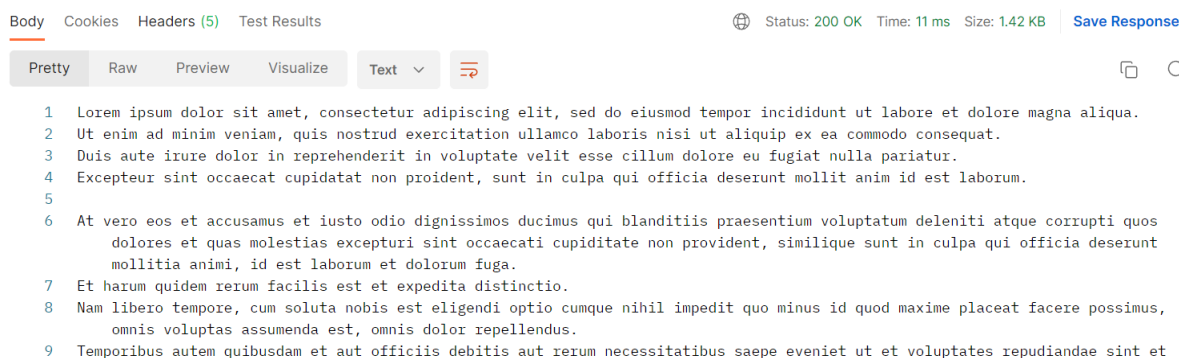


Figure 35: Contenuto del file recuperato dalla rete

Per completezza, si descrive la possibilità di visualizzare e salvare sulla memoria locale del proprio dispositivo un file che è stato importato sulla rete IPFS, recandosi all'indirizzo `https://ipfs.io/ipfs/{hash}` e specificando l'hash che fa riferimento a quella determinata risorsa.

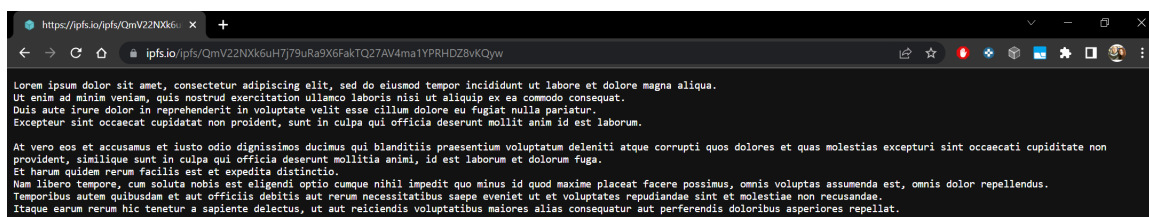


Figure 36: Visualizzazione/Salvataggio del file recuperato dalla rete IPFS

## 6 Testing

Per quanto riguarda i test sulla tecnologia, si è deciso di approfondire alcuni degli studi condotti da **ProbeLab** che hanno portato alla realizzazione di quello che è l'**IPFS Network Observatory**, una raccolta di misurazioni e analisi comparative della rete per identificare colli di bottiglia, possibili miglioramenti e per progettare ottimizzazioni del protocollo.

*ProbeLab* nasce dall'esigenza di sviluppare una comprensione più profonda di come le reti decentralizzate senza permessi possano essere rese più performanti e comparabili alle loro controparti centralizzate. Esso, infatti, rappresenta uno sforzo per applicare solide metodologie di misurazione scientifica per confrontare e ottimizzare i protocolli di rete che operano in ambienti P2P decentralizzati [34].

Le misurazioni di rete condotte da ProbeLab mostrano che la pubblicazione di contenuti sul network di IPFS rappresenta un *collo di bottiglia*. Questo è dovuto al fatto che, maggiore è la quantità di file archiviata all'interno di un peer, maggiore è il tempo necessario per comunicare al routing system quali risorse, memorizzate in quel nodo, sono disponibili per la condivisione. La comunità di IPFS ha identificato il *routing dei contenuti mantenuto dalla DHT* il principale problema da risolvere nel prossimo futuro. I ricercatori del ProbeLab hanno contrassegnato questa problematica come una delle principali priorità da affrontare nelle ricerche a venire. Per poter esplorare questo aspetto era però di preliminare importanza comprendere quale fosse lo *stato di salute della tabella di routing DHT*.

