SmartPubSub: Content-based Pub-Sub on IPFS

Pedro Agostinho
INESC-ID / Técnico Lisboa - ULisboa, Portugal

David Dias
Protocol Labs
INESC-ID / Técnico Lisboa - ULisboa, Portugal

Luís Veiga
INESC-ID / Técnico Lisboa - ULisboa, Portugal

Abstract—The InterPlanetary File System (IPFS) is a hypermedia distribution protocol enabling the creation of completely distributed applications. One of the most efficient and effective ways to distribute information is through notifications, with a producer of content (publisher) sharing content with other interested parts (subscribers). IPFS already implements topic-based publish-subscribe systems under an experimental flag. The goal of this work is to advance on that, by developing a content-based publish-subscribe system (with subscriptions as predicates about event content) to disseminate information on top of IPFS in an efficient and decentralized way, leveraging its infrastructure. We design two protocols: ScoutSubs that is completely decentralized; FastDelivery that is centered in the publisher. With these two approaches, we show the different advantages of having each of these protocols simultaneously by comparing ScoutSubs’ full decentralization, and FastDelivery’s centralization at data sources.

I. INTRODUCTION

The InterPlanetary File System [1] was born to help create a decentralized World Wide Web, where users play a central role in the distribution and storage of information without the direct intervention of big corporate organizations. Currently, IPFS is still essentially a file-sharing system operating over a peer-to-peer network. Still, it has an experimental flag a topic-based publish-subscribe system. Furthermore, in the IPFS community, some topic-based systems have also been created, e.g. Pulsarcast [2] and GossipSub [3] (that is the main one being employed). The nonexistence of a content-based publish-subscribe system in IPFS is a significant lacking. Thus, it opens space for our work to offer a content-based, publish-subscribe system on top of IPFS to enhance content selection, delivery, and access performance.

We develop a scalable, flexible and performant content-based publish-subscribe middleware, SmartPubSub, to operate over IPFS. We provide a dynamic semantic-addressing layer (meaning content-addressing a human can understand) where the information is routed through the network depending on the users’ interests, hence improving locality and latency over regular IPFS content retrieval. The publish-subscribe system is what allows such dynamic addressing, meaning users express to the network their interests, and information of interest to them upon production is forwarded towards them.

Previously, publish-subscribe and peer-to-peer have been addressed in common extensively (e.g., [4], [5]), but seldom on such widely deployed web-scale and web-based systems such as IPFS (an ecosystem that we aim to contribute to). In IPFS specifically, publish-subscribe has only been addressed with a simpler, less expressive and flexible topic-based semantics [2], [3], as opposed to the content-based we are proposing.

II. ARCHITECTURE

Our architecture was designed to be integrated with the existing IPFS routing overlay (Kademlia DHT [6], with the modifications proposed by S/Kademlia [7]). But there is a particularity with Kademlia peer ID distribution. In a network with Kademlia as its routing overlay, peers are assigned an ID and establish connections with others based only on ID distancing between them. That makes IPFS peer connection not geographically oriented, increasing the network’s resilience, but reducing performance by allowing long redundant hops like US-CHINA-US-CHINA. This results from the fact that Kademlia is a peer-to-peer structured approach.

To extend IPFS in order to support content-based publish-subscribe there is a spectrum of options between two main ones: i) extend, leverage it in some way, or build another layer structured approach on top of it, or ii) using it only as a bootstrapping mechanism, and then implement a fully non-structured (e.g. gossip-based) approach for the publish-subscribe. We decided for an hybrid approach, i.e.: i) go structured, by taking advantage of the efficiency of the existing structured approach in locating content, and leveraging the propagation paths of messages/content along the overlay; and ii) go unstructured by trying to place popular content for premium users in a geographically oriented way, thus improving locality and performance (latency). Our SmartPubSub is comprised of two subsystems that include each one its specific protocol and cater for different levels of service:

- Decentralized Protocol - ScoutSubs: This protocol provides a content-based publish-subscribe system using the existing IPFS routing tables and adding a filtering structure inspired by the Hermes system [8] to disseminate events from publisher to subscribers based on their content and the content of the subscribers’ subscriptions. ScoutSubs allows its users to possess a global semantic-based knowledge of their network.

