Semaphore:
A Decentralized Social Layer for the Internet

Alex Elder Dulisse
alex.dulisse@gatech.edu

Oscar Thomas Aguilar
oaguilar3@gatech.edu

Craig Aaron Tovey
cat@gatech.edu

July 10, 2022

Abstract

We introduce a new framework for analyzing communication networks, such as Twitter or Bitcoin, and introduce a new classification of network within that framework that is suitable for decentralized social communication. Several components, some new and some old, combine to create a decentralized, credibly neutral, communication network that supports composable social apps to be built on top. The network reaches consensus in three phases. First, broadcasts within a time epoch are gossiped around the network. Second, nodes negotiate with peers on which broadcasts are included in the time epoch. Third, nodes finalize a block of broadcasts representing a previous time epoch. Nodes synchronize with the network via measuring user engagement (proof of engagement), rather than proof of work or proof of stake. The network enables users to run their own network node or securely use a third party without sacrificing autonomy. Users interact with the network through different sets of signing keys, which allows day-to-day signing keys to be separate from keys controlling ownership. Since apps built on top of the network are interoperable, network effects accrue to the network itself, rather than a specific app. Composable apps allow for vital experimentation to improve the individual user experience and social impact of social media.

0 Preface

There are many good reasons why social media has not been "put on the blockchain", or at least not successfully. Nonetheless, social media being an entirely centralized industry causes many problems for individual users and society at large. Existing blockchains are built for financial applications and are not fit for the purpose of online communication. As will be shown in this paper, a blockchain with a fundamentally different design can enable such applications, including decentralizing social media, and fixing many of its problems:

The corporations that run social media platforms have many different roles and the forces acting on each role can negatively impact the others. Twitter is a platform, a protocol, a search engine, a user interface, a website, and, of course, a company. There is no fundamental reason for why all of these different roles need to be controlled by the same group of people. Twitter the corporation is under pressure from various governments and market actors from across the world, with each pushing the company in a different direction. These forces clearly affect all users on the platform. Since every aspect of Twitter is inextricably mixed, nothing can be customized to the preference of an individual user or to a specific application. Users of course do not have the ability to run a twitter client from their own machine. As long as the various roles of Twitter cannot be disambiguated, innovation is stifled and pressures on one part of the Twitter bureaucracy will spill over into all areas of development.

Centralized platforms must have a monolithic moderation policy due to the limitation of a centralized moderation team, even though one set of rules will not satisfy all users. Consequently, there is little room for experimentation in innovative moderation strategies that could be more effective or strike a better balance. If a policy is implemented, it is impossible to know what would have happened in the counterfactual. Good moderation is essential to enabling quality online discussion, but platforms lurch from one lackluster policy to another, leaving online conversation everywhere worse off. Even worse, banned users do not simply disappear. The user can trivially avoid a block or ban by creating a new account and returning with a chip on their shoulder. Yet somehow, this is not the biggest failing of a centralized moderation system. Bots and scammers are common on all social media platforms, especially in the replies of celebrities like Elon Musk. Banned bot accounts are replaced within moments. Facebook alone has to remove half a billion bot accounts monthly. The large number of bots that inevitably arise requires a massive investment in systems to combat spam, which makes a competing platform difficult or impossible to start. Users rely on a centralized service provider to filter bots, but the will does not exist to solve the problem.

The incentives of social media companies are not aligned with their users' interests. As a result, social media companies actively make the experience of their platforms worse for their users. The goal of a social media company is to keep users endlessly scrolling on their platform for as long as possible so they look at as many ads as possible. Armies of the smartest people in the world train the world's largest computer models to make their platforms as addictive as possible, even if it comes at the cost of worsening a user's experience, damaging a user's mental health, or even causing societal instability. People come to these platforms for the content and to keep in touch with friends, but are held captive by antagonistic, secret algorithms. Not only are content recommendation algorithms not open source or even audited, but developers are not free to make alternative content recommendation systems. Users will inevitably not only want to consume different types of content, but will want content to be recommended in different types of ways. The one-size-fits-all method cannot serve these users.

Social media companies are some of the most powerful institutions in the world because of the impact they have over public dialogue, the types of news people see, and their unparalleled ability to send targeted messaging directly to users. In addition to having the power to present users with specific content, platforms also collect data that gives them a great deal of knowledge of their users, such as the users' political beliefs. These abilities, combined, give social media companies the potential to wield significant influence over a

population. More importantly, social media platforms are the public square of the modern day. Social media companies have complete, unilateral control over what happens on their platforms and yet are not accountable to the public. This dynamic grants social media companies a tremendous amount of power. Although social media companies have made great efforts to convince the public and governments that they do not abuse their influence, everyone would be better off if they did not have the power to do so in the first place.

Although much more could be said about the harms of social media and the internet, ultimately a centralized, managed platform being the only option to access social spaces is the root cause of the issues. The alternative to these platforms is not chaos, but competition. Open alternatives allow for experimentation between different policies that serve each user best. Even though everyone relies on the internet to a greater or lesser extent, opinion on the internet and its effects on humanity has gotten significantly more negative [1]. It is unclear what the long run effects of the internet are both on individual minds and society at large. Experimentation means more than just a product that works better for each user: it is necessary for humanity to develop a more symbiotic relationship with the internet. No individual can know how to achieve this end. But through the creation of an open, credibly neutral, decentralized social layer for the internet, humanity as a whole may be empowered to find the answers.

To learn how to build one, read on:

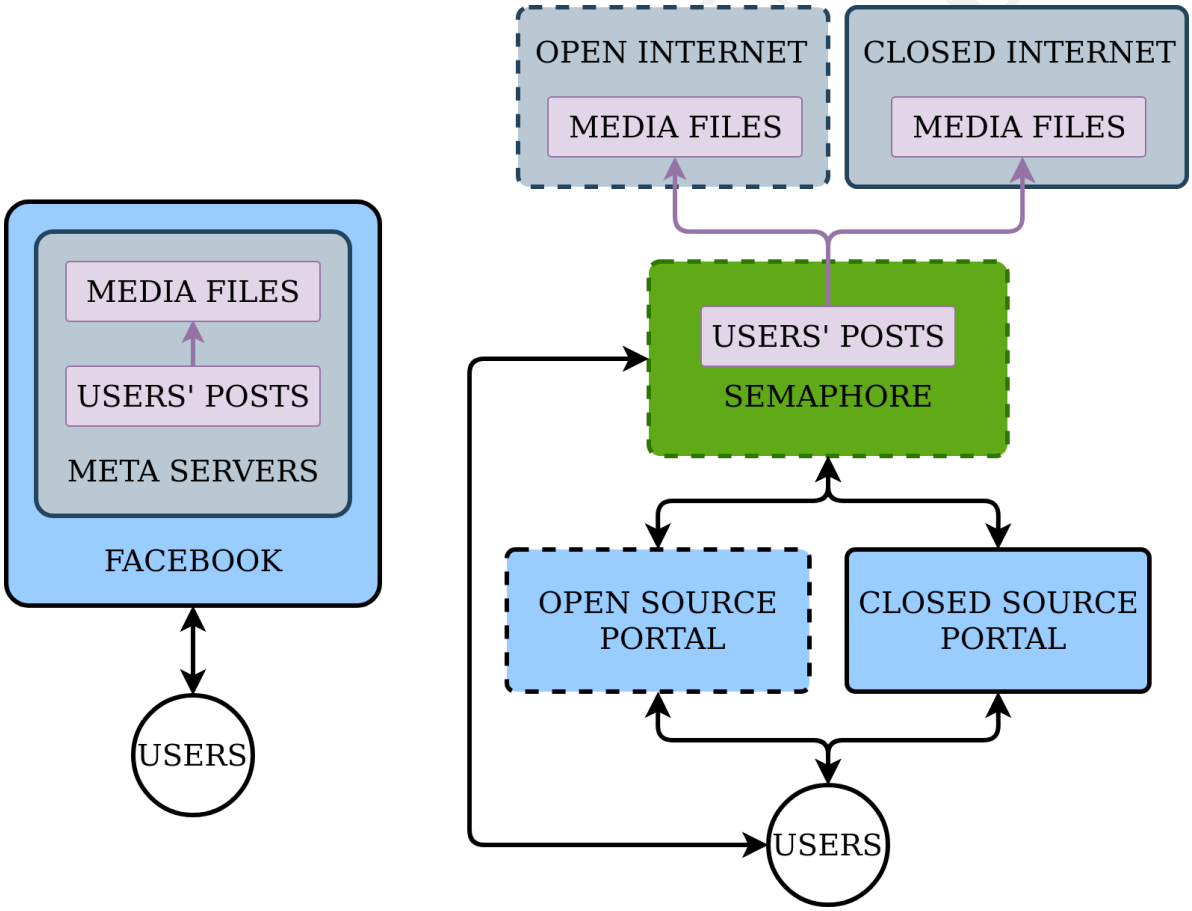
1 Introduction

We introduce Semaphore, a lightweight blockchain system that takes a new approach to allocating blockspace that makes it suitable to social interaction rather than financial transactions. There is not a network fee that must be paid to broadcast to Semaphore. Instead, users must hold an alias, which is managed by a separate blockchain. The number of aliases is finite but increases with demand, up to a theoretical maximum supply. What makes Semaphore interesting is not its "blockchain-ness." Instead, by creating a neutral platform for content, Semaphore allows for interoperable platforms to be built on top of the composable Semaphore network. In a world where Semaphore was successfully adopted, large content could be hosted anywhere — YouTube, AWS, or a private server — and pointers to the external content and small text based posts would be committed to the Semaphore network. Platforms built on top would aggregate, suggest, and moderate the content to the preferences of users. Users could run their own Semaphore node on their home PC or use a third party service to access the blockchain. Users would be free to manage their own keys, or use a service to handle key management for them. Semaphore's design allows for a user to control their own keys while allowing a third party to safely handle the day-to-day signing of messages without risking a loss of control in the event of a hack. Actually building such a system requires tradeoffs not seen in existing networks, but with good UX design these tradeoffs and all technical complexity can be hidden from users who simply want to use their social media in peace. Although there is, of course, real technical complexity in designing such a system, the result offers a real solution to the problems of existing social media.

Semaphore is an open source project that enables anything to be built on top. Parts of the ecosystem can be made to work together symbiotically or be built for entirely different purposes. However, each component can be built independently. Teams can build competing portals to the Semaphore network that are interoperable because each is allowing access to the same content. As new ideas are developed, teams can integrate the good ideas of others and bring the entire ecosystem forward. Furthermore, the incentives of one team do not necessarily spill over into other parts of the ecosystem. If one project decides to change its product to fit a new business model, those changes are not forced onto other parts of the ecosystem. Additionally, teams are free to explore niche use cases that would not sustain a monolithic social media platform. In the status quo, a social media platform must be very large to generate enough content to keep

users interested. There is no room for niches to be explored because there would not be enough users. In a world where Semaphore was successful, these niche products would have the opportunity to thrive because they can access the content of the entire network, not just the content generated by their own users.

Most users, even those with the technical knowledge, will not want to customize every aspect of their social media app. Service providers would offer such users complete products that act as their portal to Semaphore. Some may offer advanced customization features, others may be fixed like existing social media platforms. Most importantly, all portals will effortlessly work together since the Semaphore network standardizes all communications. Even though most users will use a product to access Semaphore, since they are free to switch at any time, it is impossible for the companies behind these platforms to exploit their users or force unwanted changes onto them. Since all offerings are simply different portals to the same content, companies must always compete on offering a better service rather than resting on their laurels and relying on network effects. Semaphore builds network effects, not the particular portal a user chooses to access the network. Some portals may rely on ads, some may use subscriptions, and others may use business models that cannot yet be imagined because there is not fertile enough ground for such innovation. Regardless, portals will need to compete on improving the quality of time users have, rather than maximizing the quantity of time that social media monopolies do in the status quo.



In Semaphore, since the total alias space is limited, it is impossible to create infinite bot accounts. This is because there is always a small cost associated with accessing a new alias, say $1. This cost could be borne either by the individual user or their portal service provider. Therefore any bot activity expected to yield less than $1 per bot account (which is essentially all of it) will not be practical. As soon as the bot is blocked by a user, the person behind the bot activity will need to pay the cost of an alias again to continue

interacting with the user. That of course assumes that there are many aliases on the market. If nobody is willing to sell their alias to a scammer, then it is not even possible for them to make a bot account.

Some aspects of Semaphore's design are likely to curb the worst of this type of interaction. Real users are under the same constraints as bots, meaning if they get blocked by another user for, say, harassment or violating a policy of a moderated community, the blockee cannot trivially make another account to get around the block. While this is helpful for individual users, it is especially powerful for moderated communities, where users are expected to follow certain rules. In addition, users who post at an excessive rate may be rate limited, incentivizing putting more thought into each post. For both of these reasons, users are likely to be more cordial with each other on the platform. Most users would never notice either of these constraints, but the most toxic users would not be able to use Semaphore in the toxic way that is enabled by platforms like Twitter. Since the long tail of toxic content online is a small overall percentage of content, curbing just the small fraction of posts that are very toxic would likely go a long way towards changing the way users generally perceive the toxicity of the platform. Additionally, competing portals can implement different moderation policies that adapt to how the platform is used and the desires of the users. Users can also enforce moderation action on the Semaphore network itself that are carried over into all portals.

In contrast to current paradigms, service providers for Semaphore are only there to help users interface with the actual network. Semaphore is not the service providers' platform. It is the users' platform, and service providers are only there to help make it easier to use. If a user is unsatisfied with any aspect of a service provider's product, the user is free to switch to a different service provider or to interface with the network directly, while still retaining access to all of the same content. The fact that users are able to interface with the network directly means that for the most part they will not need to because a service provider cannot trap a user. Facebook no longer needs to compete for users' attention because of the large network effects its products have built, so users are at the whim of any changes they might decide to make. Semaphore service providers cannot build network effects and must therefore always compete on providing the best possible user experience.

2 Network Rights Trilemma

There are fundamental tradeoffs in network design that enable or handicap certain applications. Semaphore is designed to be a new type of decentralized network that exists in a category distinct from Bitcoin, Ethereum, etc. The tradeoffs that Semaphore makes make it useful for social interactions, which typically have low willingness to pay, at the expense of financial applications, which existing crypto networks have explored.

2.1 Three Properties

Some terms defined, as they relate to communication networks:

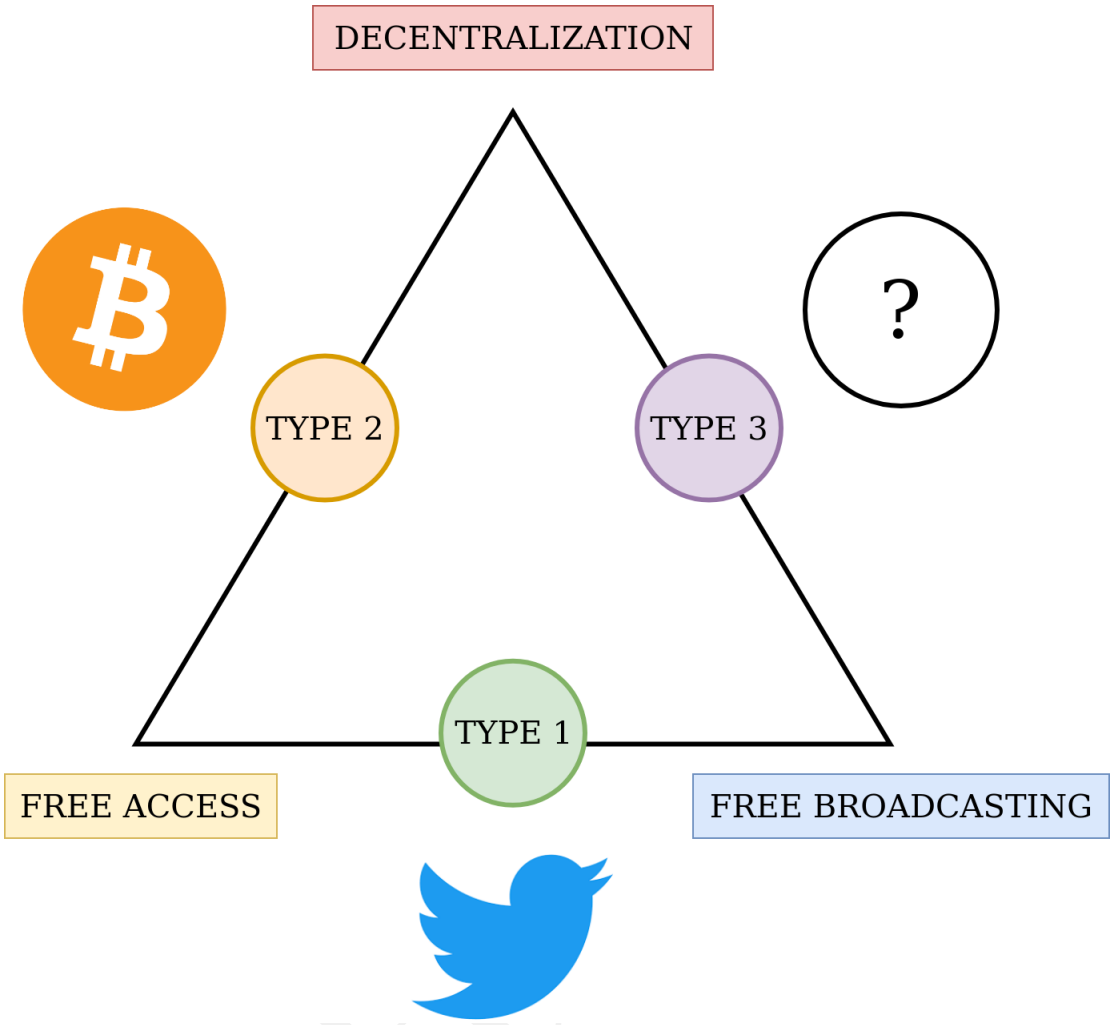
- **Decentralization**: The ability for an individual user to inexpensively receive and validate network activity without relying on a central party
- **Free broadcasting**: The ability for an individual user to broadcast a message to the members of a network at trivial or zero cost
- **Free access**: The ability for an individual user to be recognized as a participating member of the network and be guaranteed access to the functions of the network at trivial or zero cost.

2.2 Bandwidth, Computation, and Storage Tradeoffs

All things being equal, each is a desirable property of a communication network. However, a network operating at scale can only achieve two out of the three due to bandwidth, computation, and storage constraints. A network like Twitter has free broadcasting and free access, but is not decentralized. Twitter could not be made decentralized because it would be prohibitively difficult for an individual to personally run Twitter's servers. It would also be unmanageable for an individual to combat the large amounts of spam messages that would certainly flood the network. The Bitcoin network is decentralized and has free access, but not free broadcasting. It is cheap and easy to run a Bitcoin node [2] and free to make an unlimited number of wallets, but a fee must be paid to the miners to broadcast a transaction [3]. If the Bitcoin network was designed to eliminate miner fees instead, its blockchain would grow so large that users would find running a node difficult or impossible, hurting decentralization. We call this tradeoff the "network rights trilemma." Semaphore's decentralization is foundational, so the design must choose between either a guarantee of free broadcasting or free access. A social media network requiring a payment to broadcast messages would have terrible user experience and is not a viable design. Therefore the Semaphore opts to enable free broadcasting, but at the cost of giving up completely free access to the network. Because the total number of aliases that are available is kept limited but can expand over time, anybody can acquire a valid alias to access the network but there is always a cost to doing so.