Per tale ragione, è stato realizzato lo studio che si propone di esaminare l'integrità della **tabella di routing DHT Kademlia** nella rete IPFS, consultabile in dettaglio al seguente link. Questo si concentra su quattro aspetti principali:

- La percentuale di entry *obsolete* presenti all'interno della DHT;
- La distribuzione dei peer all'interno dei bucket Kademlia (introdotti nel paragrafo 2.4.1);
- Il numero di peer mancanti nei vari bucket;
- Lo stato delle tabelle di routing dei vari nodi IPFS (se questi mantengono memorizzati al loro interno i 20 peer più vicini ad essi).

Grazie a queste metriche è stato possibile valutare lo stato effettivo della tabella di routing DHT.

### 6.1 Metodologia di misurazione

Per ottenere il contenuto della tabella di routing dei peer della rete IPFS è stato utilizzato il **Crawler Nebula**. Il *Crawler Nebula* avvia un nodo IPFS libp2p richiedendo ad esso un CID casuale in modo da scoprire gradualmente tutti i peer in linea sul network. Il Crawler Nebula fornisce, a ogni scansione, gli ID dei peer nella rete IPFS, così come le tabelle di routing di questi ultimi. Una volta che il crawler ha restituito tutte le tabelle

di routing, si è in grado di ricostruire i k-bucket per tutti i nodi grazie ai peer ID che sono stati recuperati. Individuati quelli che sono tutti i peer attivi sulla rete al momento della scansione, è possibile ricavare un'istantanea globale del network e lo stato della tabella di routing per ogni nodo. Con l'elenco dei nodi ottenuto, è possibile effettuare un ordinamento di tutti i peer della rete in base alla distanza che c'è tra ognuno di essi, producendo  $n$  liste ordinate di  $n$  peer, supponendo che ci siano  $n$  nodi nella rete. Avendo queste informazioni, è possibile costruire i k-bucket previsti da questi elenchi ordinati. Il confronto dei k-bucket previsti con quelli effettivi consente di osservare qualsiasi peer mancante dalla tabella di routing effettiva, fornendo informazioni sullo stato di salute della tabella di routing.

## 6.2 Peer irraggiungibili

Per le misurazioni sono stati utilizzati i dati di 28 crawl del Crawler Nebula, ottenuti quattro volte al giorno (03:00, 09:00, 15:00, 21:00 UTC) per 7 giorni consecutivi (dal 19/04/2022 al 26/04/2022). In media, sui 28 crawl, sono stati scoperti 20.811 peer, di cui 15.371 raggiungibili. I peer irraggiungibili vengono rilevati siccome sono presenti nella tabella di routing dei peer raggiungibili.

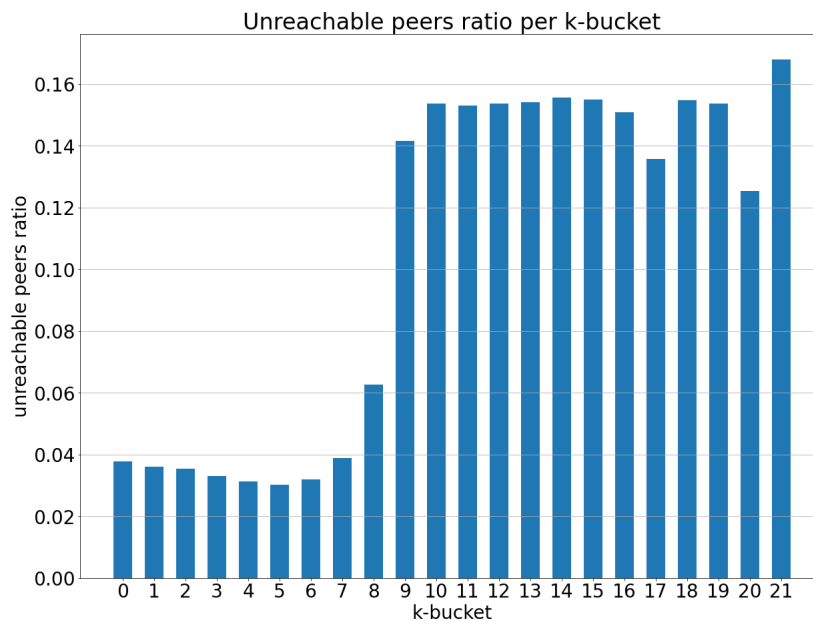


Figure 37: Peer irraggiungibili all'interno dei bucket

Questo grafico mostra il rapporto medio di peer irraggiungibili per i k-bucket. I bucket da 0 a 8 contengono tipicamente 20 peer, ovvero il massimo. Il bucket 0, con questi dati, ha in media il 3,78% di peer irraggiungibili, che corrisponde a una media