- Publisher Centered Protocol - FastDelivery: This protocol’s focus is to deliver events as fast as possible. We provide an application-level multicast, geographically oriented, to provide low latency event delivery. The publisher coordinates its publish-subscribe service and may request its subscribers to disseminate its events making the IPFS’ overlay used only for advertising. This protocol becomes interesting when subscribers are (more) interested in a (specific) source of information instead (than) of a global notion of event content properties/predicates.

Our system’s network stack is comprised of three main layers illustrated in Figure 1. Briefly, at the bottom we have the libp2p host, which represents a node at IPFS and includes
provides a content-based approach. For it shares a topic and a predicate filtering layer over it that lays. The base design of this system was inspired by Hermes [8], publishing-subscribe middleware over IPFS’ content routing over content, i.e., to define a predicate: we combine two types of attributes to characterize interest in that add specific meaning to a predicate. In SmartPubSub, allowing more flexible and richer descriptions and selection (event/message/file) that awards a semantic meaning to it, thus a predicate is an expression assigned to a piece of data (event/message/file) that awards a semantic meaning to it, thus allowing more flexible and richer descriptions and selection of its content (as opposed to assigning it exclusively a tag, category or topic). This expression is composed by attributes that add specific meaning to a predicate. In SmartPubSub, we combine two types of attributes to characterize interest in content, i.e., to define a predicate:

**Topic:** should be single word key phrases capable of capturing the essence of the described data. Common examples can be names of countries, companies, bands, clubs or sports.

**Range Queries:** are attributes with numerical meaning, composed by a characteristic, feature, property and its numerical interval/value. Common examples of these numerical characteristics can be price, temperature, height or dates. For simplicity, we assume there is prior agreement on the units used for each attribute in predicates (e.g. use dollars for prices and Celsius for temperature).

Thus, predicates need to be assigned to events published on the system and to the subscriptions, so that our publish-subscribe may understand who is interested in what. To simplify, predicates are assigned to events by their publishers and to subscriptions by the subscribers.

### B. ScoutSubs Protocol

As mentioned earlier, the ScoutSubs protocol provides a publish-subscribe middleware over IPFS’ content routing overlay. The base design of this system was inspired by Hermes [8], for it shares a topic and a (predicate) filtering layer over it that provides a content-based approach.

**Rendezvous.** The topic-based layer is created by the rendezvous nodes. There are as many rendezvous nodes as attributes, being the one representing the attribute (e.g. football) the closest peer to the key generated by its string (usually hashing). The purpose of these nodes is to provide a point of reference for the subscription forwarding. So, if we have the subscription [football, Tom Brady], we first see which of these attributes has the closest ID to ours, and then we forward the subscription towards it, minimizing the number of hops. On the other hand, the publisher needs to send its events towards all the rendezvous nodes of its event attributes to ensure coverage. We limit the effort to all (possibly hundreds or thousands) subscribers, at the expense of extra effort from publishers for more complex events. Figure 2 illustrates that.

Additionally, the path between the subscribers and rendezvous nodes needs to be backed up. All other functions and properties are inherited directly from Kademlia, i.e. the pathway from the publisher to the rendezvous node only uses information from the Kademlia’s module, not needing any backup mechanism besides the fault tolerance already present at Kademlia’s operations.

**Filtering.** To provide a content-based approach, we implemented a filtering mechanism over the rendezvous nodes. This way, before a subscription is sent towards the rendezvous node, it leaves its filter (subscription) at each intermediate node (i.e. leveraging the existing routes in the structured overlay that converge towards the rendezvous). Thus, when a publisher forwards the event to the rendezvous node, once it arrives at it, it will follow the reverse path of the subscriptions back to all the interested subscribers, as Figure 3 shows, minimizing message replication.