2.3 Three Network Types

Since one of the three properties cannot be achieved, there are three distinct types of networks that can theoretically be created. Type 1 networks, which sacrifice decentralization, are classic "web2" networks, such as Twitter, Facebook, or YouTube. Type 2 networks, which sacrifice free broadcasting, are the space of "web3" networks that have been explored up until the current date, such as Bitcoin and Ethereum. Type 3 networks, which sacrifice free access, are a new design space of decentralized networks that has not yet been explored. The remainder of this paper seeks to explain how such a system can be designed and what its applications are. Although there are clear applications in the social rather than financial space, it is likely that there are applications beyond those outlined here, just as crypto projects after Bitcoin have exploited niches in the market that Bitcoin was never designed to address. Type 3 networks are not an improvement over type 2 networks: they are a fundamentally different type of network, optimized for different types of engagement. It is natural that a network that requires fees to use would be optimized for financial use cases because users can expect the financial benefit of their transactions to outweigh the cost of fees. People do not naturally pay per social interaction, so such applications would naturally fit into a network that does not requite payments for usage.



2.4 What About Email?

Initially, email might seem like it achieves all three properties. SMTP does not specify any central server, has no cost to sending emails, and no cost to create an email address [4]. In practice, however, email is very centralized [5]. Although it is possible for an individual user to run their own email server, it is likely that Gmail, Outlook, Yahoo, etc. will reject the server. As a result, very few people run their own email server. Historically, a decentralized email network was unworkable because the spam problem was so significant that the entire system became unusable without central parties to manage servers that filtered spam. Interestingly, Hashcash was an early attempt to create email as a type 2 network, although it was not conceptualized that way at the time [6]. Email makes the mistake of making a spec that does not explicitly acknowledge its tradeoffs, even though the tradeoff is unavoidable. As a result, email failed to achieve its original vision of decentralized electronic mail. This tradeoff is central to Semaphore's design. By embracing what might initially seem like a significant downside, mitigations can be implemented and the design can significantly improve the user experience.

3 Aliases

Owning an alias is analagous to having an account with the Semaphore network. Owning an alias means that a user has associated a public key with one of a finite number of aliases, and can sign messages on the semaphore network with that public key. Ownership of aliases is handled by an external blockchain, the state of which is periodically checkpointed into the Semaphore chain. Each alias represents a number, but the number can also be represented by a human readable string.

3.1 External Blockchain Functions

The alias contract runs on an external, EVM [7] compatible blockchain so that the Semaphore protocol can be simpler and so that semaphore consensus does not need to secure valuable tokens. An alias has two states **activated** and **deactivated**. When an alias is deactivated, it behaves equivalently to an ERC-721 token [8] and is not interpreted by the Semaphore network. When an alias is activated, it is linked to a public key and frozen, so that it cannot be transferred. When an alias is activated, it is interpreted as a valid user alias on the Semaphore network, with the alias' linked public key as the user's Semaphore public key. Upon creation, an alias is in the deactivated state.

The alias contract has two additional functions beyond that of an ERC-721: `activate(string _pubkey)` and `deactivate()` and two additional variables: `activated` and `pubkey`. The `activate` function sets the contract state to activated and sets the linked public key to `_pubkey`, freezing the alias. The `activate` function only executes this functionality if the alias is currently deactivated. It is represented by the below pseudocode:

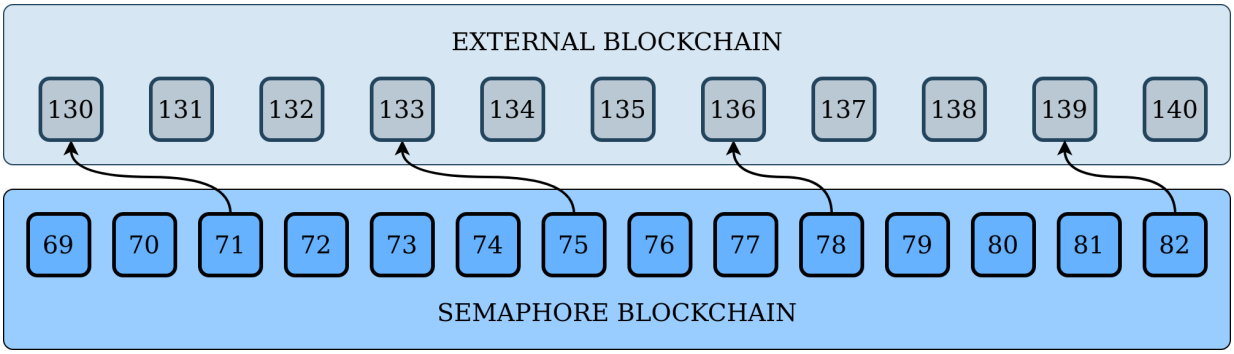
```
activate(string _pubkey):
      if !activated:
            pubkey = _pubkey
            activated = true
```

The `deactivate` function sets the contract state to deactivated and sets the linked public key to `""`, unfreezing the alias. It is represented by the below pseudocode:

```
deactivate():
      pubkey = ""
      activated = false
```

3.2 Checkpointing

It is necessary for Semaphore to stay in sync with the external chain. If the external chain reorgs, it is necessary for Semaphore to reorg as well. If there is ever disagreement between the two chains about which aliases are associated with which public key, it may result in unresolvable disagreements between nodes. To prevent such issues, block headers of the external blockchain are included into the Semaphore blockchain. A Semaphore block is considered invalid if it or any prior block contains an external header that is invalid on the external chain. Periodically, such as every 100^{th} external block, an external block header is included in the Semaphore chain, after short delay. The short delay is to ensure that nearly all Semaphore nodes have seen the external block. The inclusion of an external block header is called a **checkpoint**.



Semaphore does not guarantee that the external block header is included in a particular Semaphore block, since it is possible that nodes will take different times to receive the blocks, but does guarantee that the chain picks a single block in which the header is included. The precise mechanism of including the block header is explained in Section 8.4. Simply put, an external header is included when a majority of validators chooses to include it. For an external header to be valid:

- it must exist as a header of the external chain.
- the block index of the header mod 100 must equal 0.
- the block index of the header must be greater than the most recent header checkpointed into the Semaphore chain.

Not including any external block headers does not invalidate the Semaphore chain. Although there could be a requirement that an external block header be included after a certain number of blocks, this requirement is undesirable. In the rare event that something goes wrong on the external chain, it is good for Semaphore validators to be able to isolate the Semaphore alias public keys from the external chain. If a new external block header is not included in the Semaphore chain, then the alias public keys cannot update until the next external header is included. All other functionality continues without interruption.

Henceforth, "block," "chain," "blockchain," etc. refer to the Semaphore chain and not the external chain unless explicitly stated otherwise.

3.3 Interpretation of External Blockchain Functions

Each valid alias represents a number in the total valid alias space: $[0, 2^{32} - 1]$, a four byte number. However, not all aliases are minted initially. For instance, only aliases in [0, 10,000] may be initially minted, but the alias space later expands to [0, 500,000] and so on. Of the minted aliases, only activated ones can broadcast to the Semaphore network. When an alias is activated on the external blockchain, Semaphore nodes update their local alias database, setting the alias' public key to the public key, `pubkey`, linked on the external blockchain. The update is executed when the external block containing the activation transaction is checkpointed into the Semaphore chain. However, broadcasting on the Semaphore network is only enabled once the next external block is checkpointed. If a user owning an alias wishes to broadcast to the Semaphore network, the user must sign using the private key corresponding to the alias' public key, not the private key of the address that holds the alias token. When the alias is deactivated on the external chain, broadcasting is disabled from the alias and the database value for the alias public key is set to `NULL`. This process is executed when the transaction containing the deactivation is checkpointed into the Semaphore chain.

3.4 Key Management

Users may want to keep the private keys for their alias in a secure location that is inconvenient to use to sign broadcasts on a day-to-day basis. To reduce friction for users, they may set **temporary keys**, different from the **master key** stored on the external blockchain. Temporary keys can be set by signing a message from the master key committing to the temporary keys for a certain period of time. The temporary keys are considered valid for the alias until the period of time elapses or until the master keys revoke the private keys. Users may then keep temporary keys stored on their device and revoke the keys if they are hacked or lose their temporary keys. One of the most significant barriers to mainstream crypto adoption is the difficulty of managing one's keys safely. Semaphore's design, enabling **owner/operator key separation**, makes this challenge significantly more manageable. The ability to separate day-to-day signing keys from keys representing ownership is a significant improvement in user experience because the hot keys do not put the user's ownership at risk. Additionally, separation of keys allows businesses built on top of Semaphore to attract users. A business could offer users free aliases as long as they use their platform for a certain time. During that time, the user would own the day-to-day key, but the business would own the key representing alias ownership.

3.5 Wrapped Names and Custom Nyms

Rather than only being able to refer to an alias by its number, each number can be mapped to a human readable name with one to one correspondence. Each 4 byte number is interpreted as a concatenation of two 2 byte numbers. Each 2 byte number corresponds to one of 65,536 human readable words. Therefore, given an **alias number**, the wrapped name can be found, and vice versa. For instance, the alias number `10101010 00001001 10100101 11100000` can be represented as "Inaro-Obeld." `10101010 00001001` mapping to "Inaro" and `10100101 11100000` mapping to "Obeld." Although this system of wrapped names does not have any direct effect on the Semaphore network, it is a great improvement in user interface because of the improved readability and memorability of the wrapped names over their corresponding numbers.

Users may desire a custom username, or **nym**, other than the wrapped name provided by their alias number. There are a variety of ways that this could be accomplished and a full investigation of each is future work. All methods have in common the ability to map an alias number to an arbitrary string, as long as that string is not already mapped to another alias number, similar to how ENS maps unique strings to Ethereum addresses [10]. This allows users to set any name they wish to their alias in a method similar to existing social media/message board systems.

4 Bootstrapping Incentives

It is important for the incentives of speculators, who are inevitable, to align with the long run health of the network and not have negative externalities. By allowing the total number of aliases to increase over time, this can be enabled. As new people want to join the network, new aliases can be created rather than simply allowing the price to rise. Over time, the cost of an alias should fall, making the network more accessible as it grows. New aliases can go to existing holders, preventing the dilution of early holders. Profit of speculators is maximized by creating genuine community growth, rather than pumping and dumping because new users increase the value of the network for everyone.

4.1 Incentives of Typical Token project

Most token projects do not have a mechanism to change supply in response to demand. Most NFT projects have a total supply that is fixed. Therefore, as demand to be a holder of the token increases, the result is

an increase in the price of the token. For a speculative user whose only incentive to maximize profit, their incentive is to increase the price of the token as much as possible and attempt to "sell the top" before price decreases. Since essentially all successful NFTs are Veblen goods [9], building utility for the internet as a whole on such projects is very difficult. The market for aliases is similar to an NFT market in some ways but is different in others. Each alias is a unique token that can be bought or sold on the market like any other NFT. However in the Semaphore system, the total number of aliases can be increased. Therefore, an increase in demand to hold an alias can result in an increase in supply. Additionally, owning an alias of course provides the holder with the functionality of being able to broadcast to the Semaphore network, rather than being simply a receipt for a piece of artwork.

4.2 Superliner Network Value

According to Metcalfe's law, the value of a network is proportional to the square of the number of users [11]. This would suggest that if the total number of aliases increases N times, the total value of the network would increase N^2 times. It is impossible to know if the value grows in proportion to exactly the square, but an increase of N times would surely result in a total value increase of greater than N times, as long as the increase in the number of aliases represents an increase in actual users. As more users join the network, there is more content for all of the existing users. More users means that there is a greater incentive to build applications on top of the network and exploit niche use cases.

4.3 Alias Emissions

When a new alias is created, someone must be the holder of the alias upon its minting. The emission of new aliases can be either dilutive or nondilutive to existing alias holders. Of course, diluting alias holders does not reduce the functionality of their aliases, but is nonetheless undesirable. Diluting early alias holders not only creates risks of centralization, because a monolithic organization gets to decide who can enter the network, but also punishes early users for acquiring aliases sooner rather than later. To create a social layer for the internet that is owned by its users, it is wrong to forcibly reduce the ownership of early adopters by significantly diluting their ownership of the network. Furthermore, for the network to expand, the cost of an alias will need to drop over time. The initial aliases created will be significantly fewer than the number that would be desirable for a thriving social platform and may have a higher price than most would be willing to pay. The decrease in price over time can be made up for by increasing the quantity of aliases owned by existing alias holders.

A metaphor: Holding an alias is analogous to holding a plot of land in the island of Semaphore. There is only a finite amount of land. To broadcast to the Semaphore network, a user must own a plot of land of at least a particular size. This means only a finite number of people can fit on the island. Over time, the required size of a plot gets smaller so that more people can join the island. Reducing the size of a plot does not create more land, it simply makes it more granular. If existing alias holders get to mint one additional alias, this is equivalent to the required size of a plot reducing by half. New alias should therefore not be thought of as a reward for holding an alias or even anything new being created at all, but instead thought of as allowing one's existing ownership of the Semaphore network be expressed on the external blockchain in a more granular way. Therefore when early holders of aliases acquire new aliases, they are only getting the ability to subdivide the land they already owned, rather than getting paid in new land as a reward for holding old land.

4.4 Incentive Compatability

It is not possible to create a decentralized system that is not incentive compatible for both users and speculators. Bitcoin could not have gained popularity if early users did not have an incentive to mine, for example. Once aliases are available on the market, it is impossible to prevent speculation on the price, as is very common with existing NFT projects. It is therefore essential that the incentives of speculators are aligned with users and the health of the network. By not diluting existing holders as new aliases are created, and instead distributing them in proportion to ownership, this can be achieved. As demand to enter the network increases, price tends to increase, but this increases the cost of entering the network and will eventually reach a point where nobody else wants to acquire an alias to join the network. If an increase in price is stopped by increasing the number of aliases, the total value of the network can increase as holders most willing to sell decide to sell their additional aliases. Alias holders who decide not to sell will own the same percentage of the network, but benefit from the positive externality of more users on the network.

5 Time

Nodes communicate with each other to keep a time that is synchronous between nodes and accurate to the real time. The time is used to decide if messages have valid timestamps and coordinate other network functions. The formula used to calculate time is resistant to timewarp attacks and security is demonstrated mathematically and through simulation.

5.1 Timestamping Purpose

Nodes on the network must ensure that they keep a synchronous view of the current time to ensure useful timestamping and stay in consensus. Not only does the timestamping of messages does play a vital role in the operations of the network, but it is also very useful for human facing application. For instance, in nearly every communication system from Twitter to Discord, messages are timestamped and Semaphore includes this functionality. More importantly, nodes will make decisions about whether or not to accept a broadcast based on the time the node receives the broadcast. If nodes have very asynchronous times, they will disagree about what should be accepted. Although a small amount of time de-synchronization is inevitable and not a major issue, disagreement is best minimized. Nodes split all times into discrete epochs of a few seconds, with each block containing the activity of a single epoch. De-synchronization must be less than the length of an epoch or nodes may disagree about if broadcasts are valid. Only if an attacker controls a sufficient fraction of aliases may the timing of nodes be manipulated beyond this threshold.

5.2 Maintaining Synchronicity

A node maintains two distinct times: **local time** is the time determined by the node to be the true time. **network time** is the time that the node reports to its peers as the current time and is adjusted over time based on a node's local time and the reported network times of a node's peers. It is likely that nodes will have small disagreements in their respective local times and the purpose of adjusting network time is to remove this disagreement. To update a node's network time (T_{net}), it samples k peers and averages the reported times (T) to generate a sample time (T_{sample}) of the perceived times on the network.

$$T_{sample} = \frac{\sum_{i=1}^{k} T_i}{k}$$

New samples are used to update T_{net} via exponential smoothing with smoothing factor α and T_{net} is initialized to T_{loc} before sampling any peers.:

$$T'_{net} = \alpha T_{net} + (1 - \alpha) T_{sample}$$

Each node also keeps a reporting error threshold (R). If $|T - T_{net}| > R$, then the reported time T is rejected and the node samples a different peer. A node will stop sampling a peer who's reported times it rejects for the duration of a punishment period. Nodes also keep a time error threshold (E). If the node's $|T_{net} - T_{loc}| > E$, suggesting that an attacker has made the network clock run fast or slow, then a node will go into a "countering" state and alter its reported network time to bring the network back in sync with the true time by adding a correction factor (C) to its reported network time. C is updated each round with the following formula, where d is the correction drift rate and ϵ is a safety factor to reduce the number of peers who reject the altered time:

$$C' = \begin{cases} C - d, & \text{if } T_{net} - T_{loc} \geq E \text{ and } |C| < E - \epsilon \\ C + d, & \text{if } T_{loc} - T_{net} \geq E \text{ and } |C| < E - \epsilon \\ C - \text{sgn}(C) \cdot d, & \text{otherwise} \end{cases}$$

Accounting for these corrections, the formula for a node's reported time is as follows:

$$T = \begin{cases} T_{net}, & \text{if } |T_{net} - T_{loc}| < E \\ T_{net} + C, & \text{otherwise} \end{cases}$$

5.3 Managing Ping

Due to network lag, when a node queries a peer for its network time, once the response is received some time will have passed, meaning that the reported time will no longer be accurate. If there were no ping, the difference of network time and reported time would be the inaccuracy of a node's network time compared to the sampled peers. However, in reality the lag between the peer and the node is added to the difference. This effect can be mitigated with the following technique: The difference in time passing since querying peers and the change in reported time and network time can be analyzed to estimate ping. A specific ping correction factor (P) can be calculated for each sample time based on the measured lag from query to response (ΔT).

$$P = T_{net} - \frac{\sum_{n=1}^{k} T_n}{k} - \frac{\sum_{n=1}^{k} \Delta T_n}{2}$$

If the lag to a peer was identical to the lag from the peer, then P would exactly remove the effect of network lag. However this is unlikely to be the case. If the return trip is longer, then P will overestimate lag and vice versa. However, the expected difference between the trip to and from a node is 0: a trip from Alice to Bob is "away" from the perspective of Alice, but "towards" from the perspective of Bob. In other words, it is impossible for every node to have a shorter "to" time but longer "from" time. Therefore after applying P to the reported time, the expected value of the net effect of network lag is 0, rather than a strictly positive value.

