Peer-to-Peer Permissionless Consensus via Authoring Reputation

Francisco Sant'Anna *Department of Computer Science, Rio de Janeiro State University*

Abstract—Public Internet forums suffer from excess and abuse, such as SPAM and fake news. Centralized platforms employ filtering and anti-abuse policies, but imply full trust from users. We propose a permissionless Sybil-resistant peer-to-peer protocol for public communications. Our main contribution is a reputation system that moderates content and, at the same time, delivers network consensus. We can trace a parallel with Bitcoin: new posts create reputation (vs proof-of-work), likes and dislikes transfer reputation (vs transactions), and aggregate reputation determines consensus (vs longest chain). The reputation mechanism depends exclusively on the human authoring ability (*proof-of-authoring*), which is slow and scarce, thus suitable to establish consensus. Human creativity contrasts with plain economic resources (e.g., proof-of-work/ stake), which do not appraise social interactions and also tend to concentrate over the time. As an application example, we prototype a permissionless distributed version control system that, based on consensus, resolves conflicts automatically.

Index Terms—Bitcoin, blockchains, CRDT, distributed consensus, peer-to-peer, publish-subscribe, reputation system, VCS

1 Introduction

Content publishing in Internet forums and social media is increasingly more centralized in a few companies (e.g. Facebook and Twitter). These companies benefit from closed protocols and network effects to keep users locked in their platforms [1], [2]. On the one hand, these companies offer free storage, friendly user interfaces, and robust access. On the other hand, they concentrate more power than required to operate by collecting users' data and "algorithmizing" consumption. Peer-to-peer alternatives [3] eliminate intermediaries, but strive to enforce overall state consistency while dealing with malicious users, given the decentralization of authority and infrastructure.

In an ideal Internet forum, all messages or posts (i) reach all users, (ii) are delivered in a consistent order, and (iii) are respectful and on topic. In a centralized system, items (i) and (ii) are trivially achieved assuming availability and delivery order in the service, while for item (iii), users have to trust the service to moderate content. In a decentralized setting, however, none of these demands are easily accomplished. A common approach in gossiping protocols is to proactively replicate and disseminate posts among peers until they reach all users [3]. However, this approach does not guarantee consensus since posts can be received in conflicting orders [4]. Consensus is key to eradicate Sybils, which are the major threats to open forums and decentralized collaborative applications in general: without consensus, it is not possible, *at the protocol level*, to distinguish between correct and malicious users.

Consensus is particularly challenging to the point that decentralized protocols partially abdicate of it. On the one hand, federated protocols [5] offer *multi-user/single-node consensus*, in which multiple users can exchange messages consistenlty within a single trusted server, but not globally across multiple servers. On the other hand, a protocol like Scuttlebutt [6] offers *single-user/multi-node consensus*, in which a single user has full authority over its own content across machines, but multiple users cannot reach consensus even in a local machine. Our goal is to provide *multi-user/multi-node consensus* (just *consensus*, from now on) in the context of decentralized content publishing.

Bitcoin [7] is the first permissionless protocol to resist Sybils through consensus. Its key insight is to rely on a scarce resource—the *proof-of-work*—to establish consensus. The protocol is resistant to Sybils because it is expensive to write to its unique timeline (either via proof-of-work or transaction fees). However, Bitcoin and cryptocurrencies in general have many drawbacks in the context of content publishing: (i) they enforce a unique timeline to preserve value and immunity to attacks; (ii) they lean towards concentration of power due to economy of scale effects; and (iii) they impose an external economic cost to use the protocol. These issues threaten our original decentralization goals. In particular, a unique timeline implies that all Internet content should be subject to the same consensus rules, which neglects all subjectivity that is inherent to social content. Another limitation of cryptocurrencies is that it is not possible to revoke content in the middle of the blockchain, which is inadmissible considering illegal content (e.g., hate speech).

In this work, we adapt Bitcoin's idea of a scarce resource to reach consensus to context of content publishing. Our first contribution is to recognize the actual published contents as the protocol scarce resources, since they require human work (*proof-of-authoring*). Work is manifested as new posts, which if approved by others, reward authors with reputation tokens named *reps*, which in turn serve as a means to evaluate other posts in the same forum with likes and dislikes. With such proof-of-authoring mechanism, token generation is expensive, while verification is cheap and made by multiple users (akin to Bitcoin's proof-of-work). Our second contribution is to allow that users create diversified forums of interest (instead of a singleton

blockchain), each counting as an independent timeline with its own subjective consensus etiquette. Due to decentralization, posts in timelines form a causal graph with only partial order, which we promote to a total order based on the reputation of authors. The consensus order is fundamental to detect conflicting operations, such as likes with insufficient *reps* (akin to Bitcoin's double spending). Our third contribution is to support content removal without compromising the integrity of the decentralized blockchain. Users have the power to revoke posts crossing a limit of dislikes, and peers are forced, at the protocol level, to remove associated payloads, only forwarding metadata to preserve integrity. We integrated the proposed consensus algorithm into Freechains, a practical peer-to-peer content dissemination protocol [8].

As an application example, we prototyped a permissionless distributed version control system (DVCS) that relies on consensus to apply automatic merges. A DVCS is relevant because (i) it is a collaborative application, (ii) with commits that need evaluation from users, and (iii) with merges that require human intervention, which we automate. Hence, as a forth contribution, we show how the consensus mechanism can empower quasi conflict-free replicated data types (CRDTs [9]) when they do encounter conflicting operations.

In summary, we propose (i) a consensus mechanism based on proof-of-authoring, (ii) for independent user-generated blockchains, (iii) which supports content removal while preserving integrity, and (iv) which can automate conflict resolution in *quasi-CRDTs*. As a main limitation, the forum graphs are ever growing data structures that carry considerable metadata overhead. The costs to store and validate posts increase over time This is a fundamental limitation shared with Over the time, these costs We evaluate TODO: performance of the algorithm and overhead of the blockchain In addition, they require per-block validations that may become a bottleneck for real-time applications.

The rest of the paper is organized as follows: In Section 2, we introduce the basic functionalities of Freechains to create, evaluate, and disseminate posts. In Section 3, we describe the reputation and consensus mechanism applied to permissionless public forums. In Section 4, we discuss some correspondences with CRDTs, and prototype a simple DVCS. In Section 5, we compare our system with publish-subscribe protocols, federated applications, and fully peer-to-peer systems. In Section 6, we conclude this work.

2 FREECHAINS

Freechains [8] is an unstructured peer-to-peer topic-based publish-subscribe protocol, in which each topic or *chain* is a replicated *Merkle Directed Acyclic Graph* [10] (just *DAG*, from now on). The DAG represents the causal relationships between the messages, whose cryptographic links ensure persistence and self-certification of the whole chain. The protocol operation is typical of publish-subscribe systems: an author publishes a post to a chain, and the other users subscribed to the same chain eventually receive the message by synchronizing DAGs.

