# IPFS-FAN: A Function-Addressable Computation Network

Alfonso de la Rocha
*Protocol Labs*
alfonso@protocol.ai

Yiannis Psaras
*Protocol Labs*
yiannis@protocol.ai

David Dias
*Protocol Labs*
david@protocol.ai

***Abstract*—Permissionless computation is one of the missing pieces in the *web3 stack* in order to have all the tools needed to *"decentralise Internet services"*. There are already proposals to embed computation in decentralised networks like smart contracts, or blockchain networks for computational offloading. Although technically sound, their computational model is too restrictive to be used for general purpose computation. In this paper, we propose a general architecture of a decentralised network for general-purpose and permissionless computation based on content-addressing. We present a proof-of-concept prototype and describe in detail its building blocks.**

***Index Terms*—P2P, Permissionless, merkle-link, IPFS, Web Assembly, distributed computing**

## I. Introduction

Blockchain technology set the beginning of a new golden age for P2P technologies with the appearance of Bitcoin. Since the Bitcoin blockchain was introduced more than 10 years ago, lots of new blockchain-based platforms have been developed, launched, and are being in active use. Initially the vast majority of blockchain platforms targeted the *financial system*, but these platforms gradually evolved to accommodate increasingly ambitious use cases such as *Ethereum*'s decentralised applications [1], or a variety of decentralised storage platforms, such as IPFS [2] This new generation of P2P systems does not limit itself to *the financial system*, but is building the substrate to *fix the Internet* from its most commonly spread and well-known flaws, mainly centralization, big tech dominance, vendor lock-in, and privacy. The overarching idea is to deal with those issues through a decentralised architecture built upon P2P protocols. This substrate of P2P protocols aiming to overcome the current limitations and unbalances of the Internet are commonly referred to as the *Web3 stack*.

From decentralised storage, to transport abstractions, the *Web3 stack* has already consolidated projects to supersede many of the components of the current centralised Internet. However, a general proposal to embed computation in decentralised networks that is able to scale and compete with its centralised counterparts is still missing. There are existing proposals for decentralised computation: from Ethereum smart contracts [1]; to global collaborative computational infrastructures like iExec [3] and Golem [4]; and decentralised general-purpose computation networks such as the Fluence Network [5] and Dfinity's Internet Computer [6]. These platforms use varying approaches to embed computational capabilities in a global decentralised network, but lack in some way the ability to become the "de-facto" architecture for general-purpose computation in P2P networks.

Along with P2P technologies, content-addressable networks have also received significant attention during the last decade, due to the promising features that they offer. Location-independent content retrieval and arbitrary in-network caching can increase delivery performance significantly and reduce network resource requirements. Content-based object addressing offers a way to uniquely identify every resource that exists in the network - not only content. All of these features are extremely useful for the construction of a network hosting heterogeneous resources.

The recent advances in these two fields already offer all the fundamental components required to build a consistent proposal of a decentralised network for general-purpose computation. In this paper we present a general model to build a decentralised network for general-purpose and permissionless computation based on content- and *function-addressing*. In our system, both, content and functions are uniquely identifiable and globally addressed. Functions run code and perform computations over content in the network. The input of functions are content-addressed, i.e. uniquely identified, objects stored in the network. The output is the identifier of a newly generated content-addressable object, together with the result of the computation. Data and code are available by all peers in the network, enabling the composability of functions and the deployment of complex use cases. These unique features, primarily based on the content-addressing principle, eases the decentralization of the system and improves its scalability compared to existing projects. The modular architecture of our proposal builds a core that can be leveraged by other projects (including the aforementioned ones) to flexibly embed computation in decentralised networks. The reference implementation of our system is built on top of the InterPlanetary File System architecture [7].

The contributions of our work are: (i) the identification of the general building blocks required to build a decentralised network for general-purpose and permissionless computation based on content-addressing; and (ii) a functional proof-of-concept prototype of the *"IPFS Function Addressing Network"*, IPFS-FAN [8], leveraging existing modules from the Web3 stack and without the need of ad-hoc implementations.

