Peer-to-Peer Permissionless Consensus via Reputation

Francisco Sant'Anna, Fabio Bosisio, Lucas Pires

[1] UERJ - PEL - Pós-Graduação em Engenharia Eletrônica

francisco@ime.uerj.br, fbosisio@gmail.com, lucasampires@gmail.com

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 content sharing. 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 as follows: consolidated 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.*

1. Introduction

Content publishing in Internet forums and social media is increasingly more centralized in a few companies (e.g. Facebook and Twitter) [23]. On the one hand, these companies offer free storage, friendly user interfaces, and robust access. On the other hand, they concentrate excessive power, controlling and monetizing over private user data. Peer-to-peer alternatives [19] eliminate intermediaries, but strive to achieve consistency when dealing with malicious users.

In an ideal Internet forum, all messages or posts (i) reach all users, (ii) are delivered in consistent order, and (iii) are respectful and on topic. In centralized systems, items (i) and (ii) are trivially achieved assuming availability and delivery order, while item (iii) assumes that users trust the service to moderate content. In decentralized systems, however, none of these demands are easily accomplished. A common approach in gossiping protocols is to proactively disseminate posts among peers until they reach all users [19, 8]. However, this approach does not guarantee consensus since posts can be received in conflicting orders [17]. Consensus is key to distinguish malicious behavior *at the protocol level*, and therefore, satisfy item (iii). Without consensus, each client must rely on local anti-abuse policies, such as SPAM filters and blacklists, but which only apply a posteriori, when the abusive messages have already been flooded in the network.

Bitcoin [12] is the first permissionless protocol to resist Sybil attacks [5]. Its key insight is to rely on a scarce resource—the *proof-of-work*—to establish consensus. The protocol is Sybil resistant because it is expensive to write to its unique timeline (either via proof-of-work or transaction fees). However, Bitcoin and cryptocurrencies in general are not suitable for content sharing because (i) they enforce a unique timeline to preserve value and immunity to attacks; (ii) they lean towards concentration of power due to scaling effects; (iii) they impose an external economic cost to use the protocol; and (iv) they rely exclusively on objective rules to reach consensus. These issues threaten our original decentralization goals, and more importantly, they renounce subjectivity, which is inherent to content moderation. For instance, a unique timeline implies that all Internet content should be subject to the same consensus rules, while objective rules alone cannot distinguish abuse amid content. Another limitation of cryptocurrencies is that it is not possible to revoke content in the middle of a blockchain, which is inadmissible considering illicit content (e.g., hate speech) [11].

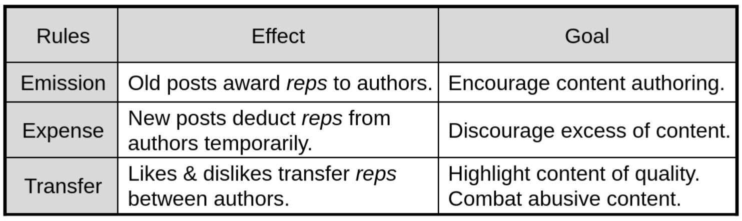
In this work, we adapt the idea of scarce resources to reach consensus and eradicate Sybils, but in the context of social content sharing. Our first contribution is to recognize the actual published contents as the scarce resources themselves, since they require human work. Work is manifested as new posts, which if approved by others, reward authors with reputation tokens, which are further used to evaluate other posts with likes and dislikes. With such *proof-of-authoring* mechanism, token generation is expensive, while verification is cheap and shared by multiple users. Due to decentralization and network partitions, posts in a timeline 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 reputation (akin to Bitcoin's double spending). Our second contribution is to allow users to create diversified forums of interest (instead of a singleton blockchain), each counting as an independent timeline with its own subjective consensus etiquette. Our third contribution is to support content removal without compromising the integrity of the decentralized blockchain. Users have the power to revoke posts with dislikes, and peers are forced to remove payloads, only forwarding associated blockchain metadata. We integrated the proposed consensus algorithm into Freechains [16], a practical peer-to-peer (P2P) content dissemination protocol that provides strong eventual consistency [6]. To show the practicability of the consensus mechanism, we simulated months of activity of a chat channel and years of a newsgroup forum, both extracted from publicly available Internet archives.