The goals of this section are twofold: (i) to depict the chain DAGs as result of the basic protocol operations; and (ii) to illustrate how permissionless protocols inevitably require some form o Sybil resistance.

Freechains supports multiple arrangements of public and private communications, which are detailed in Table 1. In this section, we operate a *private group* to describe the basic behavior of chains without consensus. We use the actual command-line tool provided by the protocol to guide the discussion through concrete examples. At the end of this section, we also exemplify a *public identity* chain for the sake of completeness. In Section 3, we focus on the behavior of *public forums*, which involves untrusted communication between users and require the proposed reputation and consensus mechanism.

All Freechains operations go through a *daemon* (akin to Bitcoin's full nodes) that validates posts, links the DAGs, and communicates with other peers to disseminate the graphs. The command that follows starts a daemon in the background to serve further operations:

```
> freechains-daemon start '/var/freechains/' &
```

The actual chain operations use a separate client to communicate with the daemon. The next sequence of commands (i) creates a shared key, (ii) joins a private group chain (prefix $), and (iii) posts a message into the chain:

```
> freechains keys shared 'strong-password'  <- (i)
A6135D..   <- returned shared key
> freechains '$family' join 'A6135D..'      <- (ii)
42209B..   <- hash representing the chain
> freechains '$family' post 'Good morning!' <- (iii)
1_EF5DE3.. <- hash representing the post
```

A private chain requires that all participants use the same shared key to join the group. A `join` only initializes the DAG locally in the file system, and a `post` also only modifies the local structure. No communication occurs at this point. Figure 1.A depicts the state of the chain after the first post. The genesis block with height 0 and hash `42209B..` depends only on the arguments given to `join`. The next block with height 1 and hash `EF5DE3..` contains the posted message.

Freechains adheres to the *local-first* software principle [11], allowing networked applications to work locally while offline. Except for synchronization, all other operations in the system only affect the local replica. In particular, joining a chain with the same arguments in another peer results in the same genesis state, even if the peers have never met before. Hence, before synchronizing, other peers have to initialize the example chain with the exactly same steps as above.

In Freechains, the synchronization operation of chain DAGs is explicit, in pairs, and unidirectional. The command `recv` asks the daemon in *localhost* to connect to daemon in *remote-ip* to receive all missing blocks from there:

```
> freechains '$family' recv '<remote-ip>'
1/1  <- one block received from <remote-ip>
```

If applied in the new peer, the command above would put it in the same state as the original peer in Figure 1.A. The complementary command `send` would synchronize the DAG in the other direction. Note that Freechains does not synchronize peers automatically. There are no preconfigured peers, no root servers, no peer discovery. All connections happen through the `send` and `recv` commands which have to specify the peers explicitly. In this sense, Freechains is

Type	Prefix	Arrangement	Behavior	Examples
Private Group	$	Private 1↔1, N↔N, 1↩	Trusted groups (friends or relatives) exchange encrpyted messages among them. The communication can be in pairs (1↔1), groups (N↔N) or individual (1↩).	- E-mails - WhatsApp groups - Backup of documents.
Public Identity	@	Public 1→N, 1←N	A public identity (person or organization) broadcasts authenticated content for a target audience (1→N) with optional feedback (1←N).	- News sites - Streaming services - Public profiles in social media
Public Forum	#	Public N↔N	Participants with no mutual trust communicate publicly.	- Q&A forums - Chats - Consumer-to-consumer sales

TABLE 1
The three types of chains and arrangements in Freechains.

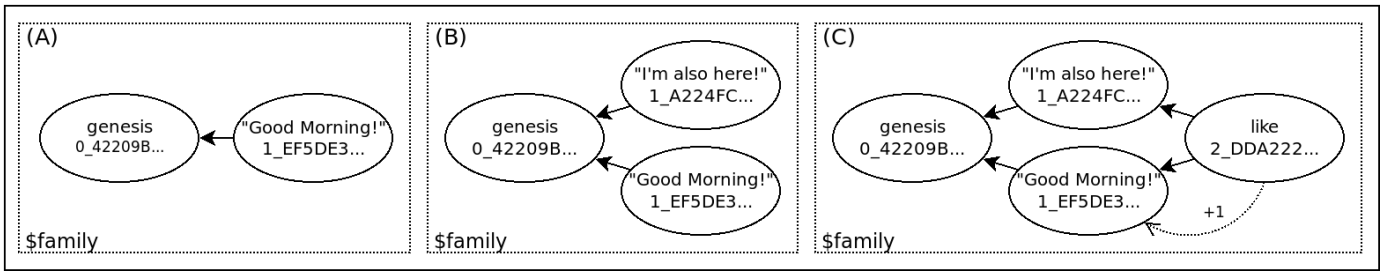

Fig. 1. Three DAG configurations. (A) Single head pointing to genesis block. (B) Fork with heads pointing to genesis block. (C) Like pointing to previous heads and also to its target.

conceptually a *pubsub* on how users publish and consume content, but it still requires extra network automation.

In order to query the state of the replica, the next sequence of commands checks the hash(es) of the block(s) at the head of the local DAG (the latest blocks), and then reads the payload of the single head found:

```
> freechains '$family' heads
1_EF5DE3..
> freechains '$family' payload '1_EF5DE3..'
Good morning!
```

The presented commands to join, post, synchronize, and query the chains are sufficient to create decentralized applications that publish and consume content.

We now focus on how typical chains are not simple sequential lists of posts, but actually become DAGs with multiple heads that demand a consensus mechanism. The main reason is because the network is inherently concurrent and users are encouraged to work locally. Continuing with the example in Figure 1, suppose that the new peer posted a message before the `recv` above, when its local DAG was still in its genesis state. In this case, as illustrated in Figure 1.B, the resulting graph after the synchronization would now contain two blocks with height 1. Note that forks in the DAG create ambiguity in the order of messages, which is a fundamental obstacle to reach consensus. In private chains, we can apply simple methods, such as relying on the local source timestamps of the blocks. However, in public forums, a malicious user could modify his local time to manipulate the order of messages.

To conclude the basics of the chain operations, users can rate posts with *likes* and *dislikes*, which can be consulted later:

```
> freechains '$family' like '1_EF5DE3..'
2_BF3319..
> freechains '$family' reps '1_EF5DE3..'
1  <-- post received 1 like
```

As illustrated in Figure 1.C, a like is a regular block with an extra link to its target. Every new block points to the previous heads, establishing a causal logical timeline in the chain. For instance, the JSON that follows represents the block `2_DDA222..` with the backs and extra like links:

```
{
  "id":    "2_DDA222..",  // block hash id
  "backs": ["1_EF5..","1_A22.."] // back links
  "time":  1650722072223, // source timestamp
  "data":  "E95DBF.."     // hash of the payload
  "like":  "1_EF5DE3.."   // like link (optional)
}
```

In private groups, like operations are unlimited and behave as a means to quantify engagement, much like in typical centralized systems. In public forums, however, likes are restricted, have to be signed by users, and are at the core of our proposed consensus algorithm.