ISBN 978-3-903176-39-3© 2021 IFIP

## II. Related Work

Current proposals for permissionless computation in decentralised networks can be classified in three different groups:

(i) *Smart Contracts* offer a constraint environment to perform computations in blockchain networks. They are generally identified by a network-wide unique id that can be used by any peer in the network to call functions in it. Calling a function of a smart contract triggers the execution of its code in all the peers of the network (or at least in the ones responsible for the consensus in the network), as any update in the state of data triggered by a smart contract needs to be validated and consensuated by all peers in the network. In order for this to be possible, executions in smart contract must be deterministic. They are extremely useful for the use cases accommodated by blockchain networks, but are not suitable to deploy use cases that require general-purpose computation, or are CPU-intensive. Smart contracts are widespread in a great gamut of projects in the blockchain space [1], [9].

(ii) *Decentralised Computation Marketplaces* [3], [4] appeared as an alternative way of overcoming the computational limitations of smart contracts. Platforms like Golem [4] and iExec [3] offer a way of executing computationally intensive and general-purpose programs in a decentralised manner. Decentralised Computation Marketplaces aggregate computational resources from providers in a decentralised network. Developers can rent resources from the available pool of resources to run their jobs. Unlike smart contracts, in computation marketplaces the data and the code being executed is not public, and developers request their execution "on-demand", sending the code and the data to the network. The output of the computation is reported to the user, an application or a blockchain network (or a multitude of them), while a cryptocurrency over a blockchain network is used to pay for the resource rental. Decentralised Computation Marketplaces, however, are not suitable to host decentralised applications running general-purpose computations.

(iii) Finally, *Decentralised General-Purpose Networks* are general-purpose, permissionless computation platforms that can host any kind of decentralised application. The main exponents of this type of networks, *Fluence Network* [5], *Dfinity's Internet Computer* [6], share many of the core components proposed by our model, such as: the use of a universal runtime based on WebAssembly embedded in every peer in the network; the concept of functions as the minimal unit of code executable in the network; the use of a declarative or programming language to orchestrate the composability of functions deployed in the network; and the use of an entry point to describe the code or application deployed.

Both proposals lack components required for the implementation of a complete general-purpose permissionless network: none of them use self-describing strategies to identify code so it can be uniquely and globally addressed in the network - Fluence functions have a unique id per deploying peer, while the Internet Computer uses a unique id per subnetwork; they do not use globally accessible decentralised data structures to represent application's state (data is locally accessible and location-dependent); they are not completely permissionless and limit in some way the peers that can participate from the protocol. Our proposed architecture for a general-purpose permissionless network addresses all of these limitations. The generality of our model offers the core over which projects such as *Fluence* and *Dfinity's Internet Computer* would be able to deploy their proposals and leverage many of their already implemented assets.

Finally, *function-addressing*, i.e. the identification of code deployed in the network through a unique identifier, is not a new concept, and it has been proposed in different ways in the academic literature in the field of Information-Centric Networks (ICN). Named Function Networking (NFN) [10] presents a scheme were names are computation expressions which include data and function names. Additionally, the authors in [11] propose NFaaS, a framework that extends the Named Data Networking architecture to support in-network function execution. Functions can be downloaded and executed in any node of the network, extending computation to the edge of the network.

## III. Use cases and impact

A distributed network equipped with a content-addressed computation model as the one described in this paper opens the door to a great a gamut of applications and use cases "out of the box" such as:

*Global serverless infrastructure:* The ability to deploy code that can be called and run from anywhere through a unique identifier offers application developers a platform to deploy their back-end functions without the need for additional infrastructure. End-users' front-ends can directly interact with content and trigger the execution of code in the network, unleashing the promise of real decentralised applications, improving applications maintainability and scalability, and developers' User Experience [12].

*Load-balancing by design:* The properties of content-addressing ensures that code and content are location-independent. This means that the execution of functions does not need to necessarily occur where the content or the code is hosted. The higher the demand for a specific function in the network, the more nodes get to store the code, and are capable of running the function. Scalability becomes a much easier challenge to deal with, as the closest peer with available resources could run the function without having to scale the infrastructure.