The rest of the paper is organized as follows: In Section 2, we describe the design of the reputation and consensus mechanism for public forums and discuss some security threats. In Section 3, we describe the concrete reputation rules we implemented for Freechains and evaluate the performance of the protocol in real-world public forums. In Section 4, we compare our system with publish-subscribe protocols, federated applications, and fully P2P systems. In Section 5, we conclude this work.

2. The Reputation and Consensus Mechanism

In the absence of moderation, permissionless P2P forums are impractical, mostly because of Sybils abusing the system. For instance, it should take a few seconds to generate thousands of fake identities and SPAM millions of messages. For this reason, we propose a reputation system that works together with a consensus algorithm to resist Sybil attacks.

In our system, users can spend tokens named *reps* to post and rate content in forums: a `post` initially penalizes authors until it consolidates and counts positively; a `like` is a positive feedback that helps subscribers to distinguish content amid excess; a `dislike` is a negative feedback that revokes content when crossing a threshold. Table 1 summarizes the general reputation rules and their goals. To prevent Sybils, users with no *reps* cannot operate under these rules, requiring a welcoming like from any other user already in the system. The fact that likes are zero-sum operations, which only transfer



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

Table 1. General reputation rules in public forums.

reputation, eliminates the incentives from malicious users to invite Sybils into the system. The only way to generate new *reps* is to post content that other users tolerate, which demands non-trivial work resistant to automation.[1] In this sense, having one or hunderds of machines in the network does not affect the capacity to write into the chains.

Bitcoin employs proof-of-work to mitigate Sybil attacks. However, CPU or other extrinsic resources are not evenly distributed among humans, specially in communications using battery-powered devices. Instead, considering the context of public forums, we propose to take advantage of the human authoring ability as an intrinsic resource. 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 only transfer *reps* between users. In our analogy with Bitcoin, Sybil attacks would require to add human resurces, instead of CPU power.

Nevertheless, posts scarcity is not yet sufficient to combat Sybils because consensual order is still required to prevent inconsistent operations. For instance, consider a malicious author with a single unit of *reps* posting SPAM messages from multiple peers at the same time. According to the *Expense* rule of Table 1, only one of these messages should be accepted by the network. However, without consensus, it is not possible to globally determine which message to accept, since each peer would supposedly accept the first message it receives. On the one hand, in order to validate operations consistently, we need the same message ordering across all peers in the network. On the other hand, due to network partitions, we can only represent public forums as DAGs (directed acyclic graphs) with causal relationships between messages, which provides at most partial order.

2.1. Basic Reputation Consensus

Our key idea to stablish consensus in forum DAGs is to favor forks with posts from users that constitute the majority of the reputation. These forks have more associated work and are analogous to longest chains in Bitcoin. In technical terms, we adapt a topological sorting algorithm to favor reputation when deciding between branches in a forum DAG.

Figure 1.A illustrates the evolution of a forum DAG in the presence of a network partition: A common prefix has signed posts from users a, b, and c, when they were still all connected. We assume that within the prefix, users a and b have contributed with better content and therefore have more reputation combined than c has alone (i.e., $8{+}5 > 3$). We will use this fact to order branches in the consensus algorithm. Then, user c disconnects from a and b, and evolves with new users x and y in `branch-1`. In the meantime, a and

[1] Note that although human and AI content may become indistinguishable, their frequency and relevance in forums is still subject to human evaluation in our proposal.