For the sake of completeness, Freechains also supports public identity chains (prefix @) with owners attached to public/private keys:

```
> freechains keys pubpvt 'other-password'
EB172E.. 96700A..   <- public and private keys
> freechains '@EB172E..' join
F4EE21..
> freechains '@EB172E..' post 'This is Oprah' \
   --sign='96700A..'
1_547A2D..
```

In the example, a public figure creates a key pair and joins an identity chain attached to her public key. Every post in the chain needs to be signed with her private key to be accepted in the network.

Freechains is around 1500 LoC in Kotlin and is publicly available[1]. The binary for the JVM is around $6Mb$ in size and works in Android and most desktop systems.

Note that the basic operations of Freechains to (i) create decentralized identities, (ii) publish content-addressable data, (iii) maintain Merkle DAGs, and (iv) synchronize peers are not new. However, without extra restrictions, any number of users at any number or peers might inadvertently or maliciously post any kind of content and rate posts any number of times, thus threatening the value of the protocol. As discussed in the Introdution, Sybil resistance through consensus is a key requirement to combat abuse. In the next section, we discuss the consensus mechanism of Freechains to support public forums.

3 Reputation and Consensus Mechanism

In the absence of moderation, permissionless peer-to-peer public forums are impractical, mostly because of Sybil attacks. For instance, it should take a few seconds to generate thousands of fake identities and SPAM millions of messages into the system. For this reason, we propose a reputation system that works together with a consensus algorithm to eradicate Sybils and make peer-to-peer public forums practical.

Section 3.1 describes the overall reputation and consensus mechanism, which can be applied to any public forum system that uses DAGs to structure its messages. Section 3.2 describes the concrete rules we implemented for public forums in Freechains.

3.1 Overall Design

In the proposed reputation system, users can spend tokens named *reps* to post and rate content in the forums: a `post` initially penalizes authors until it consolidates and counts positively; a `like` is a positive feedback that helps subscribers to distinguish good content amid excess; a `dislike` is a negative feedback that revokes content when crossing a threshold. Table 2 summarizes the reputation operations and their goals. To prevent Sybils, users with no *reps* cannot perform these operations, requiring a like from any other user already in the system. Hence, since likes are zero-sum operations, malicious users have no incentives to invite Sybils into the system. The only way to generate new *reps* is to post content that other users approve, which demands non-trivial work immune to automation. Therefore, *reps* are subject to some sort of scarcity, which prevents Sybils from acting.

Bitcoin employs CPU proof-of-work to mitigate Sybil attacks. However, CPU or alternative extrinsic resources are not evenly distributed among humans and/or machines, specially considering that most communications now use battery-powered devices. In the context of public forums, we understand that the human authoring ability is already an intrinsic resource that we can take advantage of. Creating new content is hard and takes time, but is comparatively easy to verify and rate. Therefore, in order to impose scarcity, we determine that only content authoring generates *reps*, while likes and dislikes just transfer *reps* between users.

1. http://www.freechains.org

Operation	Effect	Goal
Emission	Old posts award *reps* to author.	Encourage content authoring.
Expense	New posts deduct *reps* from author temporarily.	Discourage excess of content.
Transfer	Likes & dislikes transfer *reps* between authors.	Highlight content of quality. Combat abusive content.

TABLE 2
General reputation operations in public forums.

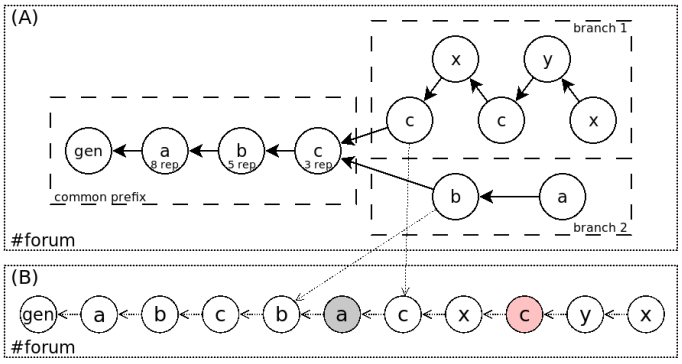

Fig. 2. (A) A public forum DAG with a common prefix and two branches. (B) Total order between blocks of the DAG after consensus.

Nonetheless, scarce operations are not yet sufficient because they demand consensus to establish an order in time across the network. As an example, consider a malicious author with a single unit of *reps* trying to post an arbitrary number of new messages in parallel using multiple peers. According to the *Expense* rule of Table 2, only one of these messages should be accepted. However, without consensus, it is not possible to globally determine which message to accept, since each peer would supposedly accept the first message it sees. Therefore, we need the same message ordering within forum chain DAGs across peers in order to validate operations consistently.

Our solution is to favor DAG forks with posts from users that constitute the majority of the reputation in the network. These forks have more associated work from active users and are analogous to longest chains in Bitcoin. Also, the operations in these forks have already been verified and we should avoid extra recalculations due to reorderings. Technically, given that forums chains are DAGs, we can adapt a topological sorting algorithm to favor reputation when deciding between branches. Going back to the malicious user example, the messages in parallel would appear as forks in the forum DAG. Only the message in the fork with more combined work would be accepted, while all other Sybil messages would be rejected. This way, in order to misuse a public forum, a malicious user first needs to cooperate in the same magnitude, which requires authoring work that contradicts his original intent.

Figure 2.A illustrates the consensus criteria. A public forum DAG has a common prefix with signed posts from users a, b, and c. Let's assume that within the prefix, users a and b have contributed with better content and have more reputation combined than c has alone (i.e., $8 + 5 > 3$). After the prefix, the forum forks in two branches: in `branch-1`, only user c remains active and we see that new users x

and y, with no previous reputation, generate a lot of new content; in `branch-2`, only users a and b participate but with less activity. Nonetheless, `branch-2` would be ordered first because, before the forking point, a and b have more reputation than c, x, and y combined. User c here might represent a malicious user trying to cultivate fake identities x and y in separate of the network during weeks to accumulate *reps*.

Figure 2.B indicates the consensus order between blocks in the forum. All operations in `branch-2` appear before any operation in `branch-1`. Note that the ordered list after consensus exists for accountability purposes, and is only a view of the primary DAG structure. At any point in the consensus timeline, if an operation fails, all remaining blocks in the offending branch are removed from the primary DAG. As an example, suppose that the last post by a (in gray) is a dislike to user c. Then, it's possible that the last post by c (in red), now with *0 reps*, is rejected together with all posts by y and x in sequence. Note that in a Merkle DAG, it is not possible to remove only the block with the failing operation, instead, we need to remove the remaining branch completely, as if it never existed. Note also that users in the branch with more reputation can react to attacks even after the fact. For instance, users a and b can pretend that they did not yet see `branch-1` and post extra dislikes to user c from `branch-2` so that a further merge removes all blocks of `branch-1` from the DAG.