*Increased Availability Backends:* By not relying on a central infrastructure and hosting an application in a decentralised network, applications relying on the system ensures close to 100% availability as long as the required code and data are replicated in enough providers in the network.

*Collaborative computation and computation offloading:* Computationally expensive tasks can benefit from a network of peers contributing their computational resources to perform certain tasks. P2P collaborative computation networks have traditionally been used for research purposes (such as to perform complex simulations [13]). More recently, these collaborative computation networks have been used to offload computations to more powerful devices.

*Computation near the data:* When performing computations over large datasets, it normally becomes more expensive to transfer the data to the computation, rather than the opposite. With IPFS-FAN, users can easily get the code near the data for this type of use cases instead of the other way around.

## IV. General-Purpose Content-Addressing Computation Network

A permissionless computation network builds an infrastructure where peers share their computational resources to host data and run computations collaboratively. Its trustless nature ensures that any peer can join, leave, and perform operations in the network without requiring special permissions or supervision from a central authority. Our system is built upon content-addressing, offering in this way a scheme to uniquely identify every resource in the system and link them unambiguously.

In order to build a content-addressable computation network, the following modules are required:

*1) P2P Substrate:* Peer routing, content routing, p2p transport protocols, etc. to enable the communication and interaction between the different peers in the network.

*2) Decentralised storage:* Responsible for the collaborative storage of data and code in the network.

*3) Universal runtime and bytecode:* Execution environment with a portable binary instruction format capable of targeting any architecture, and that can be compiled from many high-level programming languages. Code in the network is always represented using this binary format so it can target the universal runtime embedded in every peer (interoperability).

*4) Universally addressable and linkable data structure:* To uniquely identify every resource in the network, and enable the representation of self-describing code and data. The identifier of the resource should include all the information required to identify its type and verify its integrity.

*5) Decentralised programming model:* Every piece of code in the network is identified with its unique ID and exposes an interface with its available functions and signatures. In our model, each piece of code in the network is an independent actor, that consumes data from the network and generates new data to it. Everything in the network is identified with a unique identifier (both code and data). When a function in an actor is triggered, we pass as arguments the ids/links for the data to be computed in the function's actor. The output of the function execution is new data added to the network and referenced by an id. Using a universally addressable and linkable data structure enables the de-duplication of data so that if an output for an actor is already stored in the network, it can be directly referenced by its id without requiring to explicitly add it (and conveniently store it) again. Every actor in the system needs to expose the signature of the functions it has exposed. This is done through an *ABI (Application Binary Interface)*. The ABI is the entry point manifest of every actor. An actor's ABI specifies the unique ID of the actor's bytecode (in order to be able to fetch it from the network), and the signature of all the functions exposed by the actor. The unique ID of an actor is inferred from its ABI, not from its bytecode (as the ABI already links to the bytecode of the actor). Over this programming model, a programming language can be designed to orchestrate the resources and interaction with actors in the network: from the deployment of new actors, to calling them, referencing data, and implementing complex programs composing the operation of several actors over data.

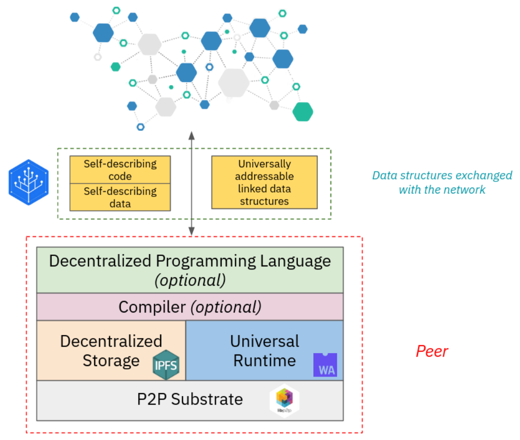
Self-describing code
Self-describing data
Universally addressable linked data structures
Data structures exchanged with the network
Decentralized Programming Language (optional)
Compiler (optional)
Decentralized Storage
IPFS
Universal Runtime
WA
P2P Substrate
Peer