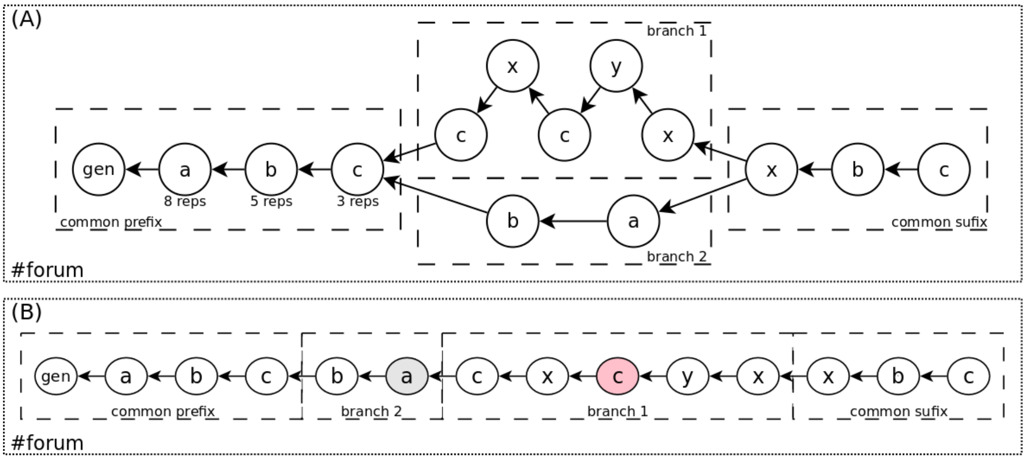


Figure 1. (A) A public forum DAG with a common prefix, two branches, and a common sufix. (B) Total order between posts of the DAG after consensus.

b participate with new posts in `branch-2`. After some time, the partitions reconnect and new posts from x, b and c merge the branches now with a common suffix.

Note that even non-malicious network partitions may experience conflicts when merging branches. For instance, dislike operations in one branch to past posts in the common prefix could possibly block posts from users in the other branch, depending on how they posts are ordered. Therefore, we need to order branches with a global consensus, such that we can validate operations consistently across the network. Note also that, to reach consensus, we should preferably rely on the maximum information that all branches agree on, which is exactly the posts in the common prefix.

Based on the reputation of users in the common prefix, Figure 1.B indicates the consensus order between posts in the forum DAG: the common prefix comes first, then all posts in `branch-2`, then all posts in `branch-1`, then the common suffix. The `branch-2` with users a and b takes priority over `branch-1` with user c because, before the forking point, a and b have more reputation than c, x, and y combined.

The basic consensus rule is therefore straightforward: Whenever a fork is found, we create sets of users with posts in each branch ($B1 = \{c, x, y\}$ and $B2 = \{a, b\}$). Then, we take each set, and sum its users *reps* based on the common prefix ($S1 = 3$ and $S2 = 13$), since it is the maximum information that all branches agree. Finally, we order the branches based on the highest sum of *reps* found (Figure 1.B).

While applying the branches in order, if any post operation fails, all remaining posts are rejected and removed from the DAG, as if they never existed. 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 suppose with *0 reps*, is rejected together with all posts in sequence, including those in the common suffix. This rejection rule for remaining posts is a direct consequence of tamper proof Merkle DAGs [4, 12] typically used in permissionless protocols, which cannot support graph modifications.

There are some other relevant considerations about forks and merges: Peers that first receive branches with less reputation need to reorder all posts after the forking point. This might involve removing content in the end-user software. This behavior is similar

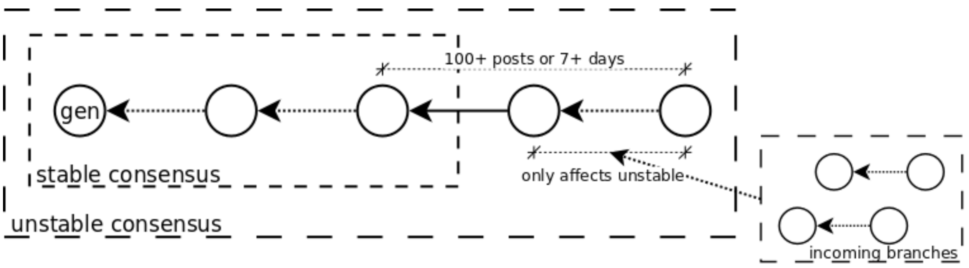


Figure 2. Stable consensus freezes the order of posts once they cross the threshold. The unstable order may still be affected by incoming branches.

to Bitcoin's blockchain reorganization, when peers detect new longest chains. Likewise, peers that first saw branches with more reputation just need to append the other branch, with no reordering at all. This behavior is expected to happen in the majority of the connected network. Legitimate posts in secondary branches might be removed due to merges, which requires that users repost the messages. Nevertheless, we expect that legitimate users gossip among themselves more often, preventing this situation. Note that unlike Bitcoin, forks are not only permitted but encouraged due to the local-first software principle [9], in which networked applications can work locally while offline. However, the longer a peer remains disconnected, the more conflicting operations it may see, and the higher are the chances of posts reordering when rejoining.