di 0,75 peer irraggiungibili su 20. Il rapporto tra peer irraggiungibili sale a circa il 15% per i bucket non pieni, ovvero quelli che vanno dal 9 al 21. Tuttavia, questa percentuale è piuttosto bassa il che significa che i nodi che lasciano la rete dopo un breve periodo di tempo **non influiscono pesantemente sulle tabelle di routing**. I peer instabili potrebbero non rimanere online abbastanza a lungo da essere inseriti nelle tabelle di routing. Pertanto, anche i bucket con un ID elevato non vengono riempiti principalmente da peer irraggiungibili, il che è positivo.

La visibile discrepanza tra il bucket 8 e 9 è, con alta probabilità, dovuta alla politica di sostituzione del k-bucket. La tabella di routing viene aggiornata completamente ogni 10 minuti, ogni peer viene interrogato e i peer che non rispondono tempestivamente vengono eliminati da essa. Inoltre, anche i nodi che non rispondono alle richieste di ricerca vengono immediatamente eliminati dalla tabella. Il numero di entry obsolete nei bucket 9-21 è elevato siccome è improbabile che i nodi richiedano qualcosa ai peer in questi bucket entro l'intervallo di aggiornamento, poiché rappresentano una piccola parte dello spazio delle chiavi. D'altra parte, interagiranno con i nodi nei bucket 0-8 e sostituiranno immediatamente le entry obsolete, riducendo il rapporto dei peer che non rispondono in questi bucket.

### 6.3 K-bucket incompleti

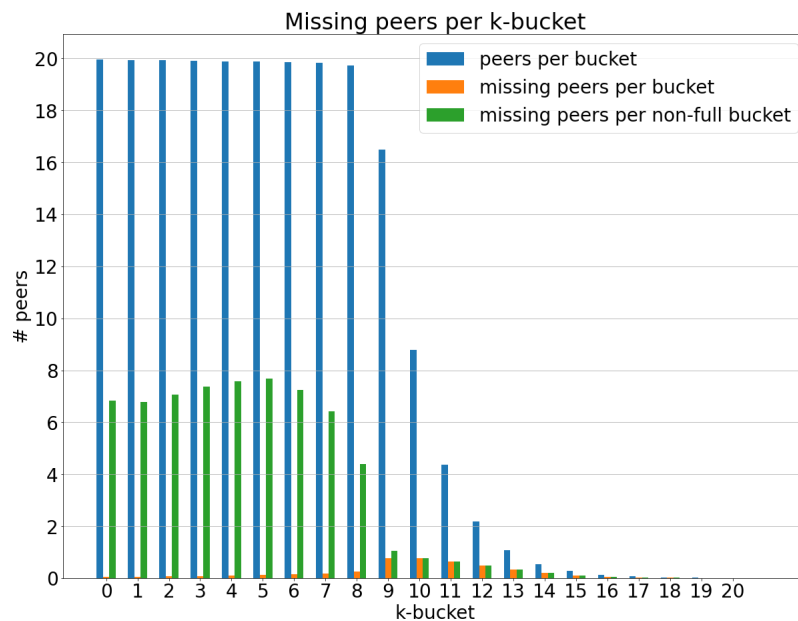


Figure 38: Peer mancanti per bucket



Un peer *mancante* è definito come un peer il cui ID si adatterebbe a un k-bucket incompleto, ma dove questo non è incluso al suo interno. Il numero massimo di peer mancanti per k-bucket può essere descritto come  $m=20-p$ , dove  $p$  è il numero di peer nel bucket. Il grafico soprastante mostra il numero medio totale di peer per bucket rispetto al numero medio di peer mancanti per bucket, e rispetto al numero medio di peer mancanti per k-bucket incompleti.