Figure 1: IPFS-FAN Architecture

## V. Preliminaries

*1) IPFS and the Web3 Stack:* IPFS is a hypermedia protocol that builds on the principles of: i) peer-to-peer (P2P) networking and ii) content addressing. IPFS builds a distributed and decentralised, P2P storage and delivery network, which runs entirely on end-user devices and has no centrally controlled components. Under the hood, the IPFS Architecture is composed of a collection of subsystems each of which can be used independently of IPFS, but together form a robust foundation for a distributed, content-addressable, P2P storage and delivery network. All these modules are known as the Web3 stack.

- *libp2p* is a modular network-layer library for P2P networks. In libp2p, peers are identified by the hash of their public key. The library includes all the essential techniques to discover and connect peers, that is, from peer-discovery to peer routing, NAT-traversal and provision for several different transports that are easy to integrate.
- The *InterPlanetary Linked Data (IPLD)* layer is a naming and data management layer used by IPFS. IPLD builds on the concept of "Merkle-DAGs", that is, Merkle-Trees out of Directed Acyclic Graphs. Every file added into the IPFS system is converted into a Merkle-DAG, locally on the device that adds the file. Each node of the DAG has its own content address, which is the result of hashing of the content itself. Once converted, every node within the DAG is compatible with any data structure that wishes to make use of them, or include them as nodes in their own DAG.
- *Multiformats* is a set of formatting rules with specific structure that is used to declare the set of protocols used

in a P2P session. In other words, Multiformats is a data representation scheme that adds *self-describing* attributes to the data it formats.

*2) Content IDentifier (CID):* IPFS uniquely identifies content in the network through a CID. CIDs are structured by Multiformats inherently include the following properties:

- *CIDs are immutable and permanent:* given that the identifier of content is its own hash-digest means that any change to the content itself will result in a totally different hash and therefore, identifier. This makes every content added to the IPFS network *immutable* and therefore, permanent as an identifier of a version of an object that will never change. The notion of permanence here is not to be confused with the property of permanent availability in the system.
- *CIDs are self-certified and verifiable:* upon fetching a content chunk from the IPFS network, a user calculates the hash of the content he received and compares it against the hash digest which he requested from the network. If the two match, then the content is guaranteed to be authentic and received without errors.

*3) Adding and fetching content in IPFS:* When adding content to the IPFS network, the content is not replicated or uploaded to any external server. The content stays local on the user's device. Instead, it is the Content Identifier (CID) together with a pointer to the user's machine that is made known to the network and in particular to the Content Routing component of the system. This is so that others can point their requests to the right machine and retrieve the content through the content resolution process described below. The user adding content is called ”Content Provider or Publisher” and the record given to the content routing system is called ”Provider Record”.

To fetch content from the network, a peer can either use the DHT to find the provider records that point to the providers storing the content it is interested in, or it can run Bitswap (a gossip-based protocol to fetch content in P2P networks), to try to gather the content from its neighbors [14].

*4) Web Assembly:* WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine [15]. It is designed as a portable compilation target for programming language, and it already has support to be run in several environments such as client and server applications. The Wasm stack machine is designed to be encoded in a size- and load-time-efficient binary format. WebAssembly aims to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms. WebAssembly describes a memory-safe, sandboxed execution environment.

## VI. IPFS-FAN: Proof of concept

IPFS-FAN is the proof-of-concept prototype ( [8]) of a decentralised network for general-purpose, trustless computation, which is based on the model presented in Section IV. IPFS-FAN builds upon already existing modules from the Web3 stack, and does not require any extra implementation to make it work. IPFS-FAN uses *libp2p for the P2P substrate, IPFS for the decentralised storage, Wasm as the universal runtime, and IPLD for the universally addressable and linkable data structure*.