There are some other relevant considerations about forks and merges: Peers that first received branches with less reputation will need to reorder all blocks starting at the forking point. This might involve removing content in the end-user software. This behavior is similar to blockchain reorganization in Bitcoin, when a peer detects a new longest chain and disconsiders old blocks. Likewise, peers that first saw branches with more reputation just need to put the other branch in sequence and do not need to recompute anything. This behavior is expected to happen in the majority of the network. Unlike Bitcoin, forks are not only permitted but encouraged due to the local-first software principle. However, the longer a peer remains disconnected, the more conflicting operations it may perform, and the higher are the chances of rejection when rejoining.

As a counterpoint to the consensus order in Figure 2.B, maybe users a and b have abandoned the chain for months, and thus `branch-1` is legit. In this case, a and b might be the ones trying to take over the chain. Yet another possibility is that both branches are legit but became disconnected for a long period of time. It is simply impossible to judge with confidence which is the case. Nevertheless, it is unacceptable that a very old branch affects a long active chain. For this reason, the consensus algorithm includes an extra constraint when merging: long-lasting branches do not merge, creating *hard forks* in the network. A hard fork occurs when a local branch crosses a predetermined and irreversible threshold, e.g., *7 days* or *100 posts* of activity. In this case, regardless of the remote branch reputation, the local branch takes priority and is ordered first. This situation is analogous to a hard fork in Bitcoin and the branches will never synchronize again. More than simply numeric disputes, hard forks represent social conflicts in which reconciling branches is no longer possible.

In summary, these are the rules to merge a branch from a remote machine `j` into a branch from a local machine `i`. The first rule to apply determines the merging result:

a) `i` is ordered first if it crosses an activity threshold (e.g., *7 days* or *100 posts*), regardless of `j`.
b) `i` or `j` is ordered first, whichever has more reputation in the common prefix.
c) otherwise, branches are ordered by an arbitrary criteria, such as lexicographical order of the block hashes immediately after the common prefix.

A fundamental drawback of Merkle DAGs is that all replicas in the system need to store the complete graph in order to synchronize and verify new blocks. Tree pruning techniques allow to remove parts of the graph to save space [12]. The hard fork threshold in the consensus algorithm also allows to prune the chain DAGs, at least for lightweight clients in resource-constrained devices. However, these devices can no longer verify older forks and need to delegate trust to more powerful peers. A more pragmatic approach, but which requires cooperation among users, is to revoke past posts, which deletes associated payloads to save some space. This approach is more feasible in private groups and public identities chains, tough.

3.2 Public Forum Chains

We integrated the proposed reputation system in the public forums of Freechains to support content moderation and enforce consensus in the chains. Table 3 details the concrete rules which are discussed as follows. Authors have to sign their posts in order to be accounted by the reputation system and operate in the chains. The example that follows creates an identity whose public key is assigned as the pioneer in a public chain (prefix #):

```
> freechains keys pubpvt 'pioneer-password'
4B56AD.. DA3B5F..
> freechains '#forum' join '4B56AD..'
10AE3E..
> freechains '#forum' post --sign='DA3B5F..' \
   'The purpose of this chain is...'
1_CC2184..
```

The `join` command in rule `1.a` bootstraps a public chain, assigning *30 reps* equally distributed between an arbitrary number of pioneers indicated through their public keys. The pioneers shape the initial culture of the chain with the first posts and likes, while they gradually transfer *reps* to other authors, which also transfer to other authors, expanding the community. The `post` command in sequence is signed by the single pioneer (in this example) and indicates the purpose of the chain for future users.

The most basic concern in public forums is to resist Sybils abusing the chains. Fully peer-to-peer systems cannot rely on logins or CAPTCHAs due to the lack of a central authority. Viable alternatives include (i) building social trust graphs, in which users already in the community vouch for new users, or (ii) imposing economic costs for new posts, such as proof of work.

We propose a mix between trust graphs and economic costs. Rule `4.a` imposes that authors require at least *1 rep* to post, effectively blocking Sybil actions. To vouch for new users, rule `3.a` allows that an existing user likes a newbie's post to unblock it, but at the cost of *1 rep*. This

Operation	Rule num	Rule name	Description	Observations
Emission	1.a	pioneers	Chain join counts *+30 reps* equally distributed to the pioneers.	`[1.b]` A post takes 24 hours to consolidate. New posts during this period will not be rewarded later. Only after this period, the next post starts to count 24 hours.
	1.b	old post	Consolidated post counts *+1 rep* to author.	
Expense	2	new post	New post counts *-1 rep* to author temporarily.	The discount period varies from 0 to 12 hours and is proportional to the sum of authors' reps in subsequent posts. It is 12 hours with no further activity. It is zero if further active authors concentrate at least 50% of the total reputation in the chain.
Transfer	3.a	like	Like counts *-1 rep* to origin and *+1 rep* to targets.	The origin is the user signing the operation. The targets are the referred post and its corresponding author. If a post has at least 3 dislikes and more dislikes than likes, then its contents are hidden. `[3.b]` Users can revoke their own posts with a single dislike.
	3.b	dislike	Dislike counts *-1 rep* to origin and *-1 rep* to targets.	
Constraints	4.a	min	Author requires at least *+1 rep* to post.	
	4.b	max	Author is limited to at most +30 reps.	
	4.c	size	Post size is limited to at most 128Kb.	

TABLE 3
Specific reputation rules for public forum chains in Freechains. The chosen constants ($30\ reps$, $24h$, etc) are arbitrary and target typical Internet forums with moderate traffic. A future revision of the protocol could support them as chain parameters.

cost prevents that malicious members unblock new users indiscriminately, which would be a breach for Sybils. For the same reason, rule `2` imposes a temporary cost of *1 rep* for each new post. Note that the pioneers rule `1.a` solves the chicken-and-egg problem imposed by rule `4.a`: if new authors start with no *reps*, but require *reps* to operate, it is necessary that some authors have initial *reps* to boot the chains.

In the next sequence of commands, a new user joins the same public chain and posts a message, which is welcomed with a like signed by the pioneer:

```
> freechains keys pubpvt 'new-author-password'
503AB5.. 41DDF1..
> freechains '#forum' join '4B56AD..'
10AE3E..     <-- same pioneer as before
> freechains '#forum' post 'Im a newbie...' \
   --sign='41DDF1..'
2_C3A40F..   <-- blocked post
> freechains '#forum' like '2_C3A40F..' \
   --sign='DA3B5F..'
3_59F3E1..
```

Note that chains with the same name but different pioneers are incompatible because the hash of genesis blocks also depend on the pioneers' public keys.