I bucket dallo 0 all'8 sono quasi sempre pieni (19,88 peer in media), mostrando un numero molto basso di peer mancanti, ovvero lo 0,12. Tra questi bucket (0-8), è possibile individuarne due categorie principali:

- Il 98,06% dei bucket è pieno e non mancano peer;
- L'1,94% dei bucket non è pieno e mancano in media 6,82 peer.

Una possibile spiegazione è che i peer si sono recentemente uniti alla rete e la loro tabella di routing non è ancora completamente popolata.

Per quanto riguarda i bucket con ID 9 o superiore, il numero di peer mancanti per bucket corrisponde quasi sempre al numero di peer mancanti per bucket non pieno, poiché questi bucket raggiungono raramente la loro capienza massima. In media, nei bucket 9-14 mancano 0,53 peer per ogni bucket, ovvero il 19,76%.

## 6.4 I peer più vicini

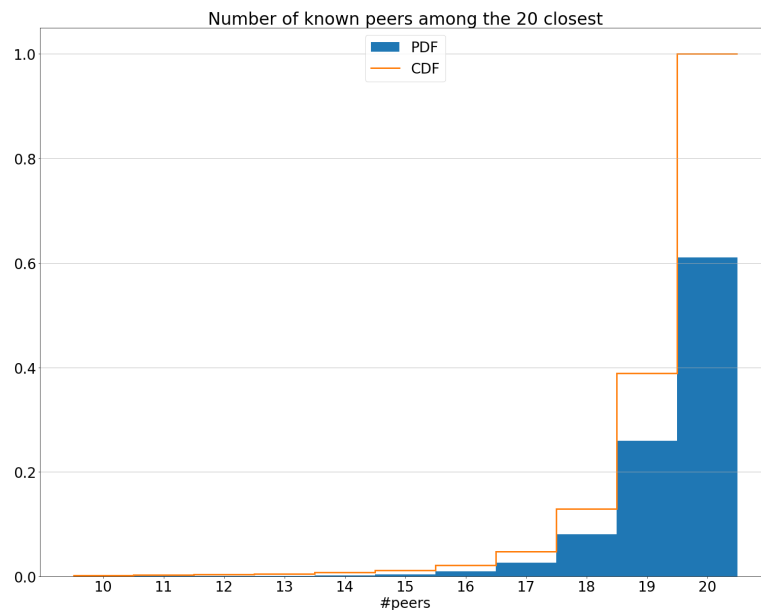


Figure 39: Numero di peer conosciuti tra i 20 più vicini

Il grafico mostra la Probability Density Function (PDF) e la Cumulative Distribution Function (CDF) del numero di peer, tra i 20 più vicini a un nodo N, nella tabella di routing di N, per tutti i nodi N. In esso viene mostrato che il 61,09% dei peer possiede tutti i 20 peer più vicini ad essi all'interno della propria tabella di routing. Solo il 4,79% dei nodi possiede meno di 18 dei 20 peer ad loro più vicini, il che rappresenta un dato sorprendentemente buono dato l'alto tasso di abbandono dei peer dalla rete approfondito in un altro studio (RFM2). Infine, lo 0,102% dei nodi conosce meno di 5 dei 20 peer più vicini.

## 6.5 Considerazioni sui dati

Questi risultati mostrano quanto sia resiliente la tabella di routing Kademlia, in quanto dimostra che i nodi saranno sempre raggiungibili dalla rete. Nonostante il Crawler Nebula non sia in grado di rilevare nodi irraggiungibili, è molto improbabile che ve ne siano molti data la distribuzione di cui sopra. Le misurazioni mostrano inoltre che la tabella di routing Kademlia DHT sembra essere integra rispetto a tutti gli aspetti misurati nella rete IPFS. In questo studio è stato mostrato che il 95,21% dei peer nella rete IPFS conosce almeno 18 dei 20 peer più vicini ad essi, il che è un dato eccellente visto l'alto tasso di abbandono osservato nella rete. Inoltre, si è notato che il numero di peer nei k-bucket segue una crescita esponenziale dai bucket 14-9, dovendo essere limitato a 20 peer per i bucket 8 e inferiori, come previsto. Infine si è riscontrato una mancanza media di 0,12 peer per k-bucket completi e il 19,76% per k-bucket non completi, il che indica grandi prestazioni in termini di aggiornamento delle tabelle di instradamento. Per maggiori dettagli riguardo allo studio effettuato da ProbeLab e ai progetti correlati, si consiglia di consultare la pagina Github ad essi dedicata.