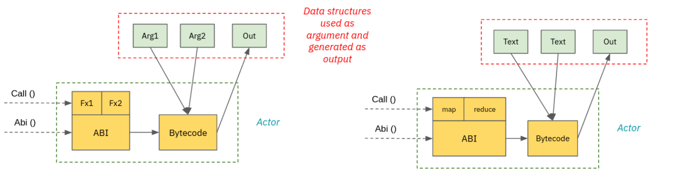

Figure 2: Actor model (left), and word-count example of PoC (right)

Specifically, IPFS-FAN is written in the Go programming language, and is a fork of an IPFS Lite peer slightly modified to embed in it a *Wasmtime* runtime. *Wasmtime* is a small and efficient runtime for WebAssembly and WASI [16]. IPFS-Lite [17] is an embeddable, lightweight IPFS peer which runs the minimal setup to provide all the basic operations to interact with the IPFS network: mainly, *add* and *get* IPLD nodes identified through a CID. For IPFS-FAN, three additional functions were added to the peer (as part of the programming model): (i) *deploy*, which deploys the ABI and bytecode of a decentralised actor, and makes it available through the CID of the ABI; (ii) *call*, which calls a function from an actor identified by a CID, using as arguments the data identified with a CID; and (iii) *abi* which outputs the ABI for an actor CID (if it exists) to understand the available functions exposed by an actor. In the proof-of-concept implementation of IPFS-FAN, the peer calling the *call* operation is the one responsible for running the code, but as described in section VII, in future versions, additional execution strategies could be devised. Listing 1 presents the signature of the new functions included to IPFS-lite, as well as the ABI format.

Listing 1: IPFS-FAN functions

```
// ABI
type ABI struct {
    Fxs []Functions
    Bytecode Cid
}
// Function signature
type Functions {
    Id string
    Args []interface{}
    Outputs []interface{}
}
// Returns CID of ABI
> deploy (bytecode: []byte, abi: ABI) cid
> abi (cid: Cid) // Returns Actor ABI
// Returns CID of output
> call (actor: Cid, fx: string, Args interface{}...)
```

With these actions, we have all the pieces in place to perform more complex computation over the model. We tested our model with the *hello world!* of distributed computation, a map-reduce word count program (see figure 2 and listing 2). We deployed a *wordcount*, $cid_{wc}$, actor with two functions: a *map* function that counts the words of the text received as an argument; and a *reduce* that receives as an argument the partial computations of different map operations and computes the final result. Let's consider $n$ different text fragments: $cid_{t1}, ..., cid_{tn}$ already in the network. The programming script to run a word count over these fragments leveraging the actor at $cid_{wc}$ is of the form shown in the following listing. In its current implementation, *IPFS-FAN* is not able to infer the ABI of the actor from its bytecode (this will be possible in the future, once compilers and tooling for this programming model are implemented).

Listing 2: Wordcount Wasm actor pseudocode

```
// Map function
MAP ([]cid_texts):
 output := map[string]int
 for all text in cid_texts do
    for all word in text do
        output[term]++
 return output
// Reduce function
REDUCE ([]cid_intermediate)
 output := map[string]int
 for all count in cid_intermediate do
    output[term] += count
 return output
```

Listing 3: Wordcount map-reduce

```
// Specify actor ABI
> abi = { Fxs:
    fx{Id: "map", Args: ...string,
        Output: interface{}}}
    fx{Id: "reduce", Args: ...interface{},
        Output: interface{}}}
}
// Deploy wordcount actor
> cid_wc = deploy(wordcount.wasm, abi)
// Count words for t1 to ti
> cid_res1 = call (cid_wc, "map",
    [cid_t1, ..., cid_ti])
// Count words for t1 to ti
> cid_res2 = call (cid_wc, "map",
    [cid_tj, ..., cid_tn])
// Aggregate partial results
> cid_out = call (cid_wc, "reduce",
    [cid_res1, cid_res2])
```

In depth, the script runs the following steps:

*1) Deploying the actor:* The actor is deployed specifying its ABI and bytecode. This adds the objects to the IPFS network, first the bytecode of the actor, and then the ABI which includes a link to the CID of the bytecode. The CID of the ABI, $cid_{wc}$, uniquely identifies the actor in the network. Code and data in IPFS-FAN are fetched and provided in the same way as it is currently done in IPFS (see section V-3).