In summary, it is important to remark that (i) branches always introduce conflicts, which (ii) require consensus to order operations, which (iii) is based on a common prefix reputation, (iv) leading to a global and deterministic result. Note that the consensus order exists only to account reputation and verify operations, and is only a view of the primary DAG structure of public forums.

2.2. Malicious Behaviors and Hard Forks

Consider a malicious scenario in which user c in Figure 1 intentionally disconnects from the network for weeks to cultivate fake identities x and y with the intention to accumulate *reps*. When merging, these fake users would become legitimate and accumulate the majority of *reps* in the forum. However, this kind of attack is refutable because, since the common prefix is the basis of consensus, users in the branch with more reputation can still react even a posteriori. For instance, users a and b can pretend that they did not yet see `branch-1` and post extra dislikes to user c so that a further merge invalidates all posts from `branch-1`, removing them from the DAG. This kind of attack is innocuous because it requires at least half of the *reps* accumulated in the common prefix. The same countermeasure can be used against a minority of malicious users colluding to boost or protect their own posts.

As a counterpoint, maybe users a and b have abandoned the forum for months, and thus `branch-1` is actually legitimate. In this case, users a and b might be the ones trying to take over the forum and would succeed, since they control the majority of *reps* in the common prefix. Yet another possibility is that both branches are legitimate but became disconnected for a long period of time. In any case, it is unacceptable that a very old remote branch affects a long active forum.

For this reason, the consensus algorithm includes an extra constraint that prevents

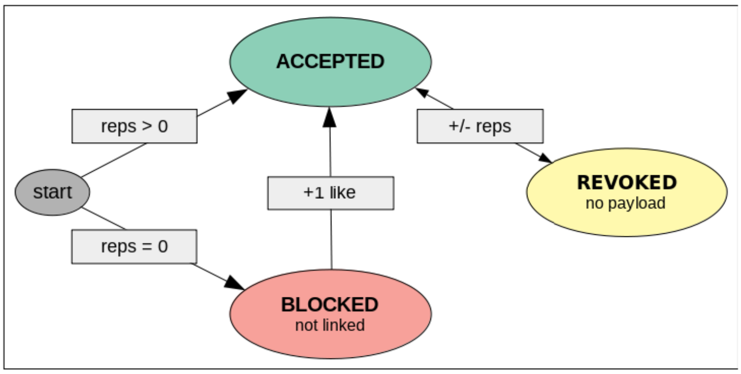


Figure 3. State machine of posts: `BLOCKED` posts are not linked in the DAG. `ACCEPTED` posts are linked and retransmitted. The payload of `REVOKED` posts are not retransmitted.

long-lasting local branches to merge, which leads to *hard forks* in the network. A hard fork occurs when a local branch crosses a predetermined and irreversible threshold of *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 become incompatible and will never synchronize again. More than simple numeric disputes, a hard fork represents a social conflict in which reconciling branches is no longer possible.

Figure 2 illustrates hard forks by distinguishing *stable consensus*, which cannot be reordered, from *unstable consensus*, which may still be affected by incoming branches. The activity threshold counts backwards, starting from the newest local post in the unstable consensus to older posts until they are permanently frozen.

In summary, the rules to merge a branch `j` from a remote machine into a branch `i` from a local machine are as follows:

- `i` is first if it crosses the activity threshold of *7 days* or *100 posts*, regardless of `j`.
- `i` or `j` is first, whichever has more reputation in the common prefix, as discussed in Section 2.1.
- otherwise, branches use the arbitrary criteria of lexicographical order of the post hashes immediately after the common prefix.

To conclude this section, we should keep in mind that our consensus algorithm is based on subjective evaluations of users and posts. For this reason, unlike Bitcoin, in which forks are discouraged and a single blockchain must prevail, public forums can split and diverge indiscriminately. For instance, users can explicitly apply hard forks at any point in time to “reboot” a community from a previous state. Ultimately, the consensus algorithm provides a transparent mechanism to help users understand the evolution of public forums and act accordingly, even if it leads to hard forks.

2.3. Content Removal

Finally, we consider that revoking posts is fundamental in the context of social content publishing, and thus, an important contribution of this work.

As described in Figure 3, a forum post has three possible states: `BLOCKED`, `ACCEPTED`, or `REVOKED`. If the author has reputation, a new post is immediately `ACCEPTED` in the forum. Otherwise, it is BLOCKED and requires a like from another user.