5.4 Accuracy and Consensus

All nodes will have slightly different T_{net}. Nodes' T_{net} are said to have achieved consensus if their T_{net} values are sufficiently close in value that honest nodes do not reject the communications of other honest nodes. Suppose that there is a "true" global time T_{world}. Nodes' T_{net} are said to have achieved accuracy $|T_{world} - T_{net}| < e$, where e is some accepted error value, say 5 seconds. Both consensus and accuracy in the absence of byzantine actors are proved as follows: To begin, assume that the individual nodes times are i.i.d.

random variables that follow a distribution with mean T_{world} and standard deviation σ_T. Finally, denote the T_{net} (or T_{sample}) calculated at iteration i of sampling as T^i_{net} (respectively T^i_{sample}). If all nodes start sampling from scratch at the same time, then T^1_{sample} for each node is the mean of i.i.d. random variables. With a sufficient k, T^1_{sample} will be normally distributed via the CLT:

$$T^1_{sample} = \frac{\sum_{n=i}^{k} T^0_i}{k} \sim N(T_{world}, \frac{\sigma_T^2}{k})$$

When initializing the sampling process, nodes will use their local times for their initial T_{net} ($T^0_{net} = T_{loc}$). The network times at the first iteration will then be distributed as follows:

$$T^1_{net} = \alpha T^0_{net} + (1-\alpha)T^1_{sample} = \alpha T_{loc} + (1-\alpha)T^1_{sample}$$

$$E[T^1_{net}] = \alpha T_{world} + (1-\alpha)T_{world} = T_{world}$$

$$Var[T^1_{net}] = \alpha^2\sigma_T^2 + (1-\alpha)^2\frac{\sigma_T^2}{k} = (\alpha^2 + \frac{(1-\alpha)^2}{k})\sigma_T^2$$

$$T^1_{net} \sim N(T_{world}, (\alpha^2 + \frac{(1-\alpha)^2}{k})\sigma_T^2)$$

Since the node is averaging its local time with the reported time from k of its neighbors, the second iteration is summing the independent variable of its own time with the normally distributed times calculated in the sampling step. While T^2_{sample} and T^2_{net} are still normally distributed with mean T_{world}, their variances becomes:

$$Var(T^2_{sample}) = Var(\frac{\sum_{n=1}^{k} T^1_n}{k}) = \frac{1}{(k)^2}(k(\alpha^2 + \frac{(1-\alpha)^2}{k})\sigma_T^2) = (\frac{\alpha^2}{k} + \frac{(1-\alpha)^2}{k^2})\sigma_T^2$$

$$Var[T^2_{net}] = \alpha^2 Var(T^1net) + (1-\alpha)^2 Var(T^2sample) = \alpha^2(\alpha^2 + \frac{(1-\alpha)^2}{k})\sigma_T^2 + (1-\alpha)^2(\frac{\alpha^2}{k} + \frac{(1-\alpha)^2}{k^2})\sigma_T^2$$

$$Var[T^2_{net}] = (\alpha^4 + 2\frac{\alpha^2(1-\alpha)^2}{k} + \frac{(1-\alpha)^4}{k^2})\sigma_T^2 = (\alpha^2 + \frac{(1-\alpha)^2}{k})^2\sigma_T^2$$

In general, T^n_{net} will have mean T_{world} and variance:

$$Var(T^n_{net}) = (\alpha^2 + \frac{(1-\alpha)^2}{k})^n\sigma_T^2$$

The variance of T^i_{net} decreases rapidly to zero with each iteration if

$$\alpha^2 + \frac{(1-\alpha)^2}{k} < 1$$

$$\alpha^2 + \frac{(1-\alpha)^2}{k} < \alpha^2 + (1-\alpha)^2 = \alpha^2 + 1 - 2\alpha + \alpha^2 = 1 - 2\alpha(1-\alpha) < 1$$

The final inequality holds because $0 \leq \alpha < 1$ (and so $2\alpha(1-\alpha) > 0$). Thus the network time will converge rapidly to the "true" time, assuming the local times are independently distributed around T_{world}, achieving both accuracy and consensus.

5.5 Accuracy and Consensus with Byzantine Actors

In the presence of byzantine actors, showing that the error in the network time is bounded is less straightforward. Let us start with a few assumptions. For one, let b represent the fraction byzantine agents.

Additionally, let the local times of honest agents be normally distributed ($T^i_{loc} \sim N(T_{world}, \sigma_T^2)$) and let $T^i_{error} = T^i_{net} - T^i_{world}$ be the error in the time of the individual, honest agents.

5.5.1 Drift Attack

First we will analyze an attack in which the adversary attempts to cause a sustained drift to the network clock. During an attack that seeks to make the network run fast, we would expect that byzantine agents would report back a network time of $T_b = T^i + R$. In this case, we assume pessimistically that byzantine agents know the honest agent's exact time in order to report the maximum time before they get rejected.

On the other hand, honest agents will begin to use the correction factor C once their time drifts sufficiently upwards. Assuming that $T_i = T_{avg}$ for every honest agent, the fraction of agents that will be using a correction factor can be calculated as

$$P(T_{net} - T_{loc} \geq E) = P(T_{loc} \leq T_{net} - E) = P(z \leq \frac{T_{net} - T_{world} - E}{\sigma_T}) = \Phi(\frac{T_{error} - E}{\sigma_T})$$

Since C steadily increases until it reaches $E - \epsilon$, at an "equilibrium" (when T reaches a steady state) we assume that C will either equal $-(E - \epsilon)$ or remain at zero, depending on where that honest agent's local time falls on the normal distribution. At this equilibrium, T_{net} will remain the same from iteration to iteration, on average. Equivalently, $T_{sample} = T_{net}$. We can then express the steady state average T_{net} as:

$$T_{net} = E[T_{sample}] = E[\frac{\sum_{i=1}^{k}(T_i)}{k}] = b \cdot E[T_b] + (1-b) \cdot E[T_{net} + C]$$

$$T_{net} = b(T_{net} + R) + (1-b)(T_{net} - \Phi(\frac{T_{error} - E}{\sigma_T})(E - \epsilon))$$

$$T_{net} = T_{net} + bR - (1-b)\Phi(\frac{T_{error} - E}{\sigma_T})(E - \epsilon)$$

$$bR = (1-b)\Phi(\frac{T_{error} - E}{\sigma_T})(E - \epsilon)$$

$$\frac{bR}{(1-b)(E-\epsilon)} = \Phi(\frac{T_{error} - E}{\sigma_T})$$

$$\frac{T_{error} - E}{\sigma_T}) = \Phi^{-1}(\frac{bR}{(1-b)(E-\epsilon)})$$

$$T_{error} = E + \Phi^{-1}(\frac{bR}{(1-b)(E-\epsilon)})\sigma_T$$

This result only provides a bound on the network error when the expression inside the inverse norm is less than or equal to one:

$$\frac{bR}{(1-b)(E-\epsilon)} \leq 1$$

$$\frac{1-b}{b} \geq \frac{R}{E-\epsilon}$$

$$\frac{1}{b} - 1 \geq \frac{R}{E-\epsilon}$$

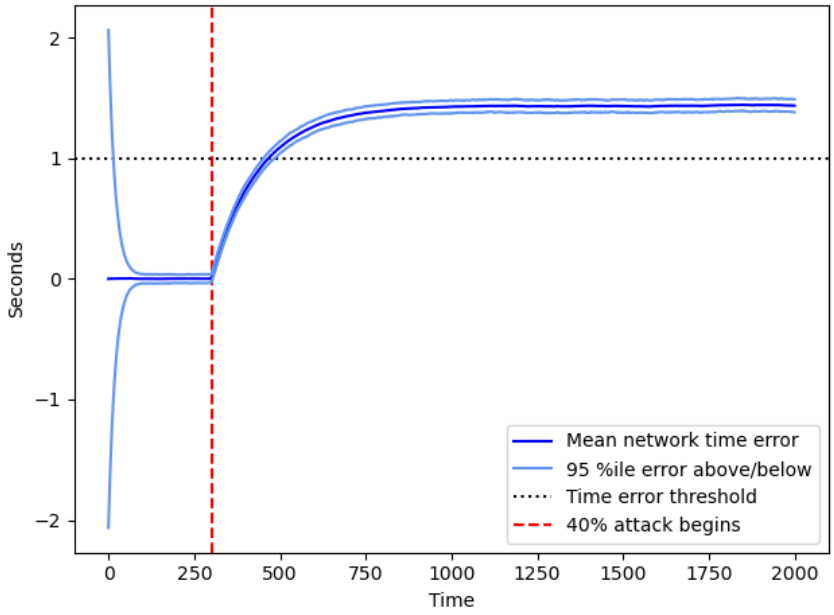
$$b \leq \frac{1}{1 + \frac{R}{E-\epsilon}}$$

$$b \leq \frac{E - \epsilon}{R + E - \epsilon}$$

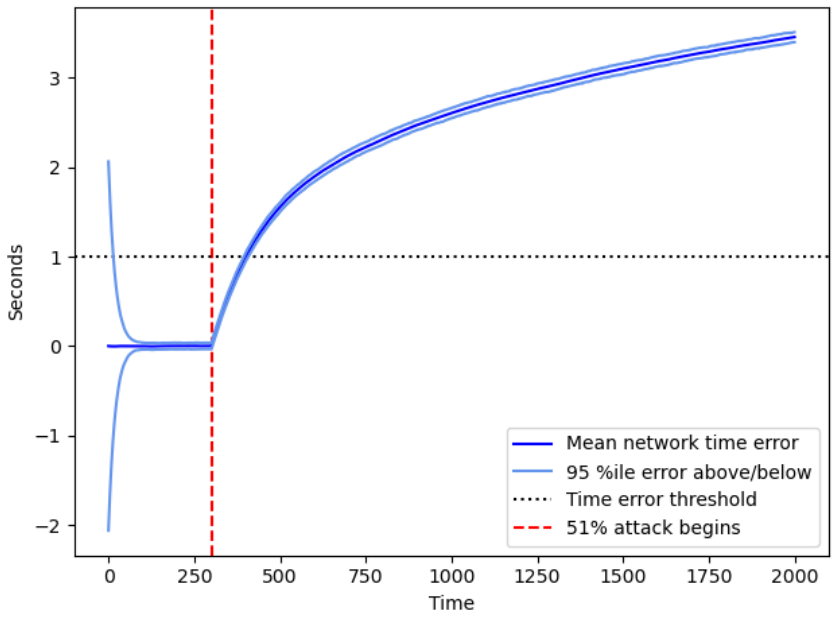
This means that the error in network time under this type of attack will be bounded as long as the proportion of byzantine agents follows the above inequality. At the extreme where $\epsilon = 0$ and $E = R$, this type of attack will succeed once $b > \frac{E-0}{E+E-0} = 0.5$. In other words, a 50% attack (at best) is required to make the entire network clock move in one direction, though in reality the threshold will be a bit lower.

Although this analysis looks only at averages, in reality the time of individual agents will move up and down due to the random nature of the sampling process, inserting variability in their correction factors. However, the total "weight" of the agents pulling the network times down with a correction factor will balance out the force of the byzantine agents moving it upwards

Below is a simulation of the drift attack on a network with 1000 nodes. Initially nodes come into tight consensus on network time. At time 300, 40% of nodes begin reporting times that are fast ($R = 1$ in this simulation). Nodes successfully keep the average network time bounded while maintaining tight consensus between honest nodes.



The below simulation is the same as the one above, except the attack takes place with 51% of nodes. Although honest nodes stay in tight consensus, they are not able to bound the time, allowing the attacker to run the clock fast.



5.5.2 Bifurcation Attack

Next, we analyze an attack in which the adversary attempts to bifurcate the network by reporting fast times to some nodes and slow times to others. Let T_s and T_f represent the average network times of the nodes that attackers are try to make run slower and faster, respectively. We assume (pessimistically) that Byzantine agents know the exact local times of honest nodes, and that they report times of $T_s - R$ to nodes with $T_{loc} \leq T_{world}$ and times of $T_f + R$ to nodes with $T_{loc} > T_{world}$. This type of attack would cause the clocks of the "bottom" 50% of nodes to drift downwards and "top" 50% of nodes to drift upwards, and has the goal of "splitting" the network by causing the two sets of nodes to reject each other's times. This will happen once $T_f - T_s > R$.

In a general scenario where attackers target a s:f ratio of the network to run slower/faster, "slower" nodes will reach an equilibrium when:

$$T_s = \frac{s}{s+f}(1-b)(T_s + C^s_{avg}) + \frac{f}{s+f}(1-b)(T_f + C^f_{avg}) + b(T_s - R)$$

Similarly, "faster" nodes will reach the equilibrium when:

$$T_f = \frac{s}{s+f}(1-b)(T_s + C^s_{avg}) + \frac{f}{s+f}(1-b)(T_f + C^f_{avg}) + b(T_f + R)$$

When the attackers target a 50/50 split of the network, the two network time T_s and T_f will move symmetrically such that $T_{world} - T_s = T_f - T_{world}$ (equivalently, $T^f_{error} = -T^s_{error} \equiv T_{error}$). It then follows that the correction factors in this scenario will also be equal and opposite ($C^f_{avg} = -C^s_{avg} \equiv -C_{avg}$). This greatly simplifies the equations above into one equation:

$$T_f = \frac{50}{50+50}(1-b)(T_s + C_{avg}) + \frac{50}{50+50}(1-b)(T_f - C_{avg}) + b(T_f + R)$$

$$T_f - T_{world} = \frac{1}{2}(1-b)(T_s - T_{world} + C_{avg}) + \frac{1}{2}(1-b)(T_f - T_{world} - C_{avg}) + b(T_f - T_{world} + R)$$

$$T_{error} = 0.5(1-b)(-T_{error} + C_{avg}) + 0.5(1-b)(T_{error} - C_{avg}) + b(T_{error} + R)$$

$$T_{error} = bT_{error} + bR$$

$$T_{error} = \frac{bR}{1-b}$$

As previously mentioned, the network will split when::

$$T_f - T_s > R$$

$$T_f - T_{world} - (T_s - T_{world}) > R$$

$$T_{error} - (-T_{error}) > R$$

$$2T_{error} > R$$

$$T_{error} > \frac{R}{2}$$

$$\frac{bR}{1-b} > \frac{R}{2}$$

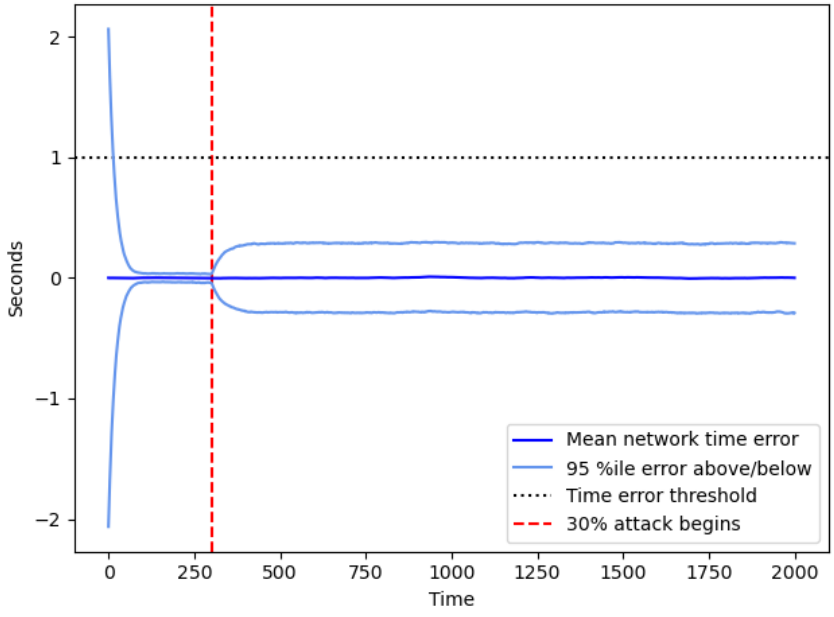
$$2b > 1 - b$$

$$3b > 1$$

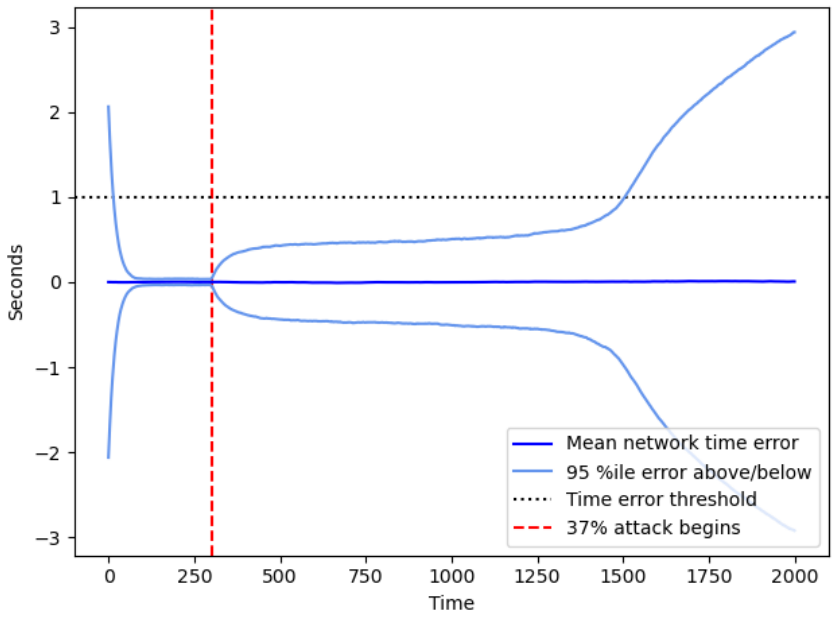
$$b > \frac{1}{3}$$

Thus, the network will split when more than a third of the actors are byzantine.

Below is a simulation of this scenario, where the attack sends fast times to nodes with a fast T_{loc} and slow times to nodes with slow T_{loc}. Additionally, executing this attack requires the attacker to know the T_{net} of each honest node that samples it. Although consensus between nodes is less tight, they remain in sufficiently tight consensus.



The below simulation is the same as the one above, except the attack takes place with 37% of nodes. Although initially after the attack honest nodes remain in consensus, they eventually fall out of consensus at around time 1500. This attack successfully splits the network in two, causing a fork.



5.5.3 All other attacks

Any other attack that pushes the network faster/slower with a different split will be a linear combination of the drift attack and the bifurcation attack. While a more general formulation could be established for an attack on the network clock, it intuitively follows that any other type of attack would require between one third and one half of the network, depending on the ratio of faster to slower nodes that the attacker wants to target.

6 Broadcasts

Users must sign all messages, known as broadcasts, to the network with their alias' private key. Broadcasts may reference other broadcasts, aliases, or external content. Broadcasts may contain instructions to the Semaphore protocol the change the functionality of a broadcast. Users have a variety of tools to moderate interactions with their content and join groups with moderation policies.

6.1 Broadcast Structure

A broadcast is a signed message from an alias that is sent to all nodes on the network. All broadcasts have a few parts in common:

- **Message**: The text content of a broadcast. This is the message that a user wishes to have broadcasted, such as the text of a post. All content in Semaphore is text, so if a user wishes to post media, such as a picture or video, the message must contain a pointer to the content.
- **Originating Alias**: The alias that is signing the broadcast.
- **Parent**: A pointer to another piece of content, user, or arbitrary data.
- **Chain Commitment**: Headers of prior blocks used to timestamp the broadcast and commit to a particular chain.

- **Signature**: Signs the contents of the broadcast to verify that the originating alias is authorized to send the broadcast.
- **Broadcast ID**: BCID for short. Not a part of the broadcast, but derived from it. The SHA256 hash of the contents of a broadcast, except for the signature.

`originating_alias`, `chain_commitment`, and `signature` are all a fixed length, but `parent` and `message` can be variable length. To remove ambiguity, a quantity `parent_len` is added to signify the size of the parent field. The total broadcast looks like: `sig`‖`originating_alias`‖`chain_commitment`‖`parent_len`‖`parent`‖`message`. The BCID is equal to `SHA256(originating_alias`‖`chain_commitment`‖`parent_len`‖`parent`‖`message)`. The signature is equal to `SIGN(SK,BCID)`, where `SIGN` is a digital signature signing function and `SK` is the alias' secret key. Semaphore uses the BLS digital signature algorithm [12]. The rationale for this decision is discussed in section 11.1.

The most basic broadcast is a **relay**. A relay does not refer to another broadcast with its parent field. A **reply** is a broadcast that is sent in response to a particular other broadcast. A reply refers to another broadcast with its `parent` field by populating it with the block index and originating alias of another broadcast, which is more space efficient than using the BCID of the broadcast. A reply can be in response to a relay or another reply. All replies to a relay, all replies to the replies, and so on are **descendant broadcasts** to the relay. A reply cannot refer to a broadcast in the same block because it may result in a reply being accepted by the network but not the parent broadcast. An alias may not post multiple times in a single epoch. This serves to both prevent denial of service attacks, reduce the number of bytes required to reference a broadcast, and to help prevent chain splits, which is discussed in section 9.