## 7 Progetti correlati

IPFS è un progetto altamente modulare che si compone a sua volta di molti protocolli e strumenti differenti. Alcuni di essi esistono come progetti indipendenti sotto il supporto generale di **Protocol Labs**, un laboratorio di ricerca e sviluppo open-source il cui obiettivo è quello di costruire protocolli, strumenti e servizi per migliorare l'Internet odierno. Di seguito, se ne descrivono brevemente i principali.

### 7.1 IPLD

**IPLD** è il modello di dati per il web *indirizzabile al contenuto*. Esso consente di trattare tutte le strutture dati collegate ad hash come sottoinsiemi di uno spazio informativo unificato, raccogliendo tutti i modelli di dati che collegano le informazioni con gli hash come istanze di IPLD. L'indirizzamento dei contenuti tramite hash è diventato un mezzo ampiamente utilizzato per connettere i dati nei sistemi distribuiti, dalle *blockchain* che gestiscono le criptovalute, ai *commit* che supportano il codice, al contenuto del web in generale. Tuttavia, sebbene tutti questi strumenti si basino su alcune primitive comuni, le loro specifiche strutture di dati sottostanti non sono interoperabili. IPLD è un unico *namespace* per tutti i protocolli basati su hash. Attraverso IPLD, i collegamenti possono essere attraversati tramite protocolli, consentendo di esplorare i dati indipendentemente dal protocollo sottostante. Per maggiori informazioni è possibile consultare il seguente link.

### 7.2 Libp2p

**Libp2p** è un framework di rete modulare per reti peer-to-peer. Consiste in un catalogo di moduli da cui gli sviluppatori di reti *p2p* possono selezionare e riutilizzare solo i protocolli di cui hanno bisogno, semplificando al contempo l'aggiornamento e l'interoperabilità tra le applicazioni. Questo include diversi protocolli e algoritmi per consentire una comunicazione peer-to-peer efficiente, come il **peer discovery**, il **peer routing** e il **NAT Traversal**. Sebbene libp2p sia utilizzato da IPFS, è uno stack autonomo che può essere utilizzato anche indipendentemente da questo sistema. Per maggiori dettagli, rivedere il paragrafo 2.5.1 o consultare la documentazione di libp2p al seguente link.

### 7.3 IPFS Cluster

**IPFS Cluster** è un'applicazione distribuita che funziona come *sidecar* per i peer IPFS, mantenendo un pinset del cluster globale e allocando in modo intelligente i suoi elementi ai nodi della rete. Più in dettaglio, IPFS Cluster fornisce l'orchestrazione di dati attraverso uno **sciame di daemon IPFS** (*swarm*) allocando, replicando e tracciando un pinset globale distribuito tra diversi peer. Se si vuole approfondire questo argomento è possibile visitare la pagina web ad esso dedicata.

## 7.4 DNSLink

**DNSLink** rappresenta un protocollo piuttosto semplice che viene utilizzato per collegare contenuti e servizi direttamente dal DNS, sfruttando la potente architettura distribuita del DNS per una varietà di sistemi che richiedono nomi o puntatori mutabili su scala globale. Grazie a DNSLink è possibile memorizzare un collegamento generico utilizzando qualsiasi nome di dominio DNS tramite i record **TXT**: DNSLink utilizza i record *DNS TXT* per mappare un nome DNS, ad esempio **ipfs.io**, a un indirizzo IPFS. Poiché è possibile modificare i record DNS (a cui è possibile assegnare nomi, percorsi e sottodomini facili da digitare, leggere e ricordare), questi possono essere utilizzati per fare riferimento sempre all'ultima versione di un determinato oggetto in IPFS. Per approfondire questo protocollo è possibile visitare la relativa documentazione al seguente link.