Blocked posts are not considered part of the forum DAG in the sense that new posts do not link back to it. In addition, peers are not required to hold blocked posts nor 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. Therefore, a reasonable policy is to hold blocked posts in a temporary but public bag and retransmit them for some visibility.

Once accepted, a post becomes part of the forum and can never be removed again, since Merkle DAGs are immutable by design. However, if the number of dislikes exceeds an arbitrary threshold (e.g., more dislikes than likes), the post becomes REVOKED and its payload is not retransmitted to other peers. Note that a post hash does not depend on its associated payload, but only on the payload hash. Hence, it is safe to remove the 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, since the post becomes ACCEPTED again.

2.4. Peer Synchronization and Byzantine Faults

Similarly to Bitcoin, the underlying network protocol replicates on each peer the full state of the Merkle-DAG representing the causal relationships between the messages. When peers synchronize, they mutually exchange missing branches such that they reach the same DAG structure [8]. As they synchronize, each peer runs the consensus algorithm locally and verifies if all received posts are consistent.

Because the protocol relies on tamper-proof DAGs, and because each peer self verifies the posts as they synchronize, the protocol becomes tolerant to Byzantine faults [10]. Note that no information coming from any peer is trusted a priori. For instance, a malicious peer that tries to synchronize branches with erratic data can be detected on the first inconsistent post. This allows the correct peer to close the connection immediately and blacklist the malicious peer from future connections.

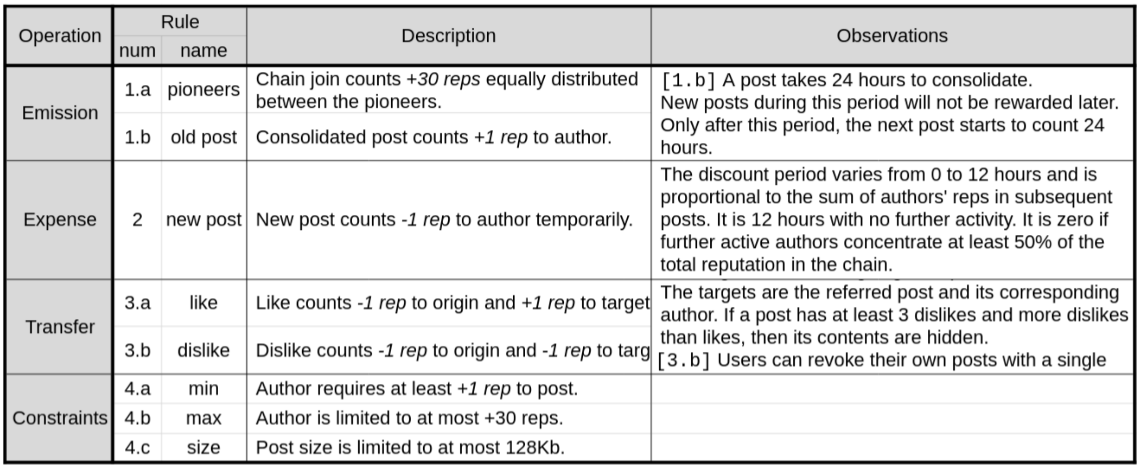
Therefore, although Byzantine peers may still perform some sort of denial of service attacks in the network, they cannot modify the DAG structure, nor create arbitrary content, nor impede that two correct nodes synchronize directly.

3. Public Forums in Freechains

Freechains [16] is an unstructured P2P topic-based publish-subscribe protocol, in which each *chain* is a replicated Merkle-DAG representing the causal relationships between posts. The protocol operation is typical of publish-subscribe systems: an author publishes a post to a chain, which subscribed users eventually receive.