Figure 3 illustrates the chain DAG up to the like operation. The pioneer starts with *30 reps* (rule `1.a`) and posts the initial message. New posts penalize authors with *-1 reps* during at most 12 hours (rule `2`), which depends on the activity succeeding (and including) the new post. The more activity from reputed authors, the less time the discount persists. In the example, since the post is from the pioneer controlling all *reps* in the chain, the penalty falls immediately and she remains with *30 reps*. This mechanism limits the excess of posts in chains dynamically. For instance, in slow technical mailing lists, it is more expensive to post messages in sequence. However, in chats with a lot of active users, the penalty can decrease to zero.

Back to Figure 3, a new user with *0 reps* tried to post a message (hash `C3A40F..`) and is blocked (rule `4.a`), as the red

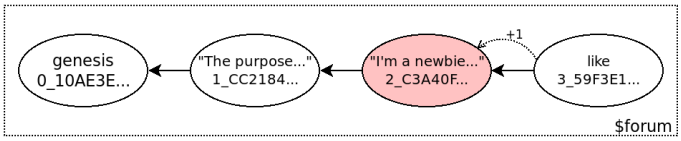

Fig. 3. The `like` approves the newbie message into the `#forum` DAG.

background highlights. But the pioneer liked the blocked message, decreasing herself to *29 reps* and increasing new user to *1 rep* (rule `3.a`). Note that the newbie post is not penalized (rule `2`) because it is followed by the pioneer like, which still controls all *reps* in the chain.

Note that with no additional rules to generate *reps*, the initial *30 reps* would constitute the whole "chain economy" forever. For this reason, rule `1.b` awards authors of new posts with *1 rep*, but only after 24 hours. This rule stimulates content creation and grows the economy of chains. The 24-hour period gives sufficient time for other users to judge the post before awarding the author. It also regulates the growth speed of the chain. In Figure 3, after 1 day, the pioneer would now accumulate *30 reps* and the new user *2 reps*, growing the economy in *2 reps* as result of the two consolidated posts. Note that rule `1.b` awards at most one post of each author at a time. Hence, new posts during the 24-hour period will not award each of them with extra *reps*. Note also that rule `4.b` limits authors to at most *30 reps*, which provides incentives to spend likes and thus decentralize the network.

Likes and dislikes (rules `3.a` and `3.b`) serve three purposes in the chains: (i) welcoming new users, (ii) measuring the quality of posts, and (iii) revoking abusive posts (SPAM, fake news, illegal content, etc). Access to chains is permissionless in the sense that the actual peers and identities behind posts are irrelevant for acceptance. Instead, it is the quality of content that is judged and accounted in the system. The reputation of a given post is the difference between its likes and dislikes, which can be used in end-user

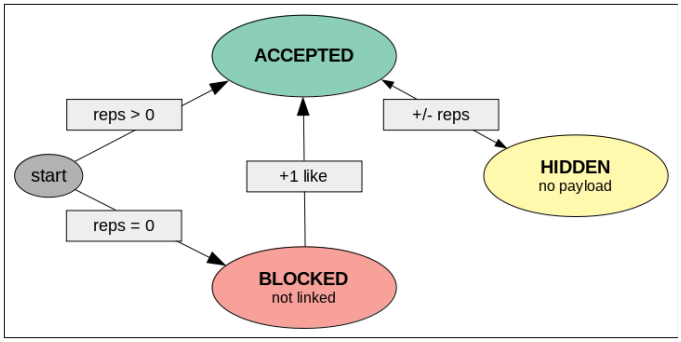

Fig. 4. State machine of posts: `BLOCKED` posts are not linked in the DAG. `ACCEPTED` posts are linked and retransmitted. The payload of `REVOKED` posts are not retransmitted.

software for filtering and highlighting purposes. The quality of posts is subjective and is up to users to judge then with likes, dislikes, or simply abstaining. On the one hand, since *reps* are finite, users need to ponder to avoid indiscriminate expenditure. On the other hand, since *reps* are limited to at most *30 reps* per author (rule `4.b`), users also have incentives to rate content. Hence, these upper and lower limits work together towards the quality of the chains. Note that a dislike shrinks the chain economy since it removes *reps* from both the origin and target. As detailed next, the actual contents of a post may be revoked if it has at least 3 dislikes, and more dislikes than likes (rule `3`). However, considering that *reps* are scarce, dislikes are encouraged to combat abusive behavior, but not to eliminate divergences of opinion.

A post has three possible states: `BLOCKED`, `ACCEPTED`, or `REVOKED`. Figure 4 specifies the transitions between states. If the author has reputation, a new post is immediately `ACCEPTED` in the chain. Otherwise, it is `BLOCKED` and requires a like from another user. Blocked posts are not considered part of the chain DAG in the sense that new posts do not link back to it. In addition, peers are not required to hold blocked posts and neither retransmit them to other peers. However, if blocked posts are not disseminated, new users will never have the chance to be welcomed with a like. A reasonable policy is to hold blocked posts in a temporary bag and retransmit them for some visibility in the network. Rule `4.c` limits the size of posts to at most *128Kb* to prevent DDoS attacks using gigantic blocked posts. Once accepted, a post becomes part of the chain and can never be removed again, since Merkle DAGs are immutable by design. However, if the number of dislikes exceeds the threshold (rule `3`), the block becomes `REVOKED` and its payload is not retransmitted to other peers. Note that a block hash depends only on its payload hash, so it is safe to remove the actual payload as long as one can prove its revoked state. Later, if the post receives new likes, it means that the payload is still known somewhere and peers can request it when synchronizing again. We consider that revoking posts is fundamental in the context of content publishing, and thus, an important contribution of this work.

3.3 The Consensus Algorithm

As discussed in Section 3.1, the consensus algorithm needs to be at the core of the protocol to provide the same message ordering across peers, and thus, validate publications consistently. Conceptually, for every message that needs to be published in a chain, the protocol (i) executes the consensus algorithm over its Merkle DAG,

The algorithm starts at the DAG heads, which are the latest blocks with no mutual causal order, and run towards the genesis block.

- complexity - recursive, no cache, no memoization, load/unload everything

3.4 Experiments

In this section, we perform some experiments to evaluate the consensus performance and the practicability of the protocol as a whole. We took two publicly available forums from the *Internet Archive* and simulate their behavior as if they were using Freechains: (i) a chat channel from the Wikimedia Foundation[2]; and (ii) Usenet's *comp.compilers* newsgroup[3]. Chats and newsgroups represent typical public forums with (i) faster interactions with shorter payloads, and (ii) slower interactions with larger payloads.

- lockdown strategy ok?

Discussion: - do not scale - restart - cannot evaluate behavior - dislike is immediate - users are not aware of the policies - - no images, videos - 128k enough for memes?