## 7.5 Multiformats

Il progetto **Multiformats** è una raccolta di protocolli che mirano a realizzare sistemi *a prova di futuro* attraverso valori di formato autodescrittivi, che consentono l'interoperabilità e l'agilità del protocollo. Gli aspetti autodescrittivi dei protocolli hanno alcune clausole da rispettare:

- Devono essere in-band (con il valore);
- Devono evitare il lock-in e promuovere l'estensibilità;
- Devono essere compatti e avere una rappresentazione in formato binario;
- Devono avere una rappresentazione facilmente comprensibile dagli umani.

Multiformats è il nome dell'organizzazione, ma può anche essere usato per fare riferimento ai protocolli che ne fanno parte. Maggiori informazioni a riguardo possono essere trovate al seguente link.

## 7.6 ProtoSchool

**ProtoSchool** mette a disposizione tutorial interattivi autoguidati che permettono di affacciarsi a concetti, protocolli e strumenti del web decentralizzato. Selezionando l'argomento d'interesse è possibile intraprendere un percorso di apprendimento che comprende una serie di domande atte a verificare la piena comprensione degli argomenti trattati, tenendo traccia dei progressi durante l'avanzamento. Per ulteriori informazioni, consultare la homepage di Protoshool.

## 8 Conclusioni

### 8.1 Punti di forza e di debolezza della tecnologia

Riassumendo sommariamente quanto visto finora a proposito di HTTP e IPFS, l'**HyperText Transfer Protocol**, o **HTTP**, è un protocollo di rete a livello applicativo creato da *Tim Berners Lee* al Cern nel 1989. Attualmente, è utilizzato per la maggior parte del trasferimento di dati sul web: contiene regole e istruzioni indispensabili affinché due macchine collegate in rete possano interagire tra di loro. Esso costituisce la base della comunicazione di dati tramite file ipertestuali sul World Wide Web. L'**InterPlanetary File System**, o **IPFS**, è invece un protocollo relativamente nuovo che si prepone di cambiare il modo in cui viene utilizzato Internet. È un progetto creato da *Juan Benet* nel 2015 con l'obiettivo di creare un web completamente *decentralizzato* attraverso un'architettura peer-to-peer distribuita su tutto il globo.

Il ragionamento che porta a identificare i punti di forza e di debolezza che, secondo il mio parere, caratterizzano IPFS è piuttosto intricato. Per quanto riguarda l'ambito relativo alla sicurezza, va notato come questo approccio presenti diversi vantaggi. In primis vi è un importante concetto di **non-modificabilità** dei contenuti. Questa caratteristica fa sì che IPFS implementi nativamente al suo interno una specie di *versioning* delle risorse, simile ai protocolli di **Git**. Inoltre, un filesystem distribuito peer-to-peer è intrinsecamente **più sicuro** di una sistema centralizzato. Nell'ipotesi in cui un server non sia più raggiungibile, non lo saranno nemmeno tutti i file contenuti al suo interno. Su IPFS, un file può essere replicato su diversi nodi, rendendo la risorsa **disponibile** praticamente in qualsiasi momento. Grazie a quest'ultima qualità, non è necessario connettersi ad uno specifico server remoto per recuperare un particolare documento, evitando quindi un potenziale **spreco di banda**. Per questa ragione, IPFS rappresenta un'interessante opzione per quei Paesi in cui le reti sono limitatamente sviluppate. La decentralizzazione dei contenuti è anche un **deterrente** verso la loro censura o la loro rimozione da Internet: se un documento non viene fisicamente eliminato da tutti i nodi che lo ospitano, sarà sempre raggiungibile. Non a caso, esiste una copia di *Wikipedia* su IPFS, raggiungibile anche dalle Nazioni che filtrano il traffico HTTP.

Nonostante ciò, non è tutto positivo ciò che all'apparenza sembra tale: come anticipato, IPFS è relativamente più recente rispetto ad HTTP e, per tale ragione, non ne possiede la stessa popolarità e diffusione. Per di più, per poter eseguire IPFS è necessario accedervi utilizzando il portale da HTTP a IPFS, o configurare manualmente un nodo IPFS sulla propria macchina. Inoltre, vi è un esiguo numero di nodi che compongono la rete decentralizzata di questo protocollo, dettata dalla sua bassa popolarità tra i fornitori di Internet.