In order to support content moderation and mitigate abuse, we integrated the proposed reputation and consensus mechanism of Section 2 with Freechains. Table 2 details the concrete reputation rules we conceived for Freechains, which is compatible with the general rules of Table 1, and which are discussed as follows through an example. Authors have to sign posts in order to be accounted by the reputation system and operate in the chains. We start by creating an identity whose public key is assigned as the pioneer in a chain named *forum*:

```
 > freechains keys pubpvt 'pioneer-password'
```



Operation	Rule		Description	Observations
	num	name		
Emission	1.a	pioneers	Chain join counts *+30 reps* equally distributed between 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 target	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
	3.b	dislike	Dislike counts *-1 rep* to origin and *-1 rep* to targ	
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 2. Reputation rules for public forum chains in Freechains. The chosen constants ($30\ reps$, $24h$, etc) are arbitrary and target typical Internet forums.

```
4B56AD.. DA3B5F..  <-- public and private keys
> freechains 'forum' join '4B56AD..'
10AE3E..           <-- hash representing the chain
> freechains 'forum' post --sign='DA3B5F..' 'The chain purpose is...'
1_CC2184..         <-- hash representing the post
```

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 (one in this example). 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 and indicates the purpose of the chain for future users.

The most basic concern in public forums is to resist Sybil attacks. Fully P2P 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 explicit 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 an existing user to like a newbie's post to unblock it, but at the cost of *1 rep*. This 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 remote user joins the same public chain and posts a message, which is welcomed with a like signed by the pioneer:

```
> freechains keys pubpvt 'newbie-password'
503AB5.. 41DDF1..  <-- public and private keys
> freechains 'forum' join '4B56AD..'
10AE3E..           <-- same pioneer as before
> freechains 'forum' post 'Im a newbie...' --sign='41DDF1..'
2_C3A40F..         <-- blocked post
```

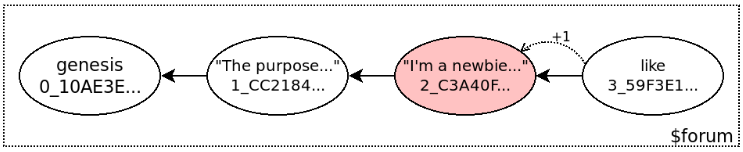


Figure 4. The `like` approves the newbie message into the `#forum` DAG.

```
> freechains 'forum' like '2_C3A40F..' --sign='DA3B5F..'
3_59F3E1..          <-- hash representing the like
```

Note that chains with the same name but different pioneers would be incompatible because the hash of genesis posts depends on the pioneers' public keys.

Figure 4 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 quickly.

Back to Figure 4, a new user with *0 reps* tries to post a message (hash `2_C3A40F..`) and is blocked (rule `4.a`), as the red background highlights. But the pioneer likes 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.

With no additional rules to generate *reps*, the initial *30 reps* would constitute the whole "chain economy" forever. For this reason, rule `1.b` rewards 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 rewarding the author. It also regulates the growth speed of the chain. In Figure 4, 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 new consolidated posts. Note that rule `1.b` rewards at most one post of each author at a time. Hence, extra posts during the 24-hour period do not reward authors 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, while rule `4.c` limits the size of posts to at most *128kB* to prevent DDoS attacks using gigantic blocked posts.

Likes and dislikes (rules `3.a` and `3.b`) serve three purposes: (i) welcoming new users, (ii) measuring the quality of posts, and (iii) revoking abusive posts (SPAM, fake news, etc). The quality of posts is subjective and is up to users to judge them with likes, dislikes, or simply abstaining. This way, access to chains is permissionless in the sense that the actual peers and identities behind posts are not directly assessed by the protocol, but instead by the judgement of other users in the system. The reputation of a given post is the difference between its likes and dislikes, which can be used in end-user software for filtering and highlighting purposes. 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. Finally, 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.

3.1. Experiments with Public Forums in Freechains

We performed experiments to evaluate the performance of Freechains and its consensus algorithm. As detailed further, we measure the following evaluation parameters: (a) metadata overhead, (b) consensus runtime, (c) graph forks, and (d) blocked messages. The main goal of this section is to stress the consensus algorithm to show that permissionless public forums are viable, regardless of a slower performance in comparison to permissioned protocols.

We simulate the behavior of two publicly available forums as if they were using Freechains: a chat channel from the Wikimedia Foundation[2], and the *comp.compilers* newsgroup[3]. Chats and newsgroups represent typical public forums with faster interactions with shorter payloads (chats), and slower interactions with larger payloads (newsgroups). We only simulate the first *10.000* messages of the forums, which represent 3 months of activity in the chat and 9 years in the newsgroup.