6.2 Message Operators

Users may add information to the message of a broadcast known as an **operator** that serves as instructions to the Semaphore protocol, rather than as human readable content. Some semaphore operators include:

- **like**: Indicates positive feelings towards a particular post. Used to increase the karma of the originating alias of the liked post. Karma represents the value of an alias' broadcasts to the users of the network and is discussed further in section 10.
- **dislike**: Indicates negative feelings towards a particular post. Does not decrease the Karma of the originating alias of the disliked post.
- **no_reply**: indicates that no post may reply to this post. Any broadcast whose parent is this broadcast's BCID is invalid.
- **resurrect**: Prove that a broadcast that has been purged from the blockchain was included at some point in the past (purging described in section 11.2).
- **block**: Block an alias from replying to the originating alias. All messages from the blocked alias that use a BCID from the blocking alias as broadcast parent alias are invalid.
- **unblock**: Ends the block of an alias blocked by the originating alias.
- **deputize**: Allow deputized alias to block on the behalf of the originating alias.
- **undeputize**: Remove deputized status from deputized alias.
- **temp_key**: sets a temporary public key.

An operator is signified by starting a message with "!" followed immediately by the operator command. For instance, a message would start with "!no_reply" to use the no_reply operator. Multiple operators may be used in a single message by including one immediately after another. If a node sees a message containing an unknown operator, the unknown operator is ignored, allowing additional operators to be added over time via soft fork. Some operators require additional information to function. To indicate what data goes with the operator, ":" is used. For instance: "!block:inaro-obeld" indicates that the originating alias wishes to block the alias "inaro-obeld." If a user wishes to start a broadcast with "!" but does not want to use an operator, a leading "\" may be used as an escape character.

Semaphore also has two reserved characters, "@" and "#", which serve as **soft operators**. These soft operators do not actually serve any role in the Semaphore protocol, but should be understood by applications as serving a particular function. "@" is used to point to aliases or other pieces of content. For instance, it should be understood that a message containing "`@inaro-obeld`" is refering to the alias "inaro-obeld." Similarly, since Semaphore broadcasts cannot contain media the "@" symbol can be used to point to content hosted externally. For instance, a user can use "`@https://www.youtube.com/watch?v=dQw4w9WgXcQ`" to post a video to the Semaphore network. "#" followed by a term is used to signify that the broadcast is related to the term in some way. For instance, a message containing "`#birdwatching`" indicates that the broadcast is related to the "birdwatching" topic and serves as an indicator for people searching for bird watching related content.

6.3 Moderation

Users may want to limit the possible interactions with their broadcasts, such as by preventing particular aliases from interacting. As previously discussed, a user may block an alias, preventing the blocked alias from replying to any of the user's broadcasts. It is necessary for this functionality to exist on the protocol level, rather than only on applications for two reasons. First, in the early life of the network before many applications are built, users need recourse against badly behaving aliases. Second, it allows for a greater level of consistency between different applications that may have different moderation policies. Since each user should be empowered to use the network in the way they desire, a user should be able to unilaterally prevent interactions they do not want and have that decision be enforced across all platforms.

Although blocking prevents direct interactions, it may not prevent all unwanted interactions. For instance, if Alice blocks Eve but Bob has not blocked Eve, if Bob replies to Alice, Eve may reply to Bob. Alice may want to prevent such an interaction from Eve and any interaction where Eve replies to any post that is connected to one of Alice's relays by any number of replies. To achieve this, Alice may put her own alias in the parent field of her relay. When the "inaro-obeld" alias is used in the parent field of a relay, its moderation rules are applied to all descendants of the relay.

One advantage of moderation that Semaphore has over existing, centralized social media services is that blocked users cannot trivially circumvent blocks by creating a new account. Since a new alias is required to circumvent a block, a blocked user will need to incur a cost to once again interact with the alias that blocked them. Knowledge of this cost would likely deter a significant portion of bad behavior in the first place.

6.4 Governed Communities

The functionality that allows a user to apply their own moderation to all descendant broadcasts also allows for the creation of **governed communities** that operate similarly to a moderated subreddit [13]. An alias is selected to be a **governor alias** and the owner of the governor alias deputizes other aliases to act as moderators. Users that wish to participate in the governed community may simply put the governor alias in the parent field of their relays and take advantage of the moderation policies of the governed community without needing to handle blocklists themselves. A moderated community may simply be a blocklist of

known bad actors or may be about a particular topic, such as a community dedicated to discussing bird watching. To manage a governed community by a team of moderators rather than an individual, ownership of the alias can be held with multisig [14] and each moderator can have their alias deputized for moderation.

7 Initial Relay

When a user wishes to broadcast a message, the originating node sends the message to its peers, who in turn send it to their peers, and so on. At each step, the broadcast is checked for validity and any violation of network rules is reported to peers and forwarded throughout the network. However, this gossip network is insufficient for peers to reach consensus since the validity of old broadcasts is ambiguous.

7.1 Gossip Network

New broadcasts must be disseminated through the network as quickly as possible to ensure that valid broadcasts are seen in time to be accepted by the network. Nodes listen for new broadcasts during an initial relay period for each epoch. The initial relay period lasts for the duration of the epoch itself, plus following **slack epoch**s. Nodes will signal support for broadcasts received during the initial relay in the next, consensus-reaching phase. For instance, if an epoch is 5 seconds and there are two slack epochs, a broadcast sent at the beginning of its epoch will have 15 seconds to traverse the network and a broadcast sent at the end of its epoch will have 10 seconds to traverse the network.

When relaying a broadcast from Alice's node to Bob's, Alice first sends the originating alias of the broadcast to Bob, in an offer to send the complete broadcast. If Bob has not yet received a broadcast from the originating alias in the epoch, he will request the complete broadcast. Upon receiving Bob's request, Alice will send the complete broadcast. After validating the complete broadcast, Bob will relay it to his peers in the same manner. If Bob already received a broadcast from the originating alias in this epoch, he will not request the complete broadcast. All messages from Alice to her peers are signed with her alias key so that her peers can prove if Alice sent a particular message. If Alice sends multiple messages at once in a batch, they are hashed into a Merkle tree and Alice signs only the root.

Initially offering the originating alias rather than the BCID reduces the bandwidth overhead of the gossip protocol at the cost of differentiation between different broadcasts from the same originating alias. However, since an alias is not allowed to originate multiple broadcasts per epoch, this is not an issue. The bandwidth overhead for a broadcast transmission drops from 32 bytes to only 4 bytes.

In a simulated network of 100,000 nodes with each node connected to 12 peers, the average number of hops necessary to reach any node is 4.86, with a worst case of 5 hops. With 8 peers, the average number of hops is 5.49 ,with a worst case of 6 hops. With a very pessimistic latency of 500ms between peers, the worst case propagation time for a broadcast to reach the entire network is significantly less than the minimum of 10 seconds allotted using the above example numbers. Actual latency is likely to be significantly lower [15]. Therefore, a node would need to have an exceptionally unreliable connection to not receive a valid broadcast before the end of the initial relay.

7.2 Malfeasance Proofs

If a node receives a broadcast that violates a network rule, such as a broadcast with an invalid signature, then the node will disconnect from the peer that sent the offending broadcast and send to its peers a proof that the peer forwarded an invalid message. If Alice sends Bob an invalid broadcast, the proof of her malfeasance is simply her signed message to Bob containing a broadcast with an invalid signature. If Alice sent many

broadcasts and only a particular one was invalid, Bob uses the offending message and its Merkle path as a proof. Further violations of network rules include:

1. Relaying a broadcast from a past epoch beyond slack epochs
2. Relaying a broadcast from a future epoch beyond a single slack epoch
3. Relaying a broadcast referencing a block that does not exist
4. Offering an alias that does not exist or is invalid
5. Relaying a broadcast that has an illegal parent
6. Relaying a malfeasance proof that is incorrect
7. Relaying any malformed data

There are some rule violations that will cause a node to disconnect but not send a malfeasance proof. These include violations that attempt to DOS attack the node. Since sending a proof would in itself DOS the nodes receiving the proof, no proof is generated. Such violations include:

1. Relaying a broadcast that consumes too many bytes
2. Sending excessive data after a request to stop

7.3 Proving a Negative

A node is only supposed to accept a broadcast as valid if it was actually originated in the epoch to which it is timestamped. A node will reject a late broadcast if it is sent directly from the originating alias. Similarly, it is easy for a node to detect if a broadcast is originated before the epoch to which it is timestamped because the node knows that the time of the epoch has not yet been reached. However, if the node receives a broadcast from a past epoch, the node can not unilaterally know if the node is late to receive the broadcast or if the broadcast should be invalid because it was originated after the epoch in which it is timestamped. It is impossible for a node to know on its own if a late broadcast is invalid or not because it is impossible to prove that a broadcast did not exist at a particular time. To resolve this issue and ration blockspace, the epoch vote process allows nodes to reach consensus on which broadcasts will be considered valid in a particular epoch.

8 Epoch Vote

The epoch vote allows validating agents to come to consensus on which broadcasts to include in a block even if not all of the nodes saw the same broadcasts during the initial relay. Consensus is achieved through repeated sampling that quickly "tips" towards either accepting or rejecting across the entire network. The number of byzantine agents that can be safely handled by the network depends on how many honest agents see broadcasts during the initial relay, but is sufficiently secure given reasonable propagation requirements. Each agent constructs its own block. In the absence of a significant number of attackers, agents will not produce competing blocks. In the event of a fork, the canonical chain is decided with a different system: proof of engagement, described in section 9.

8.1 Sampling

After the initial relay, validating agents will stop accepting new broadcasts from the gossip network. After ceasing to accept new broadcasts, agents must come to consensus on which broadcasts to accept as valid. Semaphore seeks to accept broadcasts that were seen by a sufficient fraction of agent at the end of the initial relay and reject broadcasts that were not. This is achieved via repeated random sampling of peers. This process is somewhat similar to Avalanche's Snow protocol [16] and has a similar intuition. It is assumed that every agent knows about and can connect with every other agent. This assumption can be relaxed, allowing an agent to connect with only a subset of all agents, with minimal impact on results. Once the sampling process terminates, nodes will have reached consensus on a block of valid broadcasts with very high probability, even when there are byzantine actors.

At the start of the epoch vote, all broadcasts are either known or unknown to each agent. If a broadcast B is known, the agent sets its **confidence level** for that broadcast s_B to an initial value i. Agents do not set a confidence level for broadcasts that are unknown because, obviously, they do not know about the broadcast. Let n be the total number of agents. With each round of sampling, an agent will sample k other agents at random, with replacement. Additionally, each agent keeps a count of the number of rounds of sampling, r, initialized to a value of 0 at the start of the epoch vote process. After each round of sampling, r is incremented by 1. When sampling k agents, an agent v responds with a set S_v of all broadcasts for which v has $s_B \geq 1$. If $B \in s_v$, then agent v has acknowledged or **ack**ed B (nacked, otherwise). The total number acks that B receives is the quantity A_B. Agents have a **quorum threshold** $q < k$. If, after a round of sampling, $A_B \geq q$ out of k sampled agents ack B, the agent increments s_B by 1. If $A_B \leq k - q$ out of k ack B, then the agent decrements s_B by 1. If there is no quorum $A_B \geq q$ either for or against B, meaning the total number of acks is $A_B > k - q$ and $A_B < q$ then s_B remains the same. If a broadcast B is in a sampled agent's set S_v, but not in the sampling agent's own set S, then the agent has learned about a broadcast that it did not know previously. The agent sets its confidence for the broadcast s_B to $-r - i$. If an agent finds that a new broadcast is a second broadcast from a single alias, it will nack all of the offending alias' broadcasts for the remainder of the epoch and a punishment period. A proof of the alias sending multiple broadcasts in an epoch is forwarded to peers. To maintain resilience to unreliable connections, agents will sample greater than k other agents and randomly pick k responses. If fewer than k responses are received, the agent will not perform any action.

Sampling continues for a total of T rounds, after which the epoch vote for the epoch terminates and each agent finalizes its block of broadcasts. Each agent has an **acceptance threshold** β. Once s_B reaches β or $-\beta$, the agent will not revert for broadcast B. An agent accepts a broadcast B as valid if $s_B \geq 0$ at the end of the epoch vote. With only an arbitrarily small probability will agents fail to unanimously agree on which broadcasts to accept, as demonstrated in section 8.3.5. If not all agents agree, then a fork has occurred. The method for resolving forks is explored in section 9 section and the probability of forks is discussed in section 8.3.

Rather than sending the entire set of acked broadcasts each round, nodes save bandwidth by sending a parity bitmap sketch [17] of the set of their acked broadcasts. On average a node must send only two times the theoretical minimum necessary information. Unpacking a sketch has linear computational complexity. As nodes come closer to reaching consensus, the size of the sketch sent can decrease. Once nodes believe they have reached consensus they can send a hash of their set and only use the sketch as a fallback. These techniques significantly reduce the communication overhead necessary as the number of broadcasts sent increases.

8.1.1 Drift

There is an additional **drift requirement** for an agent to ack a broadcast: the agent's confidence for the broadcast must have a positive drift. Each agent keeps a drift parameter (d), lag parameter (l) and saves the history of its confidence for each broadcast (B) in each round j (s_B^j). For the first l rounds, an agent will ack B if $s_B > 0$. After l rounds an agent will ack B if $s_B > 0$ and $s_B - s_B^{j-l} \geq d$.

8.2 Safety and Liveness Intuition

There are two main classes of attack that the epoch must resist: censorship attacks (safety), in which an adversary tries to prevent a valid broadcast from being accepted by the network, and unanimity attacks (liveness) in which an adversary attempts to make one set of agents accept a broadcast and another set of agents not accept the broadcast. First we analyze the censorship attack without agents having a drift requirement (ie: $l = \infty$). Let a represent the number of honest and accurate agents, b represent the number of byzantine agents, c represent the number of honest and inaccurate agents, such that $a + b + c = n$. For a censorship attack, byzantine aliases will never ack the broadcast. The odds of a sampled agent acking a particular broadcast is therefore $\frac{a}{n}$ and the odds of a sampled agent nacking a particular broadcast is therefore $\frac{b+c}{n}$. If $a > \frac{n}{2}$ then with each additional sample, it is increasingly likely that a majority of samples will be accurate. With each round of voting, we therefore expect that c will decrease and a will increase, making future rounds even more likely to be majority accurate.

However, excluding the drift requirement leaves the system open to a different type of attack. If $a \approx c$, then byzantine nodes can prevent honest agents from making progress by switching if they ack or nack a broadcast. For instance, if $a = 45$, $b = 15$, $c = 40$, if 5 byzantine agents ack and 10 nack, the network appears to be exactly split and the expected change in confidence is 0. If through the random sampling, $a = 46$ and $c = 39$, byzantines can instead ack with 4 agents and nack with 11. If this attack continues for the entire epoch vote, then agents will not reach unanimous consensus on if a broadcast should be accepted. To prevent this, nodes require a certain level of positive drift in confidence, or else a broadcast is rejected. Even in the absence of byzantine agents, split broadcasts take many more rounds to reach consensus than broadcasts that have less balanced coverage among honest agents, so it is desirable to reject such broadcasts because all honest broadcasts will have higher coverage due to the propagation time afforded by slack epochs. Of course, byzantine agents can try to split the network by targeting the required level of drift exactly instead of zero drift, such that some agents would meet the drift threshold and others would not. However, with each round, an increasing number of byzantine agents will need to ack the broadcast until all are acking. This situation is equivalent to the censorship attack, except all byzantine agents are acking instead of nacking. Therefore, once the drift threshold requirement is added, if the epoch vote process is robust against censorship attacks, it will also be robust against unanimity attacks as long as there is enough time for the process to run. However, the proportion $a : c$ initially must be higher for a higher drift threshold to not cause broadcasts to be rejected. With sufficient propagation time allotted for honest broadcasts, this is not an issue.

There is one more attack that must be accounted for: bifurcation+censorship attack (bifurcation attack from section 5.5.2). If an attacker has insufficient aliases to cause a successful bifurcation attack, there may still be some disagreement between honest nodes in regards to the network time, such as half of nodes running one second fast. This means the fast half of nodes will start the epoch vote before the slow half. Before the slow half starts the epoch vote, the fraction of byzantine agents will be greater than it should be: $\frac{b}{a+b}$ instead of $\frac{b}{n}$. If the byzantine agents can make the honest agents lower their confidence before the slow agents start the epoch vote, valid broadcasts my not get accepted because $b + c > a$. However, honest against will have connections to both fast and slow peers. When slow agents are sampled by fast agents, the slow agents will respond with a time error, indicating that they have not started the epoch vote yet. If a sufficient proportion of responses include that time error response, agents will not update their confidences

that round. After a time limit of greater than the maximum possible bounded time bifurcation, agents will stop paying attention to time error responses so that byzantine agents cannot stop the epoch vote by simply indicating that it hasn't started yet.

8.3 Safety and Liveness Analysis

Next we mathematically analyze the epoch vote process without a drift factor. the mathematical analysis including the drift factor is in progress. However, since unanimity attacks resolve to censorship attacks with a flipped sign, the analysis intuitively should apply to all attacks.

8.3.1 Parameters

$n = \#$ of agents.
$b =$ fraction of the n agents who are Byzantine
$a > 0.5 =$ fraction of the n agents who are initially honest and accurate.
s = an agent's confidence in correctness. The initial values are > 0 and < 0 for (honest) agents who are informed correctly and incorrectly, respectively. $d = \#$ an agent samples (with replacement) in an iteration. $q = \#$ out of k to constitute a quorum (which will cause $s++$ or $s--$.) $\Delta =$ initial value of $|s|$ for all honest agents
$\beta =$ stopping value of $|s|$.
L : a constant $0.5 < L < 1$, used in the analysis. It is typically close to 1.

8.3.2 Quorum Sampling

Let f be the fraction of agents who will respond "yes" when sampled at the beginning of an iteration. Initially $f = a$. The probability (at least) a quorum q of the d samples responds "yes" is

$$\sum_{j=q}^{j=d} \binom{d}{j} f^j (1-f)^{d-j} \geq \left(f^q (1-f)^{d-q}\right) \sum_{j=q}^{j=d} \binom{d}{j}.$$

The probability is correct because sampling is with replacement. The inequality is valid as long as $f > \frac{1}{2}$.

The probability a quorum of d samples says "no" has the same formula with the roles of f and $1-f$ exchanged. However, we need an upper bound for this probability, as follows.

$$\sum_{j=q}^{j=d} \binom{d}{j} (1-f)^j f^{d-j} \leq \left((1-f)^q f^{d-q}\right) \sum_{j=q}^{j=d} \binom{d}{j}.$$

Hence, the ratio of the two probabilities satisfies

$$\frac{P(\text{"yes" quorum})}{P(\text{"no" quorum})} \geq \frac{f^q (1-f)^{d-q}}{(1-f)^q f^{d-q}} = \left(\frac{f}{1-f}\right)^{2q-d}. \tag{1}$$

Lemma 1. Formula (1) is a valid lower bound for all $0.5 < f \leq 1$.

8.3.3 Accuracy

In this analysis agents sample until $s++$ or $s--$. That only affects the number of iterations until stability, not the accuracy. Also in this analysis, all Byzantine and initially inaccurate agents always signal "no". This pessimistic assumption can only decrease the probability of an accurate outcome. We will show that nonetheless the probability is extremely close to 1.

Let the honest initially accurate agents be labeled $k = 1, 2, \ldots, an$. For all iteration counts $t = 0, 1, 2, 3, \ldots$ let $S_k(t)$ be a random variable equal to the s value of agent k after the tth iteration. Thus $S_k(0) = 5\ \forall k$. For all $t > 6$ the variables are jointly dependent (in fact pairwise dependent). Dependence arises once it is possible negative values for some values of k at time t affect the drift for all k.

We construct a different set of random variables, $SR_k(t)$, that are dominated by the $S_k(t)$ in the sense that $SR_k(t) \leq S_k(t)\ \forall k, t$. We prove the $SR_k(t)$ variables, interpreted as s values, have extremely high accuracy. By dominance, the accuracy of the $S_k(t)$ must be at least as high, giving us our desired result.