All the filtering information is kept at a filterable which initially is simply a replica of the Kademlia routing table [6]. Upon receiving subscriptions from those nodes, it adds filters to those node’s entries. When receiving a subscription from a new node, it needs to create a new entry in the table and register the filter. Filters upon received can be also merged or ignored, when the result is the same in the forwarding process, to bound the size of the filterables.

**Event Forwarding.** To optimize event forwarding, we add a mechanism that may allow an event to jump as many hops as possible (as a shortcut) without compromising its delivery to all interested subscribers. To achieve this, when a node receives a filter from one peer towards a rendezvous node, it always forwards upstream the option to provide a redirect (jump over itself), if there were no filters forwarded from other peers to...

![Fig. 2. Subscription Forwarding](image)
that same rendezvous, as shown in Figure 4. We need then to keep track of how many filters were forwarded to each rendezvous. If the number of filters is below two, we may provide a shortcut option, but if the number gets equal to or larger than two, we must warn the node upstream that the shortcut is no longer valid.

The first advantage is that we reduce the number of hops on the network, saving bandwidth and reducing event delivery time. The other important point is that the peers that are jumped over no longer have to check their filterables, saving their precious CPU cycles. This fact is even more interesting because shortcuts are used more frequently on unpopular events, thus avoiding searching a filter table with lots of filters. This fact is even more interesting because shortcuts are used more frequently on unpopular events, thus avoiding searching a filter table with lots of filters.

**Ensuring Event Delivery.** When a peer crashes mid-forwarding process, the subscribers downstream may not receive the event. To ensure this does not happen, we implement a tracking mechanism resembling an acknowledgement chain. The challenging part is implementing this in a completely decentralized network without becoming inefficient. Thus, between the publisher and the rendezvous node, the publisher will keep forwarding the event towards the rendezvous node until the latter receives it and sends an acknowledgment back. Before acknowledging the reception and process of the event to the publisher, the rendezvous node will track the event and send tracking requests to all its backups. Tracking means that the rendezvous will check its filterable and create a map with all interested peers of that event. This way, once it receives an acknowledgment from every peer, the event will be considered as (eventually) successfully delivered, since in the worst-case, it will only need to resend the event to the peers that have not confirmed yet.

Incrementally, and in decentralized manner, at all the regular (non-rendezvous) peers downstream, part of this mechanism is repeated: they keep a map with the interested peers, and upon receiving all acknowledgments, they forward their acks upstream. This approach also allows re-sending each event only to those peers who have not received it yet. The intermediate nodes have a passive role, being the rendezvous node, the manager of the re-sending process, the one that forwards the events back if their were not confirmed at all peers before a timeout.

**Protocol Maintenance.** To maintaining the protocol working over time, there needs to be management of the filtering information. ScoutSubs does not allow explicit unsubscribing operations due to a subscription being a filter that can be merged or omitted. Therefore, we need to perform garbage collection of subscriptions no longer relevant by executing a refreshing routine (requiring subscribers to resubscribe to topics and predicates periodically).

### C. FastDelivery Protocol

As previously mentioned, FastDelivery’s main objective is to disseminate events as fast as possible. To do so, we need to go beyond IPFS' overlay structure and, in part, centralize the event dissemination at the publisher. The publisher can still provide events to the subscribers via ScoutSubs but must manage a group of premium subscribers, to which it sends events directly, or in at most two geographically oriented hops (overlay hops). Premium subscribers need to provide the publisher their endpoint, location (Region/Country), and resources (network and CPU-wise).

**Motivation.** The reason of designing this protocol alongside ScoutSubs was to reflect when a more centralized approach to disseminate information is actually the best option. In this case, when a subscriber is not searching just for a topic/content of a publication but a particular publisher with a certain reputation or popularity, this alternative becomes an option offering better performance, i.e. lower latency.

**Overall Design.** In this protocol, a publisher manages multicast groups: a data structure containing its interested subscribers and their subscriptions predicates. Each multicast group is represented by its publisher ID and the group’s predicate (apple/france/price[0,1]).

To manage the multicast group’s subscribers and recruit them if the publisher needs assistance, we decided to group subscribers into regions, ordered by capacity. Thus, once one publisher gets too many subscribers, the most powerful one gets recruited as a helper to assist the publisher.