We take existent public forums from the Internet Archive

4 Correspondences with CRDTs

Conflict-free replicated data types (CRDTs) [13] serve as a robust foundation to model concurrent updates in collaborative local-first applications [11]. However, CRDTs are not a panacea and often require human intervention to solve general conflicts, such as merge conflicts in version control systems (VCSs). Our observation is that, since the proposed consensus algorithm already relies on human interactions, we can use it to resolve conflicts automatically.

We propose a three-layered CRDT scheme to build distributed collaborative applications: state-based CRDTs at transport layer (CvRDTs), operation-based CRDTs at application layer (CmRDTs), and CRDTs with arbitrary operations after consensus is applied.

At the transport layer, Merkle DAG chains are trivial CvRDTs because they converge to the same state on synchronization [9]. At the application layer, however, DAGs loose this property because branches might be processed in different orders across peers, resulting in incompatible states. An interesting approach is to require blocks to represent commutative operations, thus resulting in CmRDTs [9]. CmRDTs have the advantage to store only small updates that are sufficient to reconstruct any version of the data. This contrasts with CvRDTs, which would require to store complete versions. At the third layer, the consensus algorithm transforms a chain DAG into a totally-ordered set, which leads to a CRDT that does not require commutative operations.

Next, we illustrate this three-layered CRDT scheme through an example of a simple permissionless distributed VCS (DVCS) implemented on top of public chains.

2. Chat: https://archive.org/download/WikimediaIrcLogs/
3. Newsgroup: https://archive.org/download/usenet-comp

4.1 A DVCS with Automatic Merges

We built a simple DVCS with the functionalities (and associated Freechains operations) as follows:

- initialize a repository (`join`)
- commit local changes to repository (`post`)
- checkout repository changes to local (`consensus/get`)
- synchronize with remote peer (`send/recv`)
- rate and revoke commits (`like/dislike`)

The system relies on public forum chains, which means that repositories are permissionless and adhere to the reputation and consensus mechanism. The main innovations in this system are that (i) users can rate commits to reject them, and (ii) checkout operations merge commits automatically based on consensus order.

As a concrete example, we model a Wiki article as a chain behaving as a VCS holding its full edition history. We assume that and article could refer to other articles using hyperlinks to other chains to model a complete Wiki platform.

Most VCS operations, except `commit` and `checkout`, map directly to single Freechains commands. For instance, to create a repository, we simply join a public chain with the name of the file we want to track. In the commands that follow, we create a repository with multiple pioneers, and then edit and commit the file twice:

```
> freechains #p2p.md join A2885F.. 2B9C32..
> echo "P2p networking is..." > p2p.md
> freechains-vcs #p2p.md commit --sign=699299..
1_4F3EE1..
> echo "The [USENET](#usenet.md), ..." >> p2p.md
> freechains-vcs #p2p.md commit --sign=699299..
2_B58D22..
```

The `commit` operation expands as follows:

```
> freechains-vcs #p2p.md checkout p2p.remote
> diff p2p.remote p2p.md > p2p.patch
> freechains #p2p.md post p2p.patch --sign=699299..
```

A commit first makes a checkout from the repository into temporary file `p2p.remote`. Then, it compares this version against our local changes, saving the diffs into file `p2p.patch`. A patch contains the minimal set of changes to apply back into the repository and represents a CmRDT operation in our model. Finally, the commit posts the patch file back into the chain. Evidently, the chain name and signature (`#p2p.md` and `--sign=...`) are parameters in the actual implementation of `commit`. The checkout operation is expanded further.

Much time later, the other pioneer synchronizes with us, checks out the file, and then edits and commits it back:

```
> freechains #p2p.md recv '<our-ip>'
2/2    <-- two commits above
> freechains-vcs #p2p.md checkout p2p.md
> echo "P2P does not scale!" >> p2p.md
> freechains-vcs #p2p.md commit --sign=320B59..
3_AE3A1B..
```

A `checkout` recreates the latest version of the file in the repository by applying all patches since the genesis block. Recall that each patch represents a CmRDT operation to recreate the final state of the data. The `checkout` expands as follows:

```
> rm p2p.md && touch p2p.md
> for blk in `freechains #p2p.md consensus` do
    freechains #p2p.md get payload $blk > p2p.patch
    patch p2p.md p2p.patch
    if [ $? != 0 ]; then
      echo $blk  <-- hash of failing patch
      break      <-- ignore remaining patches
    fi
  done
```

The `consensus` operation of Freechains returns all hashes since the genesis block, respecting the consensus order. The loop reads each of the payloads representing the patches and apply them in order to recreate the file. If any of the patches fail, the command exhibits the hash of the offending block and terminates. We discuss this behavior further, when we illustrate commit conflicts.

Since the last commit above is clearly wrong (P2P networks do scale!), other users in the network will dislike it until the block becomes `REVOKED` in the chain (as described before in Figure 4):

```
> freechains #p2p.md dislike 3_AE3A1B.. --sign=USR1
> freechains #p2p.md dislike 3_AE3A1B.. --sign=USR2
> freechains #p2p.md dislike 3_AE3A1B.. --sign=USR3
> freechains-vcs #p2p.md checkout p2p.md
> cat p2p.md
P2p networking is...
The [USENET](#usenet.md), ...  <-- no 3rd line
```

This way, the checkout operation above will apply an empty patch associated with the revoked block, removing the wrong line from the file. This mechanism illustrates how the reputation system enables collaborative permissionless edition and curation.

Next, we create a conflicting situation in which two authors edit and commit the same line of the file concurrently:

```
# PEER A (more reputation):
> sed -i 's/P2p/P2P/g' p2p.md    <-- fix typo
> freechains-vcs #p2p.md commit --sign=699299..
4_A..

# PEER B (less reputation):
> sed -i 's/networking/computing/g' p2p.md
> freechains-vcs #p2p.md commit --sign=320B59..
4_B..

# SYNCHRONIZE (exchange conflicting forks):
> freechains #p2p.md recv '<our-ip>'
1 / 1
> freechains #p2p.md send '<our-ip>'
1 / 1
> freechains-vcs #p2p.md checkout p2p.md
1 hunk FAILED -- saving rejects to file p2p.md.rej
4_B..
> cat p2p.md
P2P networking is... <-- typo fixed (not computing)
The [USENET](#usenet.md), ...
```

After they commit the conflicting changes, the peers synchronize in both directions and reach the state of Figure 5. When we checkout the file, the patches are applied respecting the consensus order. As a result, we see that the first branch is applied, but not the second, leaving the file in the longest possible consistent state. This happens because of the `break` in the checkout operation above: once a conflict is found, no further patches apply in any of the remaining branches. We chose to adopt a *first write wins* resolution to favor work in the branches with more reputation. Nevertheless, the failing patch branch is not totally ignored, since the checkout saves the conflict file and indicates the

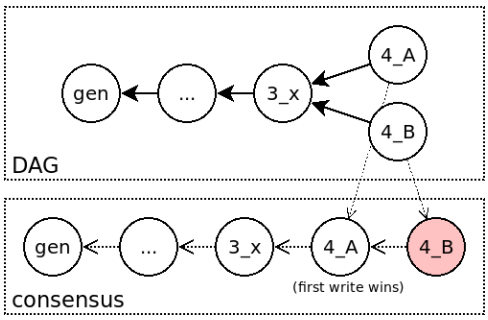

Fig. 5. The branches in the DAG are ordered by reputation. Only the first patch is applied successfully (first write wins).

block causing it. We believe this optimistic choice that does not reject both patches is the most advantageous, since it keeps the file in an usable state and warns about the conflict to resolve. For instance, the authors can later decide to dislike one of the two commits to revoke it and remove the warning.