Construction of $SR_k(t)$.

$SR_k(t)$ is built in two stages. First, we define a set of variables $R_k(t)$ for which we prove high accuracy. Second, we alter $R_k(t)$ to partially mimic $S_k(t)$. The alteration retains the high accuracy bound of $R_k(t)$, and enforces the inequalities $SR_k(t) \leq S_k(t)$.

Let L satisfy $0.5 < La < a$. Typically L equals 0.95 or some other value close to 1. For each k, define $R_k(t)$ to be a random walk defined as if exactly Lan agents had nonnegative s values when they sample according to the quorum protocol From Equation 1, setting $f = La$, the ratio of probabilities of "yes" to "no" is greater than

$$\frac{P(R_k(t+1) = R_k(t) + 1)}{P(R_k(t+1) = R_k(t) - 1)} \geq = \left(\frac{La}{1 - La}\right)^{2q-d} \equiv \mathbf{r}. \tag{2}$$

The walks are jointly independent. (Their initial drift to the right is less than the $S_k(t)$'s, La rather than a. We sacrifice a bit of drift to get independence.)

Lemma 2. For the set of accurate agents $k = 1, \ldots, an$, the events that R_k is never strictly negative, $P(R_k(t) \geq 0\ \forall t)$, have equal probability and are jointly independent. The probability is bounded below by

$$\begin{aligned} P(R_k(t) \geq 0\ \forall t) &\geq 1 - \left(\frac{1}{\mathbf{r}}\right)^{\Delta+1} &(3)\\ &= 1 - \left(\frac{1 - La}{La}\right)^{(2q-d)(\Delta+1)}. &(4) \end{aligned}$$

Proof: For each $k = 1, \ldots, an$, $R_k(t)$ is a random walk independent of the other walks. The probability ρ that $R_k(t)$ ever equals $R_k(0) - 1$ satisfies the recursive equation

$$\rho = 1/(1 + \mathbf{r}) + \frac{\mathbf{r}}{1 + \mathbf{r}}\rho^2 \Rightarrow \rho = \frac{1}{\mathbf{r}}.$$

It follows that the probability $R_k(t)$ ever equals $R_k(0) - 2$ is ρ^2, because the random walk transition probabilities do not depend on time or location. Likewise, the probability $R_k(t)$ ever equals $R_k(0) - (\Delta + 1) = -1$ is $\rho^{\Delta+1}$. Take the value of $\mathbf{r}$ from equation 2 to get equation 4. ■

Let $A^S(t)$ (resp. $A^R(t), A^{SR}(t)$) be the event that *after* every iteration $m = 1, \ldots, t$, at least Lan (i.e., a fraction L) of the $S_k(m)$ (resp. $R_k(m), SR_(m)$) variables are ≥ 0, and let A^S (resp. A^R, A^{SR}) be their infinite intersection. That is, $A^S(t)$ is the event

$$|\{k : S_k(m) \geq 0\}| \geq Lan\ \forall m \leq t \text{ and } A^S = \bigcap_{t \geq 1} A^S_t.$$

The definitions for $R_k()$ and $SR_k()$ are the same with R and SR replacing S. ■

Accuracy Theorem. If $2p \leq 1 - L$ then $1 - P(A^R)$ is less than (11).

Proof: Let random variable Z be the number of honest accurate agents k for which there exists t with $S_k(t) < 0$. By formula 4, Z has binomial distribution with parameters an and

$$p \equiv \left(\frac{1-La}{La}\right)^{(2q-d)(\Delta+1)}.$$

Lemma 3 $P(Z > (1-L)an) \leq P(Z = (1-L)an)$ (as long as p is small enough compared to $1-L$ as assumed in the statement of the theorem). Compare $P(Z = (1-L)an)$[1] to $P(Z = 1+(1-L)an)$.

$$\frac{P(Z=(1-L)an)}{P(Z=1+(1-L)an)} = \frac{\binom{an}{(1-L)an}p^{(1-L)an}(1-p)^{Lan}}{\binom{an}{1+(1-L)an}p^{1+(1-L)an}(1-p)^{Lan-1}} \tag{5}$$

$$= \frac{1+(1-L)an}{Lan}\ \frac{1-p}{p} > \frac{1-L}{L}\ \frac{1-p}{p} \geq \frac{2p(1-p)}{Lp} = 2\frac{1-p}{L} > 2\frac{1-2p}{L} \geq 2. \tag{6}$$

Thus $P(Z = 1+(1-L)an)$ is less than half $P(Z = (1-L)an)$. Note: the assumption used twice in the second line is $1-L \geq 2p$. Moreover, the next probability ratio $P(Z = 2+(1-L)an)$ satisfies $\frac{P(Z=(1-L)an)}{P(Z=1+(1-L)an)} > 2$, by the binomial coefficient property $\binom{n}{k+1}^2 > \binom{n}{k}\binom{n}{k+2}$. The ratio of the binomial coefficients is smaller than in (5) while the ratio $\frac{1-p}{p}$ is the same. By the same reasoning, $P(Z = m+(1-L)an) > 2P(Z = m+1+(1-L)an)$. Hence $P(Z \geq 1+(1-L)an)/P(Z = (1-L)an) < \frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots < 1$. This proves the lemma. ■

For the final step of the theorem's proof, we estimate $P(Z = (1-L)an)$ pessimistically (i.e., we find an upper bound that is nonetheless satisfyingly small).

$$P(Z=(1-L)an) = \binom{an}{(1-L)an}p^{(1-L)an}(1-p)^{Lan} \tag{7}$$

$$< \frac{(an)^{(1-L)an}}{((1-L)an)!} \tag{8}$$

Applying Sterling's approximation,

$$\approx \left(\frac{ean}{(1-L)an}\right)^{(1-L)an} \frac{1}{\sqrt{2\pi(1-L)an}}\ p^{(1-L)an}(1-p)^{Lan} \tag{9}$$

$$< \left(\frac{ep}{(1-L)}\right)^{(1-L)an} \frac{1}{\sqrt{2\pi(1-L)an}} \tag{10}$$

$$= \left(\frac{e}{1-L}\left(\frac{1-La}{La}\right)^{(2q-d)(\Delta+1)}\right)^{(1-L)an} \frac{1}{\sqrt{2\pi(1-L)an}} \tag{11}$$

■

Note: Since $1-L > 2p$ we could replace the $\frac{e}{1-L}$ term with $\frac{e}{2}$ if we reduce the $(2q-d)(\Delta+1)$ exponent by 1 to $((2q-d)(\Delta+1)-1)$. Also, the $\frac{1}{\sqrt{2\pi(1-L)an}}$ term is negligible compared with the other term. We could replace it with $\frac{1}{2\sqrt{\pi pan}}$. But combining that with the p to the large exponent looks absurd.

[1] Technically we should take the floor to ensure integer values

Corollary The bound (11) applies to the probability of an incorrect outcome to the protocol.
Proof: There will be a clear outcome w.p. 1 because only states with $|S_k(t)| = \beta$ are not transient.

Define the set of variables $SR_k(t) : t \geq 1, 1 \leq k \leq an$ as follows: If $A^R(t)$ occurs, $SR_k(t) = R_k(t)\ \forall k$. If $A^R(t)$ does not occur,

$$SR_k(t+1) - SR_k(t) = S_k(t+1) - S_k(t). \tag{12}$$

That is, SR mimics R as long as at least La of the R variables are nonnegative. If that condition ever fails to hold, SR behaves differently, mimicking the movement of S. Observe that no matter how we specify how SR behaves if A^R does not occur, $P(A^R) = P(A^{SR})$. (Indeed, the events are identical.)

We show by induction that there exists a coupling between $SR_k(t)$ and $S_k(t)$ such that $SR_k(t) \leq S_k(t)\ \forall k, t$. The base case $t = 0$ holds because $SR_k(0) = S_k(0) = \Delta\ \forall k$. Inductively assume $SR_k(m) \leq S_k(m)\ \forall m \leq t$. Partition the state space Ω on the event $A^{SR}(t)$ and its complement. In the former case, by the inductive assumption, $A^{SR}(t) \Rightarrow A^S(t)$ and $P(SR_k(t+1) = 1 + SR_k(t)) \leq P(S_k(t+1) = 1 + S_k(t))$. Condition further on $P(S_k(t+1) = 1 + S_k(t))$. Since agents sample independently of each other, the events $S_k(t+1) = 1+S_k(t)$ are conditionally independent over $1 \leq k \leq an$, as are the events $SR_k(t+1) = 1+SR_k(t)$. Therefore the $S_k(t+1)$ and $SR_k(t+1)$ can be coupled to preserve the inequalities $SR_k(t+1) \leq S_k(t+1)$. In the event $\neg A^{SR}(t)$ (which by definition is identical to $\neg A^R(t)$), then the mimicking movement defined by (12), together with the inductive hypothesis, ensures $SR_k(t+1) \leq S_k(t+1)\ \forall k$.

Hence, $P(A^S) \geq P(A^R)$, proving the corollary. ■

8.3.4 Convergence Rate

We estimate the number of iterations until: (i) each honest agent (whether initially correct or incorrect) has confidence $\pm\beta$ with probability at least 99%; (ii) with very high probability all honest agents have confidence $\pm\beta$. Initially we assume agents sample until getting a quorum. This merely changes the estimates by a constant factor, namely, the probability that a sample (of d agents with replacement) results in a quorum (at least q are in agreement).

As usual, index the honest accurate agents $k = 1, \ldots, an$, and let $S_k(t)$ be the confidence of agent k after t iterations. From the proof of the Corollary, the probability A^S does not occur is negligible. Hereafter we assume A^S occurs.

Define a new set of variables $\overset{\infty}{S}_k(t)$ to be what $S_k(t)$ would be if $\beta = \infty$. That is, in the imaginary $\overset{\infty}{S}$ process, agents never stop sampling and changing their confidence. Since the drift is strictly positive, eventually $\overset{\infty}{S}_k(t) \geq \beta\ \forall k$ with probability 1. This makes $\overset{\infty}{S}$ easier to analyze. We don't need to worry about the pesky sample paths where some agents have $S_k(t) = -\beta$. The simple but important inequality is:

$$\overset{\infty}{S}_k(t) \geq \beta \Rightarrow |S_k(t)| = \beta. \tag{13}$$

Index the honest initially inaccurate agents $k = an+1, an+2, \ldots, (1-b)n$. We employ (1). For all agents k, the distribution of $\frac{1}{2}(\overset{\infty}{S}_k(t) - \overset{\infty}{S}_k(0) + t)$ (i.e., the number of iterations when $\overset{\infty}{S}_k(j)$ increases, $\overset{\infty}{S}_k(j+1) = 1 + \overset{\infty}{S}_k(j)^2$) dominates a binomial distribution with parameters t and

$$\frac{\left(\frac{f}{1-f}\right)^{2q-d}}{1 + \left(\frac{f}{1-f}\right)^{2q-d}}$$

where $f = La$.

[2]We use notation $\overset{\infty}{S}_k(j)++$ to denote this event, and $\overset{\infty}{S}_k(j)--$ its complement.

We want t large enough that $\overset{\infty}{S}_k(t) \geq \beta$ with high probability for each k. For now, we are not trying to bound the probability of the more stringently defined event $\{|S_k(t)| = \beta \ \ \forall k\}$.

Let z be the value of (8.3.4). Let $Z \sim \ Bin(z,t)$. The stochastic dominance stated above is equivalent to

$$P(\frac{1}{2}(\overset{\infty}{S}_k(t) - \overset{\infty}{S}_k(0) + t) \geq \tau) \geq P(Z \geq \tau).$$

For $k > an$, the initial confidence is $S_k(0) = -\Delta$. The event $\overset{\infty}{S}_k(t) \geq \beta$ is equivalent to $\frac{1}{2}(\overset{\infty}{S}_k(t) - \overset{\infty}{S}_k(0) + t)$ $\geq \frac{1}{2}(\beta + \Delta + t)$, which therefore has probability at least $P(Z \geq \frac{1}{2}(\beta + \Delta + t))$.

The binomially distributed Z has expected value $E[Z] = tz$ and variance $\sigma^2(Z) = tz(1-z)$. By the Gaussian approximation (central limit theorem), $P(Z \geq E[Z] - 3\sigma(Z)) = P(Z \geq tz - 3\sqrt{tz(1-z)} > 0.99$. Put these inequalities together to get, for $k < an$,

$$P(\overset{\infty}{S}_k(t) \geq \beta) > 0.99) \text{ if } \frac{1}{2}(\beta + \Delta + t) \leq tz - 3\sqrt{tz(1-z)}. \tag{14}$$

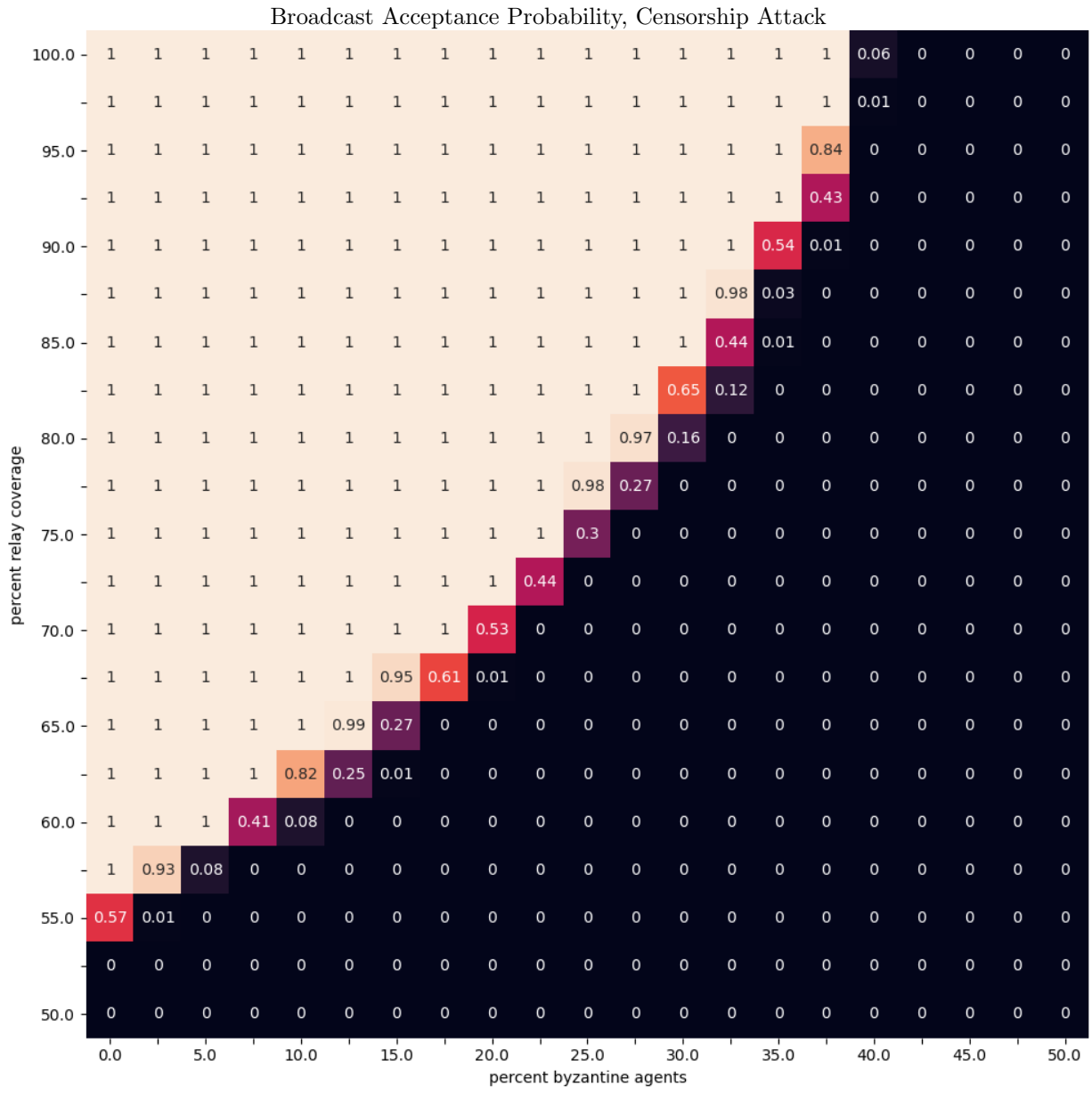
Solve for t to get

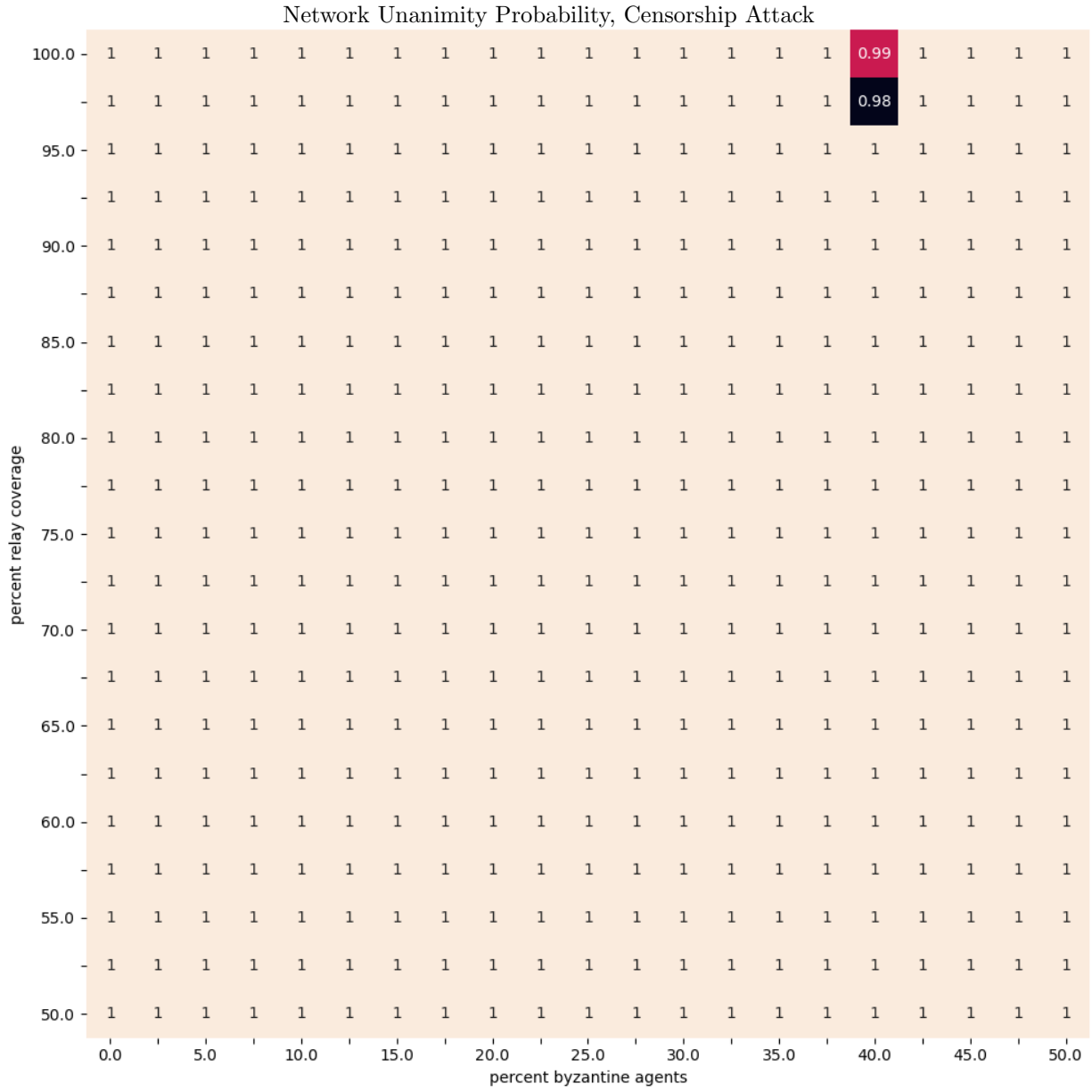
$$\begin{aligned} t &\geq \frac{\beta + \Delta + z + 9(1-z)}{2z} + \frac{1}{z}\sqrt{(z + \beta + \Delta + 9(1-z))^2 - (z + \beta + \Delta)^2} && (15) \\ &= \frac{1}{z}\left(\frac{1}{2}(\beta + \Delta + z + 9(1-z)) + 3\sqrt{9(1-z)^2 + 2(z + \beta + \Delta)(1-z)}\right) && (16) \\ &\Rightarrow \ P(\overset{\infty}{S}_k(t) \geq \beta) > 0.99 \Rightarrow P(|S_k(t)| = \beta) > 0.99). && (17) \end{aligned}$$

The same formula with $\beta + \Delta$ changed to $\beta - \Delta$ applies to the initially accurate agents $1 \leq k \leq an$.