Per concludere, aggiungerei che la vastità del progetto e l'assenza di un'implementazione canonica di riferimento per IPFS potrebbero rappresentare un punto di debolezza per coloro che vogliono approcciarsi ad esso. Per permettere che una tecnologia venga sfruttata a pieno, indipendentemente dal dispositivo da cui se ne vuole usufruire, è necessario che essa sia standardizzata, e quindi che abbia regole e protocolli ben precisi. Nonostante Kubo rappresenti la prima e più diffusa implementazione per il file system distribuito,

l'assenza di una linea univoca da seguire potrebbe rappresentare uno scoglio significativo per coloro che sono nuovi al protocollo: l'ampia varietà di implementazioni disponibili, ognuna delle quali specializzata nello sperimentare e ottimizzare uno specifico caso d'uso, potrebbe disorientare l'utente neofita fino a condurlo all'abbandono della tecnologia.

## 8.2 Parere personale sulla tecnologia

Oggigiorno esistono diversi progetti che puntano ad abbandonare il modello del web **centralizzato** il quale, facendo uso del protocollo HTTP, è protagonista di diverse problematiche lato sicurezza, dagli attacchi *man-in-the-middle* ai *DDoS*, i quali mettono fuori gioco il server fino a non renderlo più in grado di erogare il servizio ai client richiedenti, impedendo a chiunque di connettersi.

A fronte dello studio svolto per la realizzazione di questo elaborato, credo che **IPFS** rappresenti un valido sostituto ad **HTTP** e che possa effettivamente essere visto come una futura alternativa ad esso. *IPFS*, infatti, rispetta la visione originale del web, rendendolo libero e veramente paritario: la distribuzione delle risorse tra nodi supporta la resilienza di Internet, e quindi la persistenza dei contenuti nel tempo, oltre a rendere i file stessi incensurabili. Tuttavia ritengo che sia difficile per IPFS affermarsi sul mercato, nonostante i punti di forza che questo nuovo progetto implementa.

HTTP, infatti, rappresenta il protocollo standard a livello applicativo usato come sistema per la trasmissione di informazioni sul web, è incluso in praticamente tutte le apparecchiature informatiche, è sul mercato da molto più tempo rispetto alla sua controparte decentralizzata ed è semplice e veloce da utilizzare. Inoltre, è importante mettere alla luce una limitazione rilevante di IPFS: per poter accedere alla sua rete distribuita è necessario utilizzare il portale HTTP, o configurare manualmente un nodo sul dispositivo dal quale si intende utilizzare il protocollo. Questo, secondo il mio parere, porta alla luce un'ulteriore problematica: la maggior parte dei consumatori di Internet non rappresenta coloro che vengono definiti come "utenti consapevoli", indi per cui la semplicità di utilizzo, la riluttanza al cambiamento e l'enorme diffusione che oramai coinvolgono HTTP sovrastano i vantaggi implementati da IPFS.

Come si è potuto constatare dall'elaborato, vi è una grande differenza tra i due protocolli. Su IPFS si sta lavorando giornalmente e l'interesse è alto, ma è anche vero che sarebbe improprio affermare che vi sia un brillante futuro ad aspettarlo. Nonostante ciò, le prospettive di sviluppo sono ottime, così come lo sono per altre tecnologie legate alla decentralizzazione, in primis le **blockchain**.