*2) Calling actor functions:* From there on, a peer A in the network can call the actor through its CID, $cid_{wc}$. When a peer calls an actor, it first downloads the actor's ABI, inspects if the function called is available in the actor, *map*, and if the function's signature is correct. If this is the case, the bytecode for the actor is downloaded using the link in the actor's ABI, and the arguments specified. The bytecode for the actor is downloaded from the network following the link included in the ABI, and the arguments for the function are downloaded using their CIDs, $cid_{t1}, ..., cid_{tn}$. Using the downloaded bytecode and arguments, the peer then runs the function, and adds a new object to the network with the result, $cid_{res1}$. In parallel, any other peer B can do exactly the same computation using the same actor function with other arguments generating a different result, $cid_{res2}$.

*3) Using results from previous computations:* Other peers in the network are able to continue with the partial results from previous computations, $cid_{res1}$ and $cid_{res2}$, as these are globally accessible in the network. For example, peer C can call an actor function and *reduce* on the data generated by previous computations. This is done by calling the function's actor as it was done by peers A and B, $cid_{out} = call(cid_{wc}, "reduce", [cid_{res1}, cid_{res2}])$, benefiting from peer A's and B's previous results and not having to compute them again.

This simple proof-of-concept served to show practically all the features of our network model: (i) self-describing code and data, accessible by any peer in the network. (ii) A universal runtime that enables any peer to run the code hosted in the network. (iii) A distributed directive language to program complex operations composing functions from several actors leveraging the code deployed in the network.

## VII. Discussion and Future Work

This proof-of-concept presents the basic implementation of a general-purpose content-addressed computation network, and it gives a foundation to implement many of the use cases presented in section III. It also serves as the core layer over which to implement other computational networks. At the same time, there are several open research problems calling for further effort to reach the levels of efficiency of centralised cloud environments.

- *Configurable execution strategies*: In the baseline implementation of the network, the peer calling an actor's function is the one responsible for the execution of the code over the data in its own runtime, but the network allows other execution strategies too. Similarly to IPFS, in IPFS-FAN, provider records are stored in a redundant manner. Jobs could, therefore, be run in the 'k' closest nodes with enough computational resources available to run the load, balancing in this way load in the network; alternatively a strategy could be designed so that loads are forced to be run spatially proximal to the input data to avoid having to download it in the machine executing the code. This concept could be extended even further to the point of extending the *call* directive so peers can also choose and configure their desired execution strategy.
- *Full-fledged programming languages and tooling*: Our proof-of-concept uses a low-level directive language based on three operations to showcase the programming model of the network. To unleash its full potential, the network should include a full-fledged programming language that enables the seamless deployment of actors and references to data as if they were traditional stack calls and pointers to local data offering a seamless developer

experience. The programming language would compile the code to be run in the network (functions are actors, variables are links to data). This would require the design of compilers that automatically generate the code, interacts with the network deploying actors and inferring their ABI. The fact that our model has a general architecture and shares the common runtime with other projects in the space would allow the adaptation of existing tools that these projects use in our model,

- *Security and privacy*: The proof-of-concept does not consider the privacy and security of the system. Every peer in the network can see all the data and code, and run any load. In a real environment this shouldn't be the case, and additional security and privacy schemes need to be put in place to enable access policies over data and code, denial of service attack prevention, and privacy over the code executed and the data used. A lot of consistent proposals for this are already in place in other decentralised computation network proposals that can serve as good inspiration for our case.
- *Heterogeneous resources in the network*: In this proof-of-concept all peers are treated equally. In a real deployment of the network may have a pool of peers with different capabilities and resources ABIs could include information about the minimum amount of resources required for the execution of a function, so that the right peer is selected to optimise resource usage. This also enables the aggregation of a pool of resources for users to rent in order to run their computations.