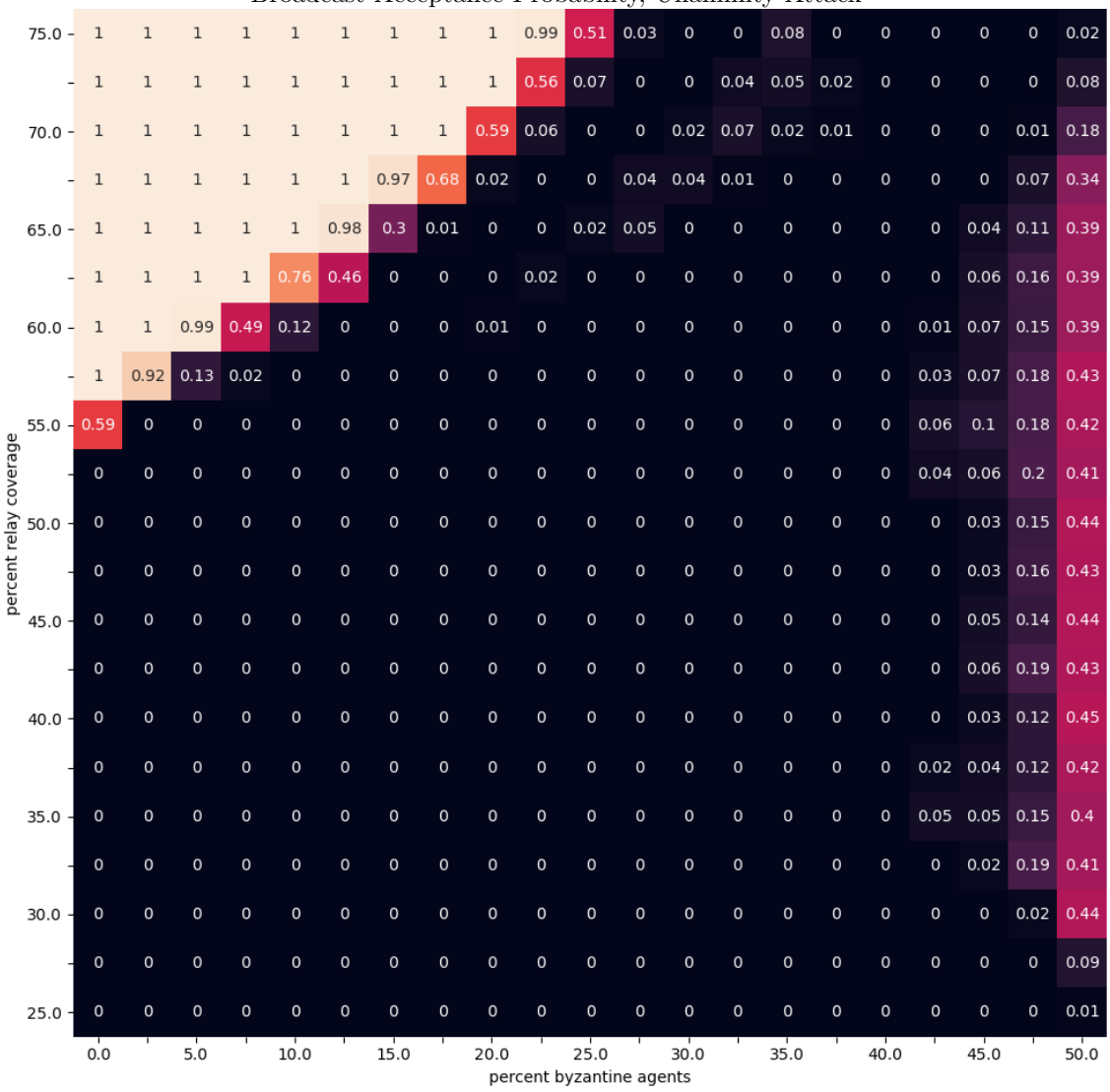
8.3.5 Simulation

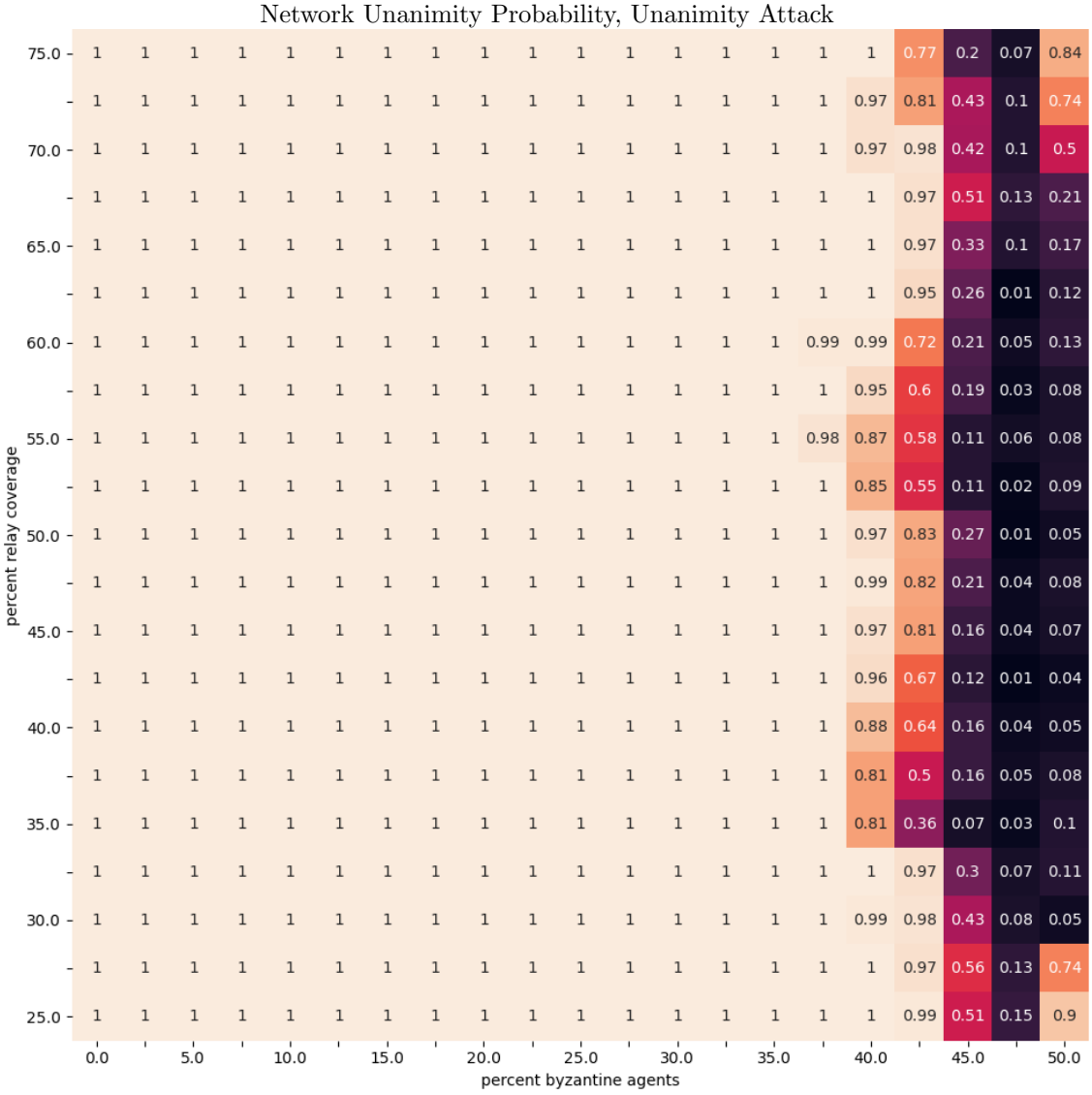
Below we simulate attacks on the epoch vote process at different levels of byzantine agents and initial relay coverage. The epoch vote parameters of the simulation are as follows: $q = 8$, $\beta = 50$, $K = 10$, $i = 5$, $L = 40$, $D = 1$, $T = 200$. The first pair of charts is a censorship attack in which byzantine agents nack all broadcasts. The second pair of charts is a unanimity attack in which byzantine agents ack or nack to keep honest nodes as divided as possible on each broadcast. The broadcast acceptance probability charts show the odds that any given broadcast will be accepted by an honest agent at the end of the epoch vote. The network unanimity probability shows the probability that all honest nodes reach the same result at the end of the epoch vote for each broadcast. Note the different on the Y axis between both paris of charts. Although these epoch vote parameters are not tuned for best possible performance, these simulated results show that the epoch vote process has both safety and liveness as long as long as byzantine agents do not exceed $\frac{1}{3}$ of the total.


Broadcast Acceptance Probability, Censorship Attack
percent relay coverage
percent byzantine agents
100.0
95.0
90.0
85.0
80.0
75.0
70.0
65.0
60.0
55.0
50.0
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
40.0
45.0
50.0


Network Unanimity Probability, Censorship Attack
percent relay coverage
percent byzantine agents
100.0
95.0
90.0
85.0
80.0
75.0
70.0
65.0
60.0
55.0
50.0
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
40.0
45.0
50.0
0.99
0.98

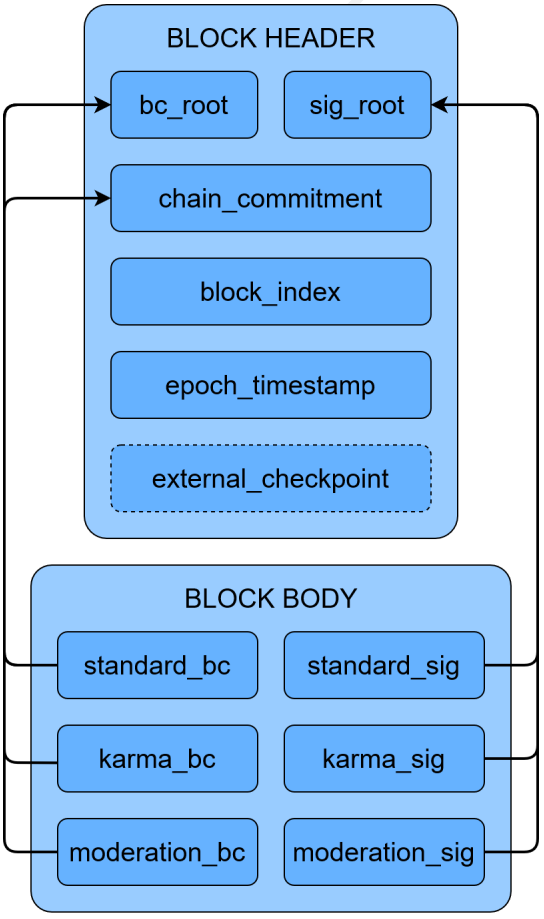
Broadcast Acceptance Probability, Unanimity Attack




Network Unanimity Probability, Unanimity Attack
percent relay coverage
75.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.77 0.2 0.07 0.84
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.97 0.81 0.43 0.1 0.74
70.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.97 0.98 0.42 0.1 0.5
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.97 0.51 0.13 0.21
65.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.97 0.33 0.1 0.17
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.95 0.26 0.01 0.12
60.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.99 0.99 0.72 0.21 0.05 0.13
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.95 0.6 0.19 0.03 0.08
55.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.98 0.87 0.58 0.11 0.06 0.08
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.85 0.55 0.11 0.02 0.09
50.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.97 0.83 0.27 0.01 0.05
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.99 0.82 0.21 0.04 0.08
45.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.97 0.81 0.16 0.04 0.07
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.96 0.67 0.12 0.01 0.04
40.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.88 0.64 0.16 0.04 0.05
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.81 0.5 0.16 0.05 0.08
35.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.81 0.36 0.07 0.03 0.1
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.97 0.3 0.07 0.11
30.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.99 0.98 0.43 0.08 0.05
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.97 0.56 0.13 0.74
25.0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.99 0.51 0.15 0.9
0.0 5.0 10.0 15.0 20.0 25.0 30.0 35.0 40.0 45.0 50.0
percent byzantine agents

8.4 Block Construction

At the end of the epoch vote, each agent constructs a block of all accepted broadcasts. The broadcasts of a block must all include the same chain commitment. This constraint is explained in section 9.2. To avoid redundancy, each broadcast has its chain commitment removed and a block-wide chain commitment is included only once in the block header. The broadcasts are split into three disjoint sets: broadcasts with moderation operators, broadcasts with like or dislike operators, and the remaining broadcasts. These sets are known as `moderation_bc`, `karma_bc`, and `standard_bc`, respectively. The broadcasts in each set are then ordered in ascending order of originating alias number, and hashed into a Merkle tree. The broadcasts in each set are concatenated and added to the block body in the following format: `standard_bc`‖`karma_bc`‖`moderation_bc`. Each Merkle root is concatenated in the same order as the broadcast sets and hashed. The resulting hash is the **broadcast root**. the signatures authorizing each broadcast are stored in separate Merkle trees with the **signature root** stored in the block header as well, sorted in the same order. The broadcast root, signature root, chain commitment, block index, and epoch timestamp are concatenated in that order and added to the block as the block header. If a block includes a checkpoint, it is appended too. The **block ID** is the hash `sha256(broadcast_root`‖`signature_root`‖`chain_commitment`‖`block_index`‖`epoch_timestamp`‖ `external_checkpoint)` [18] of the block header.



8.5 Delegation

Some nodes will not be on 24/7 and some users may not operate a node at all, instead relying on a service to access the network. In such a case, it is desirable for aliases to delegate their authority to other aliases

so that the maximum number of aliases play a role in the epoch vote and so that the delegating alias can receive the necessary chain commitments to broadcast to the network. Each alias can set another alias as its delegate. If a validator cannot reach an alias' node, it will instead attempt to reach it's delegate. This process repeats until either the validator successfully connects, a certain number of attempts fail, or a loop in delegates is encountered. If either of the latter two occur, the validator will not sample the node again for a period of time. Rather than storing this information on the blockchain, it can instead simply be gossiped between validating nodes and saved locally, since unanimous agreement is not necessary. Any validator that does not configure their delegation correctly will simply be ignored by other validators, meaning there is no incentive for malicious validators to abuse the delegation system. Delegation can be incentivized by rejecting broadcasts from aliases that do not properly delegate or validate themsleves, making it less likely for nonparticipating aliases to get their broadcasts accepted by the network. Lastly, delegations only last for a period of time before they must be recommitted. This prevents an alias from permanently delegating to a malicious validator if the owner loses their keys or simply forgets that they are delegating.

9 Proof of Engagement

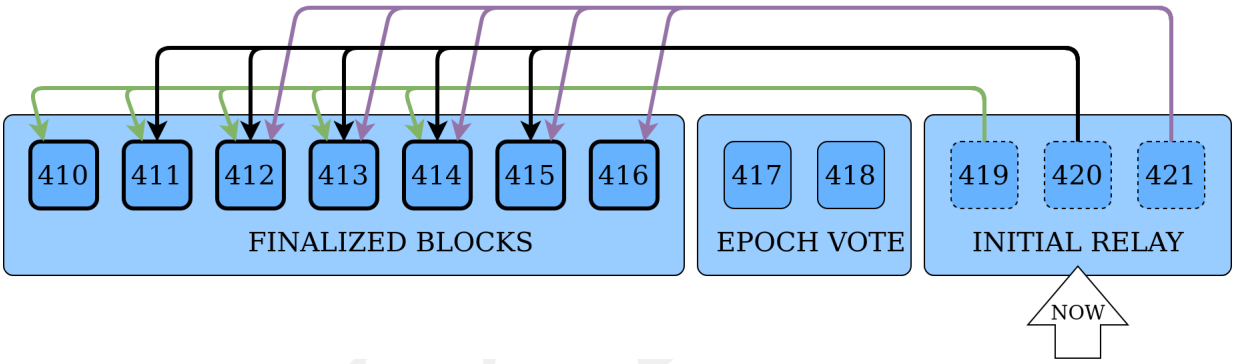
Relying only on the epoch vote means that validators can execute long range attacks and is susceptible to rare but potentially unresolvable liveness failures. To resolve these issues, proof of engagement counts the total number of aliases that commit to a particular chain. Security is guaranteed as long as an attacker does not have a majority of active aliases. To prevent short range attacks, nodes place trust in online validators in such a way that still does not allow for long run attacks. In the event of a successful attack, social recovery is made simple by all attackers identifying themselves via chain commitments.

9.1 Proof of Engagement Purpose

There are three issues with using the epoch vote system exclusively to reach consensus. First, although the validators will come to consensus on a single block with very high probability when there are not a large number of byzantine validators, the epoch vote does not provide an objective method of resolving forks. Even though the epoch vote system guarantees that there will not be a fork during normal operation, it is possible that a temporary degradation in connectivity between continents, a bug in code, or even a successful epoch vote attack could cause a fork in the epoch vote. In the unlikely event that this occurs, the system should gracefully recover without requiring social coordination for a hard fork to restore consensus. Second, the chain is at risk of long range attacks from a majority of validators that reorg a large number of blocks. Without the proof of engagement process, malicious validators could quietly remove a particular broadcast from the chain history and reconstruct all following blocks. Although malicious validators can always reconstruct historical blocks, the proof of engagement system necessitates that if any single piece of historical content is removed from a historical block, then all historical content must be removed from all blocks committing to the altered block. Additionally, proof of engagement ensures that a malicious, reconstructed chain will be rejected by nodes in favor of the canonical chain history. Third, a somewhat more philosophical point, the canonical history should ideally be decided by what a majority of users decide, rather than just those running nodes participating in the epoch vote. Typically in blockchain systems, if a majority cabal of validators decides to enact a soft fork, users are forced to go accept the coercive change. With the proof of engagement design, if even a single validator sides with the majority of users, the users' chain will be the canonical one instead of the malicious chain, even though the malicious chain has more validators.