4.2 Discussion

In summary, the proposed reputation and consensus mechanism empowers a simple DVCS with cooperative authoring and automatic conflict resolution. It only requires the standard *diff & patch* tools and the basic API of Freechains.

We apply the proposed three-layered CRDT scheme as follows: The first layer transports the whole commit history of small patches as a CvRDT between peers, which eventually reach the same DAG state. At this layer, the DAG is just raw data with no attached semantics. In the second layer, peers need to interpret the DAG as a CmRDT of small editions to recreate the file. The *patch* tool is mostly commutative, except when branches modify the same lines. Hence, in these situations, we resort to the third layer with the consensus order, which we apply sequentially until a patch conflict occurs. The final state of the file is guaranteed to be consistent, i.e., the result of a sequence of correct patch applications.

Applications that rely only on commutative operations can traverse the DAG in any causal order, possibly in parallel. This is the case of most social apps with threaded conversations, in which branches typically do not interfere with each other. For instance, it is not problematic to rely on block timestamps to display messages in chats, forums, and social media posts. This is also the general case for notifications in these applications, such as status updates and social engagements.

Towards richer distributed collaborative applications, we can employ CRDTs to model data other than raw text. As an example, *Automerge* [14] manipulates JSON objects, which provides non-trivial datasets with robust merging policies.

5 Related Work

Many other systems have been proposed for distributed content dissemination [3], [5]. Here we consider the classes of publish-subscribe protocols, federated applications, and fully peer-to-peer systems.

5.1 Publish-Subscribe Protocols

Decentralized topic-based publish-subscribe protocols, such as *XMPP* [15], *ActivityPub* [16], and *gossipsub* [17], decouples publishers from subscribers in the network. A key limitation of *pubsubs* is that the brokers that mediate communication still have a special role in the network, such as authenticating and validating posts. Nevertheless, some pubsubs do not rely on server roles, and instead, use peer-to-peer gossip dissemination [18], [19], [20], [21], [17], [19]. Most of these protocols focus on techniques to achieve scalability and performance, such as throughput, load balancing, and real-time relaying.

However, these techniques alone are not sufficient to operate permissionless networks with malicious Sybils [22]. Being generic protocols, pubsubs are typically unaware of the applications built on top of them. In contrast, as stated in Section 2, the pubsub of Freechains is conceptually at the application level and is integrated with the semantics of chains, which already verifies blocks at publishing time. For instance, to flood the network with posts, malicious peers need to spend reputation, which takes hours to recharge (rule 2 in Table 3). In addition, blocked posts (Figure 4) are not a concern either, because they have limited reachability. Another advantage of a tighter integration between the application and protocol is that Merkle DAGs simplify synchronization, provide persistence, and prevent duplication of messages. Full persistence resists long churn periods, and de-duplication tolerates CmRDTs with operations that are not idempotent.

In summary, Freechains and *pubsub* middlewares operate at different network layers, which suggests that Freechains could benefit of the latter to manage the interconnections between peers.

5.2 Federated Applications

Federated applications, such as e-mail, allow users from one domain to exchange messages with users of other domains seamlessly. *Diaspora*, *Matrix*, and *Mastodon* are more recent federations for social media, chat, and microblogging [5], respectively.

As a drawback, identities in federations are not portable across domains, which may become a problem when servers shutdown or users become unsatisfied with the service. In any of these cases, users have to grab their content, move to another server, and announce a new identity to followers.

Moderation is also a major concern in federations [5]. As an example, messages crossing domain boundaries may be subject to different policies that might affect delivery. With no coordinated consensus, it is difficult to make pervasive public forums practical. For this reason, Matrix supports a permissioned moderation system[4], but which applies only within clients, after the messages have already been flooded in the network.

As a counterpoint, federated protocols seem to be more appropriate for real-time applications such as large chats rooms. The number of hops and header overhead can be much smaller in client-server architectures compared to peer-to-peer systems, which typically include message signing, hash linking, and extra verification rules.

4. Matrix moderation: https://matrix.org/docs/guides/moderation

5.3 Peer-to-Peer Systems

Bitcoin [7] is probably the most successful permissionless network but serves specifically for electronic cash. IPFS [10] and Dat [23] are data-centric systems for hosting large files and applications, respectively. Scuttlebutt [6] and Aether [5] are closer to Freechains goals and cover human-centric $1{\rightarrow}N$ and $N{\leftrightarrow}N$ public communication, respectively.

Bitcoin adopts CPU proof-of-work to achieve consensus, which does not solve the centralization issue entirely, given the high costs of equipment and energy. Proof-of-stake is a prominent alternative [24] that acknowledges that centralization is inevitable (i.e., the richer gets richer), and thus uses a function of time and wealth to elect peers to mint new blocks. As an advantage, these proof mechanisms are generic and apply to multiple domains, since they depend on an extrinsic scarce resource. In contrast, we chose an intrinsic resource, which is authored content in the chains themselves. We believe that human work grows linearly with effort and is not directly portable across chains with different topics. These hypotheses support the intended decentralization of our system. Another distinction is that generic public ledgers require permanent connectivity to avoid forks, which opposes our local-first principle. This is because a token transaction only has value as part of the longest chain. This is not the case for a local message exchange between friends, which has value in itself.

IPFS [10] is centered around immutable content-addressed data, while Dat [23] around mutable pubkey-addressed data. IPFS is more suitable to share large and stable content such as movies and archives, while Dat is more suitable for dynamic content such as web apps. Both IPFS and Dat use DHTs as their underlying architectures, which are optimal to serve large and popular content, but not for search and discovery. In both cases, users need to know in advance what they want, such as the exact link to a movie or a particular identity in the network. On the one hand, DHTs are probably not the best architecture to model decentralized human communication with continuous feed updates. On the other hand, replicating large files across the network in Merkle DAGs is also impractical. An alternative is to use DHT links in Merkle payloads to benefit from both architectures.