To organize the subscribers’ predicates, we save their subscriptions in a simple list, in the case of all predicate’s attributes being of the topic type. If there are any range type attributes, we use a binary tree to organize the subscription. If the multicast group’s predicate has several range attributes, it will need the same number of range trees and go through their query results.

For simplicity, we consider a helper’s capacity as the number of other subscribers it can help the publisher manage. After agreeing to assist the publisher, the helper will only support the structure that manages the subscriptions’ predicates to forward the publisher’s events to the interested subscribers. This support structure is a list or range trees with the subscribers delegated by the publisher.

In terms of advertising a publisher’s multicast group, we use advertisement boards at the rendezvous nodes of the attributes.
of the group’s predicate, to ensure it can be discovered by any subscriber. But a publisher may also prefer to keep its endpoint address private.

III. RELATED WORK

Peer-to-Peer Content Distribution. A P2P system [9] is commonly seen as one for decentralized sharing of computational resources over the network, maintaining proper functioning even in the presence of node failures, connectivity problems, and churn. P2P networks are highly scalable and fault-tolerant because each node can act both as server and client, and so, the number of servers grows linearly with the number of clients, preventing bottlenecks. IPFS inherits the advantages of a P2P system such as Kademlia [6] by virtue of being purely decentralized (and thus highly scalable) and structured (therefore with efficient look-up). But users can only access content in IPFS whose specific CID they already know (or get through a name system) which is restrictive and inflexible. Existing topic-based publish-subscribe offers on IPFS [2], [3] only partially address such limitations and inefficiency (i.e. inability to define interest in content in fine-grained manner).

Publish-Subscribe. Publish-Subscribe [10] is a message paradigm that provides complete decoupling between data consumers and data producers both in time, space and synchronization. SDN-like publish-subscribe [11] creates specific overlays regarding each topic for event propagation, with SDN-based systems in mind. HoP-and-Pull (HoPP) addresses topic-based publish-subscribe for IoT ecosystems [12], where network decentralization is limited, employing RIOT-OS and IoT LoWPAN. Topic-connected overlays (TCO) [5] introduces k-topic-connected overlays to ensure efficient propagation topologies and reliable event propagation, as long as fewer than k nodes interested in the same topic fail simultaneously. They are complementary to SmartPubSub as they do not address the web ecosystem and are not content-based (neither Pulsarcast [2], GossipSub [3] over IPFS). SmartPubSub also builds multi-topic-overlays around rendezvous (one per topic in subscriptions), employs replication, and optimizes event forwarding by merging filters (content-based predicates).

POSSUM [13] targets IoT, using triple-vectors (combining topics and queries in knowledge graphs with RDF) to optimize subscription filtering with clustering and normalization. PSDG [14] introduces delivery guarantees, without hampering throughput, in content-based publish-subscribe systems with distributed broker/mesh architectures. PopSub [15] takes into account event popularity and brokers’ resources in event propagation to improve resource efficiency in brokers, and reduce delivery latency in high-load. These systems are complementary to SmartPubSub. Although content-based, they are not designed to address decentralized web-scale environments, such as SmartPubSub (based on Kademlia/IPFS). They optimize subscription filtering but employ centralized coordination in IoT, or distributed/hierarchical broker topology (wholly known in advance), to achieve high-throughput and not geo-scale.

IV. CONCLUSIONS

We propose a novel publish-subscribe middleware, Smart-PubSub, to provide global content/semantic-addressing over IPFS. It offers two protocols: ScoutSubs and FastDelivery. It is not only physically content-based oriented where information is stored (as in IPFS), but one that truly provides a semantic content-based approach, with information forwarded through the web depending on its semantic content and user interest. The results (available in full in [16]) show that decentralizing the web infrastructure can have more benefits, but it is not a perfect solution. With that in mind, we presented FastDelivery to showcase where some centralization can be advantageous, more efficient and useful for IPFS users when they are interested in both the source and content of the events.

Acknowledgements This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020.