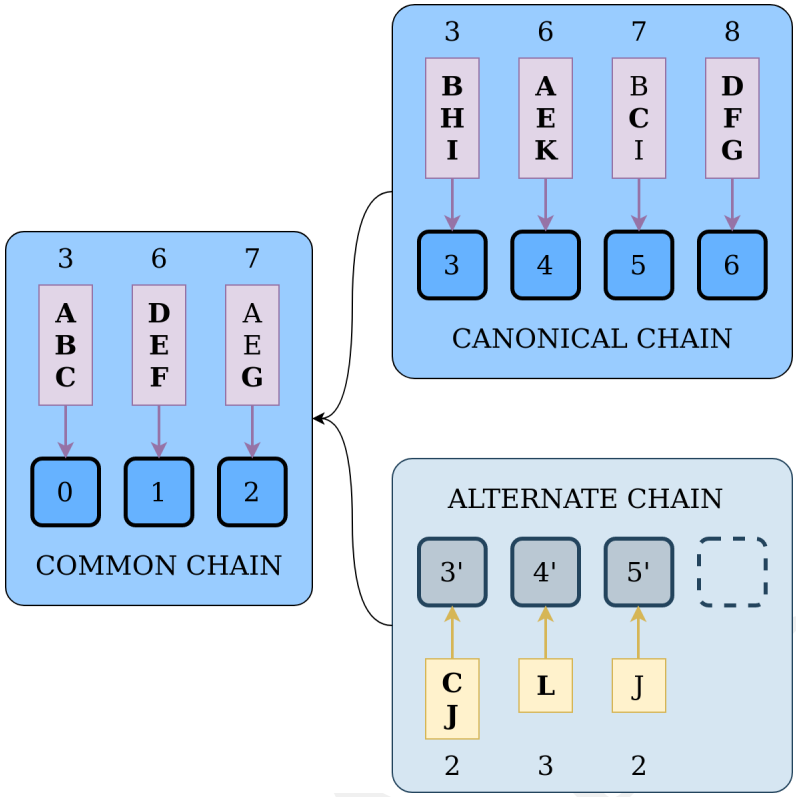
9.2 Chain Commitments

One difference between Semaphore's blockchain architecture and other blockchain systems is that each Semaphore broadcast can only be included in a particular block because the chain commitment of each broadcast is only valid in a particular block. However, a certain amount of time is required for the initial relay and epoch vote to take place. While this process is taking place, broadcasts obviously cannot commit to the outcome because the process has not yet completed. Therefore, instead of having a gap between blocks during which no broadcasts can be accepted, blocks instead do not commit to the block directly prior. Typically, in a blockchain, block n commits to block $n-1$, where n is the index of the block in the blockchain. In Semaphore, there is a delay of V blocks to allow time for the verification of each block. Therefore Semaphore block n commits to blocks $n-V-1$ through $n-2V-1$ to allow for the verification delay and commit to all finalized parts of the blockchain history. The typical blockchain design is just the special case where $V=0$. Multiple blocks are committed to so that there are not separate chains running in parallel: all blocks commit to all previous blocks except for the V blocks immediately prior. Rather than only the block header committing to prior blocks, each broadcast in the block commits to prior blocks and, crucially, all broadcasts in a block must have the same chain commitment.



9.3 Proof of Engagement Mechanism

The basic intuition behind proof of engagement is very simple: when presented with competing chains, an agent will pick the one that has been committed to by more aliases. To cause a reorg, an attacker would need to control more aliases than have committed to the canonical chain. More specifically, if an agent is comparing two competing chains, it assigns each chain a weight based on chain engagements and select as canonical the chain with the higher weight. To calculate the weight of each chain, the agent first finds the last common block that is shared by both chains. The number of unique aliases committing to each chain is the **weight** of the chain. If, however, an alias commits to both competing chains, then the alias is said to have **equivocated**. If an alias equivocates, only its last commitment is calculated. Furthermore, for each equivocation of an alias, validators in the epoch vote process will prevent the alias from broadcasting for $k \cdot 2^n$ epochs, where n is the number of times the alias has equivocated and k is the minimum timeout. After $\sum_{i=0}^{n} k \cdot 2^n$ epochs, n resets to 0. This is to prevent an alias from repeatedly equivocating between chains.



However, using only the weight calculation and naively switching to the heaviest chain immediately leaves the chain vulnerable to short range attacks. For instance if a new block gets created with 100 aliases broadcasting, an attacker with 101 aliases could immediately reorg the block. For now, we restrict analysis to agents with online nodes. Offline nodes reconnecting will be discussed afterwards in this section. Agents know that the odds of having made a mistake in the epoch vote are slim to none, so an unknown block that gets a significant number of engagements is almost certainly being made maliciously. Therefore agents implement a **minimum reorg depth rule**, which keeps an agent from reorging their chain until the other chain is of a certain depth past the last common block. If the minimum reorg depth is 100 blocks, an agent will only switch to an alternative chain if it has more engagements after 100 blocks. Implementing this rule, an attacker who seeks to cause a reorg by filling blocks with engagements must control more aliases than the number of aliases that broadcast on the honest chain for the entire time that the depth is less than the minimum reorg depth

Once more aliases than an attacker possesses commit to a chain, there is no way for the attacker to reorg the chain. However, if an attacker does successfully attack the chain, recovery from the attack is very easy. It is obvious to all agents which aliases participated in the attack, so through social coordination the attacker's aliases can be burned through a soft fork. Although such a fork would be a last resort and should never be necessary, it would be preferable to allowing the Semaphore network to fail. Additionally, the threat of such a fork being possible should disincentivize an attack from being attempted in the first place.

The process through which agents with offline nodes who are reconnecting to the network sync to the current state of the chain is more complicated because offline agents, having not participated in the epoch vote, cannot know which fork was the correct output of the epoch vote. In the happy case there will only be a single chain presented by an agent's peers, but this cannot be assumed. A trivial solution is for a reconnecting agent to download headers for unknown blocks and assign each zero weight, recalculating weight from scratch once online and only validating missed blocks once there is a clear winning chain. However, this procedure means that the node cannot fully participate until enough blocks have passed for there to be a chain longer

than the minimum reorg depth.

To make the syncing process converge faster agents can decide to put trust into delegate agents, in a process distinct from epoch vote delegation. If aliases publicly signal their support for a particular alias who acts as a delegate, offline agents can ask delegates to provide the weights for the missed blocks. An agent will use the provided weights if a majority (such as 51% or $\frac{2}{3}$) of delegates agree. Colluding delegates only have the power to execute short range attacks. Even if delegates give dishonest weights to an agent, the agent will return to the true chain once a sufficient number of aliases broadcast their commitments. Additionally for this reason, the delegation system does not need to be a part of the protocol specification. It is conceivable that two reconnecting agents see different weights on a chain in such a way that they fork onto competing chains. Once the agents are online they will calculate the weights and converge onto a single chain as the new, agreed upon block weights supercede the old, contentious block weights.

9.4 Safety and Liveness

All agents construct their own block in the epoch vote, so to add or remove content from a block maliciously, an attacker must either manipulate the epoch vote itself or convince an honest agent to switch chains after the epoch vote. Manipulating the epoch vote has already been covered so we will only consider making an honest agent switch chains. To make an honest agent switch chains, the chain must have greater sustained commitments than the current chain of the agent. For an attacker to successfully accomplish this, they would need to have control of a majority of the active aliases. In other words, controlling enough aliases to cause a reorg would require more aliases than simply manipulating the epoch vote. Safety is guaranteed because nothing may enter a block without first being accepted in the epoch vote. Liveness properties of proof of engagement are similar to proof of work in that there may be competing blocks, but with each additional block the network will likely settle on one or the other. Because each broadcast must have a valid chain commitment, if a broadcast commits to the one chain it cannot be included in the the other. Therefore all users are incentivised to commit to the chain most likely to be considered to be canonical. If an agent is forked from the canonical chain, such as by losing internet connectivity during the epoch vote, they will quickly switch back to the canonical chain due to the greater number of chain commitments it will receive.

9.5 Lower Bound Engagement Proofs

Signature validation can be slow and participating in the epoch vote requires internet connectivity. This means that "SPV" style validation [19], or in the case of Semaphore SBV validation, is desirable. To achieve this, **lower bound engagement proofs** act as a simple way of determining the weight of a chain. In proof of work, a block's weight can be checked without validating the contents of the block, but this is not the case if weight is defined by the signatures of the block. SBV nodes must download the block headers of a chain they wish to validate. Another node can construct a proof by taking the most recent broadcast from each alias and its Merkle path and submitting them to the SBV node. The SBV node can now validate only as many signatures as the total number of unique aliases in the chain. Although this would not have a significant speedup over short time frames, Over longer periods of time, where an alias is likely to have broadcasted multiple times, the number of signatures that must be checked is significantly reduced. Furthermore, the signatures may be aggregated so that only a single check must be performed. While not as easy as proof of work SPV, the work required in proof of engagement SBV is bounded so that a node can determine the weight of the canonical chain faster than validating the entire chain if necessary. Future work includes making the lower bound engagement proof a zero knowledge proof [20], which would be even faster to validate.

10 Karma System

Semaphore's karma system measures the contribution of an alias to the network and allows aliases with higher karma greater access to the blockchain when there is congestion. An attacker can only manipulate the system in proportion to the number of aliases under their control. The amount of karma that an alias must have to broadcast to the network is set by the protocol.

10.1 Blockspace Allocation

Type 2 networks have a market for blockspace because blockspace is limited, forcing transactors to bid up the price of blockspace until the quantity supplied equals the quantity demanded. Semaphore, a type 3 network, also has limited blockspace but does not allow users to bid up the price. In a type 2 network a user decides for themself what the value of getting their transaction included into a block is. In a type 2 network, users typically are submitting transactions for only their own benefit, so it is logical that the user sets the price they are willing to pay. However, broadcasts in Semaphore are for the benefit of all users that want to interact with the broadcast, not only the sender. Semaphore should seek to include the broadcasts that are considered to be the most beneficial to the network as a whole. The abstract concept of "most beneficial" is quantified with the karma system.

As aliases post broadcasts that other aliases interact with, the alias gains karma. Karma is consumed when broadcasting, with the quantity of karma set by the network. Karma slowly decays over time. Karma is not a cryptocurrency: it cannot be transferred from one user to another.

10.2 Karma Gain and Decay

A user's enjoyment of a broadcast is impossible to quantify, but there must be an objective measure to allocate blockspace. Furthermore, it should not be easy to manipulate the system to give an alias high karma when it is not deserved. Much of the complexity of designing karma systems in existing, centralized social media platforms, such as Reddit, comes from the fact that users can generate unlimited accounts. The algorithms must be kept as a tightly guarded secret or else users could optimize content for the algorithm instead of genuine enjoyment of other users. The system Semaphore must be simple so that it is not costly to execute and, of course, is entirely open as it is a part of the protocol. Semaphore's solution is to count the total number of unique aliases that have interacted with an originating alias and discount it over time. When the user wishes to broadcast, the karma cost is set by the network and subtracted from the user's total. A user's karma cannot drop below 1.

More specifically: for each alias, A, the network maintains an **interaction list** L_A, which is initially empty upon activation of A. When an alias B, unique from A, likes a broadcast or replies to a broadcast for which A is the originating alias, B is appended to the end of L_A. If B is already in L_A, then the old entry B is removed before B is appended. The karma of A, K_A, is the number of entries in L_A. Every n epochs, all karma decays for all aliases. The decay is a function of a base decay rate (D^b), proportional decay rate (D^p) and the total karma of an alias. The total karma decay for A is $D_A = min\left(\lfloor K_A \cdot D^p + D^b \rfloor, K_A - 1\right)$. After calculating D_A, the first D_A entries from L_A are removed. It is very difficult to know what values of n, D^p, and D^b are best and experimentation will be required to find the right balance. It is possible that the best solution is for the values to adjust dynamically based on the utilization of blockspace.

This system very intentionally gives all aliases the same impact on a user's karma. If aliases with higher karma could cause greater increases in karma than aliases with lower karma, then cliques of aliases could infinitely boost their own karma. Even if there was some allowance for differential impact between aliases without allowing karma to blow up, attackers can always maximize karma while honest users would not. By

making all aliases equivalent, to dishonestly get more karma than an honest user an attacker would need to control more aliases than interact with the honest user.

10.3 Global Karma Cost

Ideally, users will always be able to broadcast during periods of normal network load, but there will be times when demand to broadcast exceeds the capacity of the chain. When this occurs, users who have accrued the most karma should have the greatest ability to broadcast since their high levels of karma suggest that they have produced more value for the users of the network in the past. To allocate blockspace to users who both demand blockspace and have sufficient karma, the network sets a **karma price** P_i for each epoch i. If alias A broadcasts in epoch i, then K_A is decremented by P_i as described above. Therefore A must have $K_A \geq P_i$ to broadcast in epoch i. Each epoch can contain only up to a total of T broadcasts. To set P_i for each epoch, the protocol aims to fill each block with $t = T/2$ broadcasts. If there are more broadcasts than the $T/2$ target, the karma price will increase and vice versa by up to Δ times, where Δ is a constant set by the protocol. P_i is set based on the previous j blocks as follows:

$$P_i = max\left(\frac{\sum_{k=1}^{j} P_{i-k}}{j} \cdot \left(1 + \frac{\sum_{k=1}^{j}\left[t_{i-k} - \frac{T}{2}\right] \cdot \Delta}{j \cdot \frac{T}{2}}\right), 0\right)$$

10.4 Additional Complications

In addition to the tuning of n, D^p, and D^b, there are a variety of other changes that can be made to how blockspace is allocated. It is difficult to know the effect that they will have on users' experience of the network without experimentation, but they are enumerated below for further consideration. First, a dynamic rate limit can be implemented when the karma price is above a certain level. For instance, an if an alias wants to broadcast in a particular epoch, the alias may only do so if it has not broadcasted in $k \cdot P_i$ epochs, where k is a constant set by the protocol. Additionally, the rate limit could grow with successive broadcasts. For example an alias could be required to wait a minimum of 2^n epochs after it's n^{th} broadcast. The timeout could be reset by not broadcasting for $\sum_{i=1}^{n} i^2$ epochs. Instead of not allowing an alias to broadcast at all for 2^n epochs, there could simply be a higher karma price for the alias. For instance an alias' karma price could be $P_i \cdot (1 + 2^n)$, with the additional cost reset to 0 after not broadcasting for $\sum_{i=1}^{n} i^2$ epochs. Additionally, broadcasts could have a variable cost based on the number of bytes the broadcast consumes. For instance, if a broadcast consumes B bytes, its karma cost could be $B \cdot s$, where s is a variable value set by the protocol. Lastly, the karma gained could be a function of the amount of karma spent to interact. When the karma cost is very low or zero, there may be incentive to bribe off-chain for engagement for higher priority access to the blockchain when congestion increases. Making karma gained a function of karma spent would mean that such bribing would be significantly less beneficial because interactions gained during low congestion would be outweighed by fewer interactions during high congestion. There are undoubtedly numerous other complications to the karma mechanism that could modify the allocation of blockspace, but all would likely fall into the categories of restricting access and changing karma price.

11 Hardware Constraints

It is important that it be possible for users to run a Semaphore node. Semaphore can have larger resource requirements than a Bitcoin node, which is optimized to run on very low end hardware, because there is less incentive to violate network rules. Nonetheless, it is important that users can trustlessly access the network and participate in consensus if desired. Based on network usage estimates, bandwidth, computation, and storage will be within a typical user's reach. Bandwidth is likely to be the most significant constraint.

11.1 Broadcast Size

Each broadcast is made up of a signature, alias, chain commitment, parent length indicator, parent and the message itself. However, much of this data does not need to be stored in the blockchain. As stated earlier, the chain commitment is stored only once per block. Since BLS signatures are used, nodes can non-interactively aggregate all signatures in a block into a single, 32 byte signature [21]. Therefore, the only data that needs to be stored for each message is the parent info and the message. Rather than using the BCID for replies, replies instead point to the parent message by concatenating the block number and alias, a total of eight bytes, four for the alias and four for the block number. For the purposes of estimation, we assume that the parent data is 8 bytes on average and that the average message is 100 bytes. Both estimates are likely larger than the true values. Therefore each broadcast is 112 bytes on average. To include the like or dislike operator takes two bytes. However, multiple broadcasts can be liked simultaneously. Just like with parents, likes can use the alias and block number rather than BCID. Assuming batches of 10 likes at a time, it takes 8.6 bytes per like. If the average broadcast gets 15 likes, then on average a broadcast and its likes will consume 241 bytes.

11.2 Storage

Typically, content will see less engagement over time. It would be desirable for broadcasts that are not receiving interaction from aliases after a period of time to be removed from the storage of nodes. After a period of 3 months, for example, broadcasts that have not received any interaction for a period of time such as two weeks can purged from the blockchain. This means the broadcasts are set as **nonreferenceable**, meaning that the broadcast cannot be liked or replied to, as if it did not exist. Once a broadcast is nonreferenceable, nodes can safely delete it. The role of nodes is to reach consensus on new broadcasts; the saving of old broadcasts can be left to internet archivists. Nodes will always store old block headers, so any nonreferenceable broadcast can be referenced again using the `resurrect` operator. Therefore, if a user wants to reference an old, nonreferenceable broadcast, she may do so as long as she has the broadcast and its Merkle path saved. By allowing users to use the `resurrect` operator, the downsides of deleting old content are significantly mitigated with the only downside being that broadcaststs that resurrect old broadcasts will be significantly larger than if they could have referenced the broadcast directly.

Before scaling to all the world's communication activity, it is important to first ensure that initial users can use the network. Therefore an initial goal is to be able to handle 10 million broadcasts per day. At this level, on chain data would grow by and estimated 2.41 gb per day, or 72.3 gb per month. If broadcasts are deleted after 3 months without continued engagement, then the total disk space would be about 217 gb. Storage is therefore likely not to be a significant bottleneck initially. With 2 tb HDDs available for under $50 [22] and 2 tb SSDs, which are preferable, available for under $225 [23], 100 million broadcasts per day would likely not cause major storage issues. Ethereum GETH full nodes nodes have needed to store over 1 tb of state [24], so even with significant adoption Semaphore storage requirements would not grow significantly larger.

11.3 Bandwidth

Information, such as the signature and chain commitment, must be known by each node while the broadcast is being relayed throughout the network since it has not yet been included in a block. The signature is 32 bytes which must be broadcasted. The chain commitment, can be omitted, however. When validating the broadcast, nodes will insert their expected chain commitment. If the signature is invalid, the node will request the chain commitment from the sending peer. Therefore we can expect the average broadcast to consume 144 bytes of bandwidth. Similarly, each batch of ten likes will consume 118 bytes. The total

bandwidth of relaying broadcasts therefore is approximately 3.21 gb. Furthermore, we pessimistically assume that for each broadcast, each node will send or receive the originating alias. At twelve peers per node, this communication overhead constitutes an additional 480 mb per day. Lastly, the sending and receiving BCIDs in the epoch vote consumes bandwidth. We pessimistically assume that the acked sets of each pair of peers is only 90% similar on average. Therefore we estimate that with these parameters the epoch vote will consume 1.28 gb daily. Sketches used in the epoch vote consume a trivial amount of bandwidth once all agents have/ the union of all BCIDs. Therefore we estimate that the daily bandwidth consumption will be 4.97 gb per day, or 149 gb per month. Depending on the connectivity of a node, there may be more upload or download, but the sum should typically be constant. Lastly, these figures are rough approximations. However, even with bandwidth consumption an order of magnitude higher, the necessary capacity is still easily within the reach of a hobbyist user.

11.4 Computation

Computational constraints are difficult to estimate without benchmarking, but some basic analysis can be completed. When a node is reconnecting via a lower bound engagement proof, the necessary computation is O(1), because only a single signature from each alias must be validated. Validating blocks, both upon resyncing the chain and processing blocks in real time is O(n), where n represents the number of broadcasts. To validate a broadcast, the signature must be valid and the broadcast must not violate any protocol rules, such as a broadcast violating a block. Validating all rules of this kind is O(1), since the node must simply check if the alias is a member of a particular set. Finding the set, such as the set of aliases blocked by a particular alias, and determining membership of the set are both O(1) operations. Similarly, tracking karma can be accomplished through O(1) operations. Therefore the limiting process is signature validation. For nodes that wish to check the validity of blocks but do not participate in the epoch vote, the work of signature validation can be somewhat reduced by signature aggregation.