### A. Long-term impact

A network with these characteristics has the potential to enable a variety of impactful use cases in the long-run such as: (i) Coexistence of different permissionless computation models: Content-addressed and function-addressed networks, and the use of a common runtime by all the peers in the network, enables the coexistence of smart contracts and other proposals for computational sandboxes from other projects [3], [4], [6], [1], [5]. (ii) Operating system and file-system-less devices for seamless UX: Once all the data and the code required to run applications by devices is hosted in the network, devices can download and leverage all these assets from the network to operate. We can imagine a scenario where devices only include the basic drivers to access and interact with the network, and use the network capabilities and resources as its decentralised operating and file-system. This would enable seamless User Experience for end-users between devices. All devices include the same basic driver to interact with the network, and all the user-specific information is stored in the network. (iii) A global cloud infrastructure: We refer to the cloud as a global platform able to fulfill all our computational and connectivity needs, when in reality the cloud is a disjoint group of infrastructure providers capable of fulfilling these needs. With our proposed network, the cloud would become a common infrastructure with commodity resources contributed by the globally accessible peers in the network (including data centers).

## VIII. Conclusions

Permissionless computation is one of the missing pieces in the *web3 stack* to have all the tools needed to *"decentralise Internet services"*. There are already proposals to embed computation in decentralised networks like smart contracts, or blockchain networks for computational offloading. Although they are useful and interesting proposals, their computational model is too restrictive to be used for general purpose computation. Other alternatives such as Dfinity's Internet Global Computer are more ambitious but have a longer-term target, and do not really meet the requirement of a fully permissionless general-purpose computation.

In this paper, we proposed a general architecture for a decentralised general-purpose and permissionless computation network based on content-addressing. We have built and presented a fully operational proof-of-concept prototype of the network, leveraging existing components from the *web3 stack*. We have shown that by taking advantage of content-addressing as a core network principle in the decentralised computational network can unlock several interesting use cases "out-of-the-box". Our intuition says that by using content-addressing as a core system primitive would also address scalability concerns.

## References

[1] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[2] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[3] Gilles Fedak, W Bendella, and E Alves. iexec: Blockchain-based decentralized cloud computing. Technical report, Technical Report. 40 pages. http://iex. ec/wp-content/uploads/pdf/iExec-WPv3 ..., 2018.

[4] The golem project whitepaper. https://assets.website-files.com/60005e3965a10f31d245af87/60352707e6dd742743c75764_Golemwhitepaper.pdf.

[5] The fluence distributed computing protocol. https://github.com/fluencelabs/rfcs/blob/main/0-overview.md.

[6] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview seriesconsensus system. whitepaper.

[7] Ipfs network site. https://ipfs.io.

[8] Ipfs compute base code. https://github.com/adlrocha/ipfs-compute.

[9] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 2016.

[10] Manolis Sifalakis, Basil Kohler, Christopher Scherb, and Christian Tschudin. An information centric network for computing the distribution of computations. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, pages 137–146, 2014.

[11] Michał Król and Ioannis Psaras. Nfaas: named function as a service. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pages 134–144, 2017.

[12] Suborbital atmo. https://github.com/suborbital/atmo.

[13] Woodruff T Sullivan III, Dan Werthimer, Stuart Bowyer, Jeff Cobb, David Gedye, and David Anderson. A new major seti project based on project serendip data and 100,000 personal computers. In *IAU Colloq. 161: Astronomical and Biochemical Origins and the Search for Life in the Universe*, page 729, 1997.

[14] Alfonso De la Rocha, David Dias, and Yiannis Psaras. Accelerating content routing with bitswap: A multi-path file transfer protocol in ipfs and filecoin. 2021.

[15] Webassembly specification. https://webassembly.github.io/spec/core/.

[16] Wasi: The webassembly system interface. https://wasi.dev/.

[17] Ipfs lite. https://github.com/hsanjuan/ipfs-lite.