Scuttlebutt [6] is designed around public identities that follow each other to form a graph of connections. This graph is replicated in the network topology as well as in data storage. For instance, if identity A follows identity B, it means that the computer of A connects to B's in a few hops and also that it stores all of his posts locally. This architecture is similar to $1{\rightarrow}N$ public identity chains of Freechains in Table 1. For group $N{\leftrightarrow}N$ communication, Scuttlebutt uses the concept of channels, which are in fact nothing more than hash tags (e.g. *#sports*). Authors can tag posts, which appear not only in their feeds but also in local virtual feeds representing these channels. However, users only see channel posts from authors they already follow. In practice, channels simply merge friends posts and filter them by tags. In theory, to read all posts of a channel, a user would need to follow all users in the network (which also implies storing their feeds). A limitation of this model is that new users struggle to integrate in channel communities because their posts have no visibility at all. As a counterpoint, channels are safe places that do not suffer from abuse. For group $N{\leftrightarrow}N$ communication, Freechains relies on the reputation system to prevent abuse, since new users require at least one like for visibility in the community. Also, a forum chain stores its own posts only, and are not simply tags over the entire content in the network.

Aether [5] provides peer-to-peer public communities aligned with $N{\leftrightarrow}N$ public forums of Freechains. A fundamental difference is that Aether is designed for ephemeral, mutable posts with no intention to enforce global consensus across peers. Aether employs a very pragmatic approach to mitigate abuse in forums. It uses established techniques, such as proof-of-work to combat SPAM, and an innovative voting system to moderate forums, but which affects local instances only. In contrast, Freechains relies on its permissionless reputation and consensus mechanisms for moderation.

6 Conclusion

We propose a new permissionless consensus mechanism for content dissemination in peer-to-peer networks. We enumerate four main contributions: (i) human authored content as a scarce resource (*proof-of-authoring*); (ii) diversified public forums, each as an independent blockchain with subjective moderation rules; (iii) removal of abusive content while preserving data integrity; and (iv) a three-layered CRDT scheme to build collaborative applications.

The main insight of the mechanism is to use the human authoring ability as a scarce resource to determine consensus. This contrasts with extrinsic resources, such as CPU power, which are dispendious and not evenly distributed among people. Consensus is backed by a reputation system in which users can rate posts with likes and dislikes, which transfer reputation between them. The only way to forge reputation is by authoring new content under the judgement of other users. This way, reputation generation is expensive, while verification is cheap and distributed.

The reputation and consensus mechanism is integrated into Freechains, a peer-to-peer protocol that offers multiple arrangements of public and private communications. Users have the power to create their own public forums of interest and apply diverse moderation policies that better suit their needs. In particular, users can revoke content considered abusive according to their own policies, not depending on centralized authorities.

On top of the proposed three-layered CRDT architecture, we prototyped a distributed version control system that resolves commit conflicts automatically: (i) at the transport layer, the full commit history is held in a Merkle DAG counting as a CvRDT; (ii) at the application layer, the commit patches operate as (mostly) commutative CmRDT operations; and (iii) after consensus is applied, the merge conflicts are automatically resolved.

TODO Finally, we do not claim that the proposed reputation system enforces "good" human behavior in any way. Instead, it provides a transparent and quantitative mechanism to help users understand the evolution of forums and act accordingly.

REFERENCES

[1] J. Zittrain, "Fixing the internet," vol. 362, no. 6417. American Association for the Advancement of Science, 2018, pp. 871–871.

[2] N. Masinde and K. Graffi, "Peer-to-peer-based social networks: A comprehensive survey," *SN Computer Science*, vol. 1, no. 5, pp. 1–51, 2020.

[3] S. A. Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Comput. Surv.*, Dec. 2004.

[4] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 5, no. 1, pp. 63–108, 1998.

[5] J. Graber, "Decentralized social ecosystem review," BlueSky, Tech. Rep., 2021.

[6] D. Tarr *et al.*, "Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications," in *Proceedings of ACM ICN'19*, 2019, pp. 1–11.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Tech. Rep., 2019.

[8] F. Sant'Anna, F. Bosisio, and L. Pires, "Freechains: Disseminação de conteúdo peer-to-peer," in *Workshop on Tools, SBSeg'20*.

[9] H. Sanjuan, S. Poyhtari, P. Teixeira, and I. Psaras, "Merkle-crdts: Merkle-dags meet crdts," *arXiv preprint arXiv:2004.00107*, 2020.

[10] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[11] M. Kleppmann, A. Wiggins, P. van Hardenberg, and M. McGranaghan, "Local-first software: you own your data, in spite of the cloud," in *Proceedings of Onward'19*, 2019, pp. 154–178.

[12] R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M. Henze, and K. Wehrle, "How to securely prune bitcoin's blockchain," in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 298–306.

[13] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Symposium on Self-Stabilizing Systems*. Springer, 2011, pp. 386–400.

[14] M. Kleppmann and A. R. Beresford, "A conflict-free replicated json datatype," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2733–2746, 2017.

[15] P. Saint-Andre, K. Smith, R. Tronçon, and R. Troncon, *XMPP: the definitive guide*. " O'Reilly Media, Inc.", 2009.

[16] C. Webber, J. Tallon, and O. Shepherd, "Activitypub," *W3C Recommendation, W3C, Jan*, 2018.

[17] D. Vyzovitis and Y. Psaras, "Gossipsub: A secure pubsub protocol for unstructured, decentralised p2p overlays," Protocol Labs, Tech. Rep., 2019.

[18] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni, "TERA: Topic-Based Event Routing for Peer-to-Peer Architectures," in *Proceedings of the International Conference on Distributed Event-Based Systems*, 2007, pp. 2–13.

[19] J. A. Patel, É. Rivière, I. Gupta, and A.-M. Kermarrec, "Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems," *Computer Networks*, vol. 53, no. 13, pp. 2304–2320, 2009.

[20] M. Matos, A. Nunes, R. Oliveira, and J. Pereira, "Stan: exploiting shared interests without disclosing them in gossip-based publish/subscribe." in *IPTPS*, 2010, p. 9.

[21] F. Rahimian, S. Girdzijauskas, A. H. Payberah, and S. Haridi, "Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks," in *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2011, pp. 746–757.

[22] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, "Gossipsub: Attack-resilient message propagation in the filecoin and eth2. 0 networks," Protocol Labs, Tech. Rep., 2020.

[23] D. C. Robinson, J. A. Hand, M. B. Madsen, and K. R. McKelvey, "The dat project, an open and decentralized research data tool," *Scientific data*, vol. 5, no. 1, pp. 1–4, 2018.

[24] L. M. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2018, pp. 1545–1550.

Francisco Sant'Anna received his PhD degree in Computer Science from PUC-Rio, Brazil in 2013. In 2016, he joined the Faculty of Computer Science at the Rio de Janeiro State University, Brazil. His research interests include Programming Languages and Concurrent & Distributed Systems.