The simulation spawns N peers, each joining the same chain with the same arguments. For each message in the original forum, we (i) set the timestamp of all peers to match the original date, (ii) create a pair of keys if the author is new, (iii) post the message from a random peer in N, (iv) like the post if the author has no reputation, and (v) synchronize the chain with M random peers. Since all messages are part of the original archive, we always perform step (iv) to unblock messages from newbies, which we also use to measure the evaluation parameter (d). Step (v) will inevitably create forks in the chain for any `M<N`, which we measure in evaluation parameter (c).

For the newsgroup, we use `N=15` and `M=5`, which represents a larger number of peers with few interconnections to stress the local-first nature of the protocol. For the chat, we use `N=5` and `M=3`, which is a smaller number of peers but with more interconnections. We executed each simulation 4 times in a desktop PC (`i7` CPU, `8GB` RAM, `512GB` SSD). Since the variations were negligible, we always discuss the median measures.

Evaluation item (a) measures the protocol overhead due to blockchain metadata, which consists of a timestamp, author signature, and hashes (post id, payload, backlinks, and likes). The original chat archive is `800kB` in size, or `80B` for each message, which includes a timestamp, a username, and the actual payload. The simulated chat chain is `8MB` in size, which indicates a $10x$ overhead. The original newsgroup is `30MB` in size, or `3kB` for each message, which includes a timestamp, a sender, a subject, and the actual payload (typically much longer). The simulated newsgroup chain is `42MB` in size, which indicates a 50% overhead. It is clear that the metatada overhead is not negligible, specially for short chat messages, but decreases as the payload increases.

[2] Chat: `https://archive.org/download/WikimediaIrcLogs/`

[3] Newsgroup: `https://archive.org/download/usenet-comp`

Evaluation item (b) accounts the consensus algorithm applied locally at each chain. We measured the time to sort posts in a local DAG both for the first time (without any caches), and incrementally (from stable consensus caches). For the chat chain, it takes `125s` and `50ms` for the initial and incremental sorts respectively, while for the newsgroup chain, it takes `100s` and `70ms`. The incremental sort is limited to 7 days or 100 posts by design (as discussed in Section 2.2), regardless of the size of the chain, which conveniently settles an upper bound on the input size of the consensus algorithm. Given that we use plain JSON files in the file system, we consider an incremental consensus under `100ms` to be a practical upper bound.

Evaluation item (c) counts the number of forks in chain DAGs, which indicates the level of asynchrony between peers following the local-first principle. We calculate the ratio of forks over the total number of messages, e.g., a DAG with 100 messages and 10 forks has a ratio of 10%. We found a ratio of 18% for the chat and 14% for the newsgroup, which confirms that the simulation achieves a reasonable level of asynchrony.

Evaluation item (d) measures how much bookkeeping is required to sustain active users in the forums. Even though the newbie rule `4.a` in Table 2 is key to combat Sybils, ideally it should not recurrently deny access to active users with low reputation. The evaluation counts the number of blocked messages requiring extra likes after the welcoming likes (which are disconsidered) and calculates the ratio over the total number of messages in the chain. As an example, if 10 users posted 110 messages requiring 20 likes, we discount the 10 initial messages and welcoming likes (when the user first appears), and find a ratio of 10% (`(20-10)/(110-10)`). For the chat with 80 users, we found a ratio of 3.7%. For the newsgroup with 5000 users, we found a ratio of 3.5%. Considering that users were not aware of the reputation rules, the low ratios indicate that their "natural" posting behavior matches the constraints of the rules. We assume that users would use the revoke mechanism to combat abusive content, having no further effects on our evaluation.

4. Related Work

Many other systems have been proposed for decentralized content sharing [19, 7]. Here we consider the classes of publish-subscribe, federated, and P2P protocols.

Decentralized topic-based publish-subscribe protocols, such as *XMPP* [15], *ActivityPub* [22], and *gossipsub* [21], 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 P2P gossip dissemination [3, 13, 21]. 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 [20]. Being generic protocols, pubsubs are typically unaware of the applications built on top of them. In contrast, as stated in Section 3, the pubsub of Freechains is conceptually at the application level and is integrated with the semantics of chains, which already verifies posts 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 2). In addition, blocked posts (Figure 3) 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. 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.