## References

- [1] Wikipedia Contributors. *File system*. 2022. URL: [https://it.wikipedia.org/w/index.php?title=File\\_system&oldid=128432877](https://it.wikipedia.org/w/index.php?title=File_system&oldid=128432877).
- [2] *What is IPFS?* URL: <https://docs.ipfs.io/concepts/what-is-ipfs/#decentralization>.
- [3] *Immutability*. URL: <https://docs.ipfs.tech/concepts/immutability/>.
- [4] *Nodes*. URL: <https://docs.ipfs.tech/concepts/nodes>.
- [5] *How IPFS works*. URL: <https://docs.ipfs.io/concepts/how-ipfs-works>.
- [6] Wikipedia Contributors. *Crittoanalisi*. 2021. URL: <https://it.wikipedia.org/w/index.php?title=Crittoanalisi&oldid=119785398>.
- [7] Wikipedia Contributors. *Funzione crittografica di hash*. 2021. URL: [https://it.wikipedia.org/w/index.php?title=Funzione\\_crittografica\\_di\\_hash&oldid=123224492](https://it.wikipedia.org/w/index.php?title=Funzione_crittografica_di_hash&oldid=123224492).
- [8] *Multibase*. URL: <https://github.com/multiformats/multibase>.
- [9] Multiformats community. *Multihash*. URL: <https://multiformats.io/multihash>.
- [10] Wikipedia Contributors. *Topological sorting*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Topological\\_sorting&oldid=1092588874](https://en.wikipedia.org/w/index.php?title=Topological_sorting&oldid=1092588874).
- [11] Wikipedia Contributors. *Directed acyclic graph*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Directed\\_acyclic\\_graph&oldid=1099254255](https://en.wikipedia.org/w/index.php?title=Directed_acyclic_graph&oldid=1099254255).
- [12] Protocol Labs. *Merkle DAGs: Structuring Data for the Distributed Web*. URL: <https://proto.school/merkle-dags>.
- [13] Wikipedia Contributors. *Hash table*. 2021. URL: [https://it.wikipedia.org/w/index.php?title=Hash\\_table&oldid=122424294](https://it.wikipedia.org/w/index.php?title=Hash_table&oldid=122424294).
- [14] Wikipedia Contributors. *Distributed hash table*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Distributed\\_hash\\_table&oldid=1090190114](https://en.wikipedia.org/w/index.php?title=Distributed_hash_table&oldid=1090190114).
- [15] Wikipedia Contributors. *Kademlia*. 2022. URL: <https://it.wikipedia.org/w/index.php?title=Kademlia&oldid=127499660>.
- [16] *Kademlia*. URL: <https://docs.ipfs.tech/concepts/dht/#kademlia>.
- [17] *Libp2p*. URL: <https://docs.ipfs.io/concepts/libp2p/>.
- [18] Protocol Labs. *Glossary*. URL: <https://docs.libp2p.io/reference/glossary/>.
- [19] Protocol Labs. *What is Libp2p?* URL: <https://docs.libp2p.io/introduction/what-is-libp2p/>.
- [20] *Bitswap*. URL: <https://github.com/ipfs/specs/blob/main/BITSWAP.md>.
- [21] *Install IPFS*. URL: <https://docs.ipfs.tech/install/>.
- [22] OpenJS Foundation. *What is Electron?* URL: <https://www.electronjs.org/docs/latest>.

- [23] *IPFS Desktop*. URL: <https://docs.ipfs.tech/install/ipfs-desktop/>.
- [24] *Command-line*. URL: <https://docs.ipfs.tech/install/command-line/>.
- [25] *IPFS Companion*. URL: <https://docs.ipfs.tech/install/ipfs-companion/>.
- [26] *IPFS Gateway*. URL: <https://docs.ipfs.tech/concepts/ipfs-gateway/>.
- [27] *Garbage collection*. URL: <https://docs.ipfs.tech/concepts/persistence/#garbage-collection>.
- [28] *Host a single-page website on IPFS*. URL: <https://docs.ipfs.tech/how-to/websites-on-ipfs/single-page-website/>.
- [29] *ipfs add*. URL: <https://docs.ipfs.tech/reference/kubo/cli/#ipfs-add>.
- [30] Wikipedia Contributors. *Spring Framework*. 2021. URL: [https://it.wikipedia.org/w/index.php?title=Spring\\_Framework&oldid=120602057](https://it.wikipedia.org/w/index.php?title=Spring_Framework&oldid=120602057).
- [31] David Castelletti. *Cos'è Spring framework e perché conviene conoscerlo*. 2017. URL: <https://davioooh.com/blog/2017/07/22/spring-intro>.
- [32] David Castelletti. *Implementare REST API con Spring Boot*. 2018. URL: <https://davioooh.com/blog/2018/03/23/rest-api-spring-boot>.
- [33] LentuxInformatica. *Postman: il tool per testare le API REST*. URL: <https://lentux-informatica.com/testare-le-api-restful-ce-postman/>.
- [34] Yiannis Psaras. *Introducing ProbeLab*. 2022. URL: <https://blog.ipfs.tech/2022-06-15-probelab/>.