Federated protocols, such as e-mail, allow users from one domain to exchange messages with users of other domains. *Diaspora*, *Matrix*, and *Mastodon* are recent federations for social media, chat, and microblogging [7], 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 [1]. 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 [7]. 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 P2P systems, which typically include message signing, hash linking, and extra verification rules (as evaluated in Section 3.1).

Regarding P2P protocols, Bitcoin [12] is probably the most successful permissionless network, but serves specifically for electronic cash. IPFS [4] and Dat [14] are data-centric protocols for hosting large files and applications, respectively. Scuttlebutt [18] and Aether [7] are closer to Freechains goals and cover human communication.

Bitcoin adopts 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 [2] that acknowledges that centralization is inevitable, 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 extrinsic resources. In contrast, we chose an intrinsic resource, which is authored content in the chains themselves. We believe that human work grows more 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 [4] is centered around immutable content-addressed data, while Dat [14] around mutable pubkey-addressed data. IPFS is more suitable to share large and stable content such as movies, 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

[4] Matrix moderation: `https://matrix.org/docs/guides/moderation`

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 [18] 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. Note however that the basic unit of communication in Scuttlebutt is unidirectional , with a public identity broadcasting a feed to its followers ($1 \to N$). For open groups communication ($N \leftrightarrow N$), 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.

Aether [7] provides P2P public communities aligned with public forums of Freechains ($N \leftrightarrow N$). 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.

5. Conclusion

In this paper, we propose a permissionless consensus and reputation mechanism for social content sharing in P2P networks. We enumerate three 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; and (iii) abusive content removal preserving data integrity.

The key insight of the consensus 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 decentralized.

The reputation and consensus mechanism is integrated into Freechains, a P2P protocol, in which users can create public forums of interest and apply diverse moderation policies. In particular, users can revoke content considered abusive according to the majority, not depending on centralized authorities. We simulate the behavior of existing chat and newsgroup forums as if they were using Freechains to show the practicability of the protocol as a descentralized alternative for public forums.

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. Human creativity contrasts with plain economic resources (e.g., proof-of-work), which do not appraise social interactions and also tend to concentrate power over the time.

References

[1] A. Auvolat. Making federated networks more distributed. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, pages 383–3831. IEEE, 2019.

[2] L. M. Bach et al. Comparative analysis of blockchain consensus algorithms. In *MIPRO'18*, pages 1545–1550. IEEE, 2018.

[3] R. Baldoni et al. TERA: Topic-Based Event Routing for Peer-to-Peer Architectures. In *DEBS'07*, pages 2–13, 2007.

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

[5] J. R. Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[6] V. Gomes et al. Verifying strong eventual consistency in distributed systems. volume 1, pages 1–28. ACM New York, NY, USA, 2017.

[7] J. Graber. Decentralized social ecosystem review. Technical report, BlueSky, 2021.

[8] M. Kleppmann. Making crdts byzantine fault tolerant. In *Proceedings of the 9th Workshop on Principles and Practice of Consistency for Distributed Data*, pages 8–15, 2022.

[9] M. Kleppmann et al. Local-first software: you own your data, in spite of the cloud. In *Onward'19*, pages 154–178, 2019.

[10] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. In *Concurrency: the works of leslie lamport*, pages 203–226. 2019.

[11] R. Matzutt et al. A quantitative analysis of the impact of arbitrary blockchain content on bitcoin. In *FC'18, Nieuwpoort, Curaçao*, pages 420–438. Springer, 2018.

[12] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, 2009.

[13] J. A. Patel et al. Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems. *Computer Networks*, 53(13):2304–2320, 2009.

[14] 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*, 5(1):1–4, 2018.

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

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

[17] C. Sun et al. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM TOCHI'98*, 5(1):63–108.

[18] D. Tarr et al. Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications. In *ACM ICN'19*, pages 1–11, 2019.

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

[20] D. Vyzovitis et al. Gossipsub: Attack-resilient message propagation in the filecoin and eth2. 0 networks. Technical report, Protocol Labs, 2020.

[21] D. Vyzovitis and Y. Psaras. Gossipsub: A secure pubsub protocol for unstructured, decentralised p2p overlays. Technical report, Protocol Labs, 2019.

[22] C. Webber et al. Activitypub. *W3C Recommendation*, 2018.

[23] J. Zittrain. Fixing the internet. volume 362, pages 871–871. American Association for the Advancement of Science, 2018.