12 Sharding Overview

Most of this paper has been an exploration of a well defined system. This section explores areas that are less well defined but have great potential to expand the power of the Semaphore network with little additional complication. Some techniques would, in fact require no changes at all. Topic sharding allows for networks with different rules, such as topic specific chains or radically different type 3 networks to use the same set of aliases and in some cases be interoperable. Simplex sharding allows for the network as a whole to process more data than any individual node. Security and simplicity is maintained by having validators in every combination of shards. Failures in security only result in the chain accepting more broadcasts than it should, rather than causing any network breaking failures. Sidechains can commit their state to the Semaphore by proving their weight. This allows potentially very complex applications, such as metaverse apps, to trustlessly commit their history without increasing the complexity of the Semaphore protocol.

12.1 L2s Inapplicable

Type 2 networks are scaling through the use of layer 2 networks that keep unnecessary data/computation off of the layer 1 blockchain. This does not work in a type 3 network focused on social applications. Typically BTC users do not care about the transactions of others. In Semaphore, this is not the case. The purpose of using the network is to view the broadcasts of others. Therefore scaling in layers is not applicable to Semaphore the way it is to type two networks. Furthermore, the concept of an L2 does not make sense since Semaphore is not doing execution, beyond checking that signatures are valid and references are legal. This

functionality does not enable useful L2s for the Semaphore network's intended functionality. Therefore, to reduce the load on a node, sharding is a superior approach [25]. All data is available on the L1, but nodes only validate a subset themselves, relying on peers on other shards or aggregation services to read data on other shards.

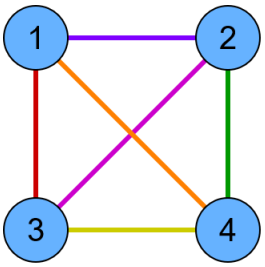
12.2 Topic Sharding

Topic sharding is arguably not true sharding. To run a topic shard is to essentially configure a node to only consider a subset of broadcasts as valid, such as broadcasts that include `#birdwatching` or are children of `Inaro-Obeld`. Nodes on the topic shard, run an independent blockchain and finalize their own blocks, but relay broadcasts to nodes outside of the shard and accept broadcasts from nodes outside the shard if they are valid according to the rules of the shard. Nodes on the topic shard also keep track of main network block headers so that outside broadcasts can be referenced if desired. For users on the shard, they have the same guarantees as users on the full network because if shard rules are broken outside of the shard, shard users neither know nor care. The downside of a topic shard, is that the shard may accept a broadcast that is not accepted by the main network. For some communities, this trade off might be acceptable; for others it may not.

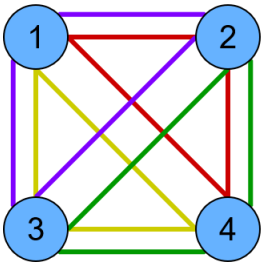
Since a topic shard is only accepting a subset of broadcasts, it is essentially a network of nodes that have soft forked the main network. However, nodes can also hard fork the rules, such as to use different rate limiting policies. Such a network cannot be considered in any way a sharding scheme since it is not compatible with the main network. However, it is worth noting that multiple, noncompatable networks can use the same aliases to manage valid identities on the network. In fact, any number of applications can reuse the same aliases, creating a universal identity set between any number of applications!

12.3 Simplex Sharding

One of the challenges in sharding designs is cross-shard communication for transactions executed between shards. Although Semaphore broadcasts are not "executed" like an Ethereum transaction, due to the network rate limits every broadcast must "execute" across all shards to make sure that an alias is not broadcasting twice within a single epoch, not waiting enough epochs between broadcasts, etc. Fortunately, cross shard communication is an easy problem to solve! Nodes simply must validate multiple shards. If there are N shards and each node validates two shards, there must be $\binom{N}{2}$ committees, one for every combination of two shards. More generally, if there are N shards and each validator manages k shards, and there is a minimum of Q validators per pair of shards, then there will be $\binom{N}{k}$ committees with a minimum of $\frac{Q}{\binom{N-2}{k-2}}$ validator per committee. Every violation of network rules can be detected by validating no more than two shards.



4 Shards
6 Committees
100 Validators per committee
600 Total Validators



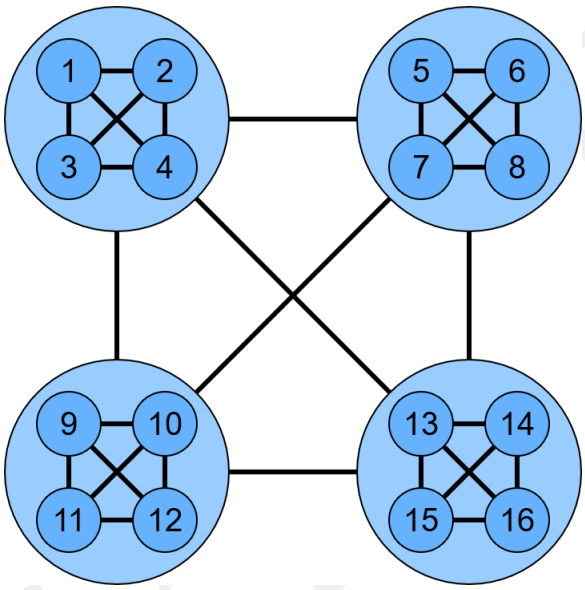
4 Shards
4 Committees
50 Validators per committee
200 Total Validators

Validators on each shard keep block headers for all shards. If an alias breaks a protocol rule in another shard, the header allows all nodes to validate a proof of the rule violation and reject broadcasts from the offending alias for a punishment period. However, since each node does not validate each shard, the number of nodes in each committee must be sufficiently large that other nodes have confidence that there are no rule violations occurring. Since Semaphore rules are not protecting money, if a rule violation does go undetected, rather than money being potentially created out of nothing, the network simply stores an additional broadcast that it should not have needed to store. Not the end of the world. Based on the number of potentially undetected excessive broadcasts, the number of shards can be increased beyond what would be feasible in a financial network, where the entire network could fail catastrophically if even a single broadcast violates a rule on any shard. Unfortunately, the number of committees grows in proportion to the square of the number of shards, since each additional shard must have a committee for each existing shard.

An alias' karma will inevitably be spread amongst each of the shards. This is good because it incentivizes or forces aliases to spread their broadcasts out among each of the shards. Since broadcasts are referenced by alias and block index, nodes can properly increase an alias' karma even if they do not know about the referenced broadcast. As more shards are added, a smaller percentage of network activity will take place in each shard. This means that, naively, the threshold for an attacker to reorg proof of engagement is lower. To resolve this, aliases are randomly assigned to a particular shard each epoch. If there are 8 shards, then every 8 epochs the order that an alias can access the shards is randomized so that in every set of 8 epochs the user can post to each shard once. This prevents a would-be attacker from putting all of their aliases into a single shard. However, if a reorg on a shard does occur, it does not effect other shards. When the new chain is produced, any broadcasts from honest aliases will still be valid as before. The attacker can only make their own broadcasts invalid.

Simplex sharding allows for more expensive creation of blocks while protecting cheap verification. In BTC, block production is very expensive but verification can be performed for trivial cost. Of course, participating in the epoch vote is very cheap because there is no mining or locked capital. However, it is possible that the network usage is large enough that either block production becomes a bottleneck, or specialized actors, possibly in data centers, produce blocks. In the latter case, scale can be achieved through making more small shards instead of few huge shards. This allows for individual users, who may not have high bandwidth, to join a single committee to check the work of the validators. If many individual users are each in a shard, even though no individuals validate the whole system any rule violation of the validators can be caught and proved to the network. If there are a few large validators who begin to break rules, proofs are submitted to

individual users in all shards and blocks from the offending validators are rejected going forward. If many validators are rejected the network is able to continue with individuals in a single committee, rather than the large validators in many comitteess, with the trade-off being that it is easier for an attacker to cause temporary forks. Through this method, the capacity on chain can be dramatically increased as long as there are enough individual users that a one out of n security assumption is sufficient for each committee. If the number of shards grows too large, the number of combinations may even be too much for validators. If this is the case, combinations can be reduced by grouping shards. For instance, if there were 16 shards then 1-4, 5-8, 9-12, and 13-16 could each be a **supershard**. For a validator, a group is treated as a single shard, but for an individual user a single pair can be validated. Since there will be many more users than validators in this scenario, the greater number of users makes up for the greater number of combinations.



12.4 Sidechains

Similar to a topic shard, there may be a community that does not care about all of the content on the network, but does not want to be a topic shard where some messages will go to the main chain and others will not. Instead, it may be beneficial to commit the state of a secondary network to the main chain. Of course, main chain users will not be able to see the content on the external network, but main chain users can have the ability to verify the validity of the state of the external network. The external network can achieve this by submitting a proof of engagement to the main chain. The external network will be producing its own block headers. The external network requires that that users additionally sign a block header. Once the header has received enough signatures, it is submitted to the main chain. Mainchain users will only know that external users have committed a hash that represents the state of the external network but do not see the actual content. However, users on the side chain can prove the state to users who come later. If a network is very small, it would be easy for bad actors to cause reorgs normally, but once the state is committed into the main chain, the side chain state cannot be reverted without reverting the main chain. The side chain may use any consensus mechanism it wishes. Only the state commitment must be formatted in a way that the main chain can interpret. Using this functionality, sidechains enable external applications to uses semaphore as a platform to confirm their state, which can likely enable interesting gaming and metaverse application.

13 Comparison to Existing Approaches

There are many other project tackling similar issues from different perspectives. Some projects simply take a different approach. Others seek to solve tangential issues and can be symbiotic with Semaphore.

13.1 DeSo (Bitclout, Formerly)

Originally, Bitclout, DeSo changed its name due to the negative association with its brand. But why would people feel negatively about this project? Let's take a look at their whitepaper [33]: In the "The First Step: Buying $DESO" section of the DeSo whitepaper, it is stated that "The supply of DeSo is capped at approximately 10.8 million, roughly half that of Bitcoin, making it naturally scarce." As if the decision of where to put a decimal point makes a coin more scarce. The section also links to instructions of how to buy $DESO and a post on diamondapp.com by "diamondhands" (verified) touting the "deflation bomb" capping the supply of Bitclout tokens. Earlier in the paper, the authors make sure not to miss an opportunity to brag about "many former-YC founders, many backed by blue-chip venture capitalists" who are building on the platform. The whitepaper also lays out a 4 step scaling roadmap: Phase 1: Proof of Stake, Phase 2: Bigger blocks, Phase 3: Warp sync, Phase 4: Sharding. The first three are not scaling improvements. Phase 1 is a consensus mechanism change and Phases 2 and 3 simply are trading off decentralization and security, respectively, for scale. It appears as of the time of writing, no phases of the road map have shipped. Additionally at launch, Bitclout deceptively displayed many top twitter accounts as if they were members of Bitclout in an attempt to make users buy their tokens.

There are significant issues with the core DeSo design as well. The DeSo blockchain, in addition to social features, supports several financial features which compete for block space. Worse, DeSo gives every user of the platform their own token, worsening the UX issues of the type 2 network even further. But the true purpose of the social tokens is be an "asset class that is tied to the reputation of an individual," meaning that "traders can make money buying and selling the ups and downs." Rather than improving the social internet, DeSo's design accentuates the internet's worst aspects for the profits of traders.

13.2 Arweave/Sia/LBRY

Each of these networks is built for the storing of data, though each works differently. In Arweave, users pay fees to have data stored in a block. Rather than being required to store the entire blockchain history, Arweave miners are more likely to find valid blocks the more historical data they store, thus creating an economic incentive to store the history. Sia users do not store data on the blockchain. Instead the blockchain stores contracts between users and storage providers. Periodically, storage providers submit a proof that the content is in fact stored. If a storage provider fails to submit a valid proof, they are penalized. Lastly LBRY is essentially a torrent indexed on the blockchain. LBRY nodes do store data, but the data is not on the blockchain itself.

These blockchains are built for a fundamentally different purpose than the Semaphore network. While Semaphore is meant to be a network effects machine, these networks are meant to be decentralized storage and do not support basic social functionality. To that end, there could be a symbiotic relationship between Semaphore and these networks. Data can be stored on one or all of the networks, pointed to by Semaphore broadcasts, and Semaphore apps automatically find where the data is available and present to the user while hiding all technical complexity. Each storage network makes different UX, security, and decentralization tradeoffs and it is unclear which strike the right balance. Currently these networks do not have significant social network effects. By serving as the social layer, Semaphore can create more demand for each of these networks without making users rely on their lackluster social functions. Furthermore, since these networks are type 2 networks, the ad hoc addition of social functionality would not lead to a good user experience.

13.3 IRC

Internet Relay Chat [26], or IRC, was never designed to be what is now considered social media but is worth analyzing anyways. Released in 1988, IRC is a simple instant messaging protocol. Users connect to servers that run boards, where users can send live messages. On its own, IRC does not support many important features of social media, such as post history from when a user was offline or usernames that can be used from multiple devices. Although users can use additional software to add some of this functionality, IRC will always be more suited to being a simple message board, which of course is what it was designed to be. There are further issues towards using IRC as social media, however. Primarily, each server is its own type 1 network. Although there are many IRC servers that exist, each is essentially its own silo, meaning IRC does not support the global conversation necessary for social media applications. In addition to users relying on server operators for all IRC functions, no verification of the information provided by server operators is possible, meaning censorship could be undetectable.

13.4 Mastodon

Mastodon [27], designed to be like a decentralized Twitter, is possibly the most well known alternative to mainstream social media since its inception in 2016. Although it is much more suited to being social media, it has many of the same downsides in regards to usability and decentralization. Mastodon can be thought of similarly to email: all users must register with a service provider who administers their interaction with the platform. This is as if Reddit users made an account with a particular subreddit rather than the platform as a whole. Although anybody can run a Mastodon instance and be a service provider for others, if the network were to grow to any real scale, it would succumb to the same fate as email: a few service providers would regulate the entire network to combat spam content. Mastodon instances are less siloed than IRC servers, with the ActivityPub protocol linking interoperable instances and applications together. However, to see content or interact with users in other instances, the service providers must decide to connect to share content. Additionally, to interact with users outside of the server, they must be addressed through the separate instance, like including an email provider's site when sending an email. Because of the content sharing requirements to interact between instances, the centralization pressures on instance operators will be even more powerful than in email. Moreover, no verification of the server operator's interaction is possible, so censorship cannot be detected. Lastly, since users must join through a particular instance, there is incentive to join an already large instance, rather than a smaller, more niche one. Even if adopted at scale, Mastodon would likely only be a minor improvement over the status quo because only a few large instance operators could operate sustainably.

13.5 Lens Protocol

Lens Protocol [28] is designed to be a "composable and decentralized social graph" that runs on the polygon network. Users on the Lens Protocol have an NFT representing their identity and create NFTs for every post and interaction. The major advantage of the Lens Protocol over the prior approaches is that it maintains composability, allowing for the development of interoperable apps. However, it has some significant drawbacks. Lens Protocol is a set of smart contracts running on Polygon, which causes three issues: centralization risk, non-sovereign blockspace, and fees. First, in order to achieve the claimed throughput of 65,000 transactions per second, Polygon is significantly more centralized than other Ethereum scaling solutions. Since Lens Protocol relies on Polygon consensus, it is vulnerable to any censorship from Polygon validators, who will have incentives beyond Lens. If the social layer of the internet was built on such a network, it would be quite easy to pay for censorship or promotion from validators off chain. Such censorship would be difficult to detect or stop for Lens Protocol users. Second, Lens has to share blockspace with other applications running

on the Polygon network. Although Polygon is supposed to prevent congestion by having high throughput, it has historically had demand for blockspace exceed supply. Due to the Jevons paradox [32], increasing capacity further would not solve the congestion issues because new blockspace induces further demand. Therefore, for a social network built on Polygon, users can expect to be crowded out at unexpected times from unrelated apps. Ultimately, social uses of are bound to be outbid by financial uses because willingness to pay is always higher for financial transactions, where a profit is expected. Additionally, having financial applications on the same chain makes validator censorship more difficult to detect and recover from because Lens Protocol users are only a small subset of total network usage. Finally, fees themselves pose a major issue. As stated previously, fees can be expected to increase at unpredictable times, but issues go deeper. Polygon is a type 2 network, but social applications are only suited to type 1 and type 3 networks. Requiring a user to pay for every Lens Protocol broadcast is a terrible user experience. This is only compounded by the fact that all user actions are minting NFTs, which has a relatively high gas cost. The issues are worse still when key management is taken into account. The only reliably safe way to store crypto funds is in cold storage, but this is not an option if fees must be paid for every broadcast. Even if the fees are very low, the keys used to unlock funds must be in a hot wallet for low friction broadcasting. This requirement makes mainstream adoption significantly more difficult because unknowledgeable users make prime targets for hackers. Self-custodying is difficult enough without requiring users to risk exposure of the private key of their funds every time they wish to like a social media post.

Lens Protocol does have some interesting fucntionality, such as allowing users to create and sell NFTs of their content. Semaphore, being primarily a network effects machine, is not meant to provide this functionality directly. However, this functionality is simply another app that can be built on Semaphore. NFTs of content, for instance, are financial instruments and are much better suited for a type 2 network such as the one on which Lens Protocol is built. A version of Lens Protocol, built to interface with Semaphore content, would likely result in a symbiotic relationship for users on both chains. Semaphore users benefit from the monetization functionality of Lens Protocol. Lens Protocol users can enjoy the reliability of a dedicated social blockchain while taking advantage of the Type 2 network for only the monetization aspects for which it is more suited.

14 Conclusion

The unique design of Semaphore allows the network to bootstrap, operate at scale, at expand its functionality over time. Central to each of these benefits is the type 3 network architecture. Aliases allow for sustainable growth by increasing the number of participants as the value of the network increases, rather than becoming just another speculative token. Additionally, by managing the aliases on an external chain, the aliases are composable with existing NFT architecture, meaning the technical barrier to entry is low. Being a type 3 network enables Semaphore to operate on its own chain with sovereign blockspace, without sacrificing security. All aliases contribute to the security of the chain simply by using the network, even if they are not running validating nodes. Additionally, since aliases must contribute to security to use the network, the cost of using an alias in an honest fashion is zero, while the cost of using aliases dishonestly to attack the network may be quite high because a successful attack can lead to the offending aliases getting burned. In other words, since the Semaphore network does not secure money, the financial downside to attempting an attack is quite high while the upside is zero. Sovereign blockspace allows for autonomous innovation without the need to convince other users of a shared chain to make changes. Semaphore users never need to compete with other chain use cases for blockspace. Blockspace is instead allocated via karma, which is gained only by contributing to the content of the network. Tuning the karma system allows for congestion to be handled smoothly so that user experience is not ruined when many people want to broadcast simultaneously. Finally, the alias system makes expanding the network in scale and functionality via parallel chains significantly more

achievable. Aliases allow for sharding to be done in a secure fashion, increasing the scale of the network. Additionally, since different networks can utilize the set of aliases, it is possible to expand the functionality of the network without needing to change the base protocol. Further innovation is possible through an ecosystem of apps that are free to experiment. Apps built on Semaphore are inherently composable, allowing for a better user experience, aligned incentives between app developers and their users, freer innovation without the need to capture network effects. For all these reasons, Semaphore is the best positioned network to be a true decentralized social layer for the internet.

15 Bibliography

References

[1] https://www.pewresearch.org/fact-tank/2020/10/15/64-of-americans-say-social-media-have-a-mostly-negative-effect-on-the-way-things-are-going-in-the-u-s-today/

[2] https://bitcoin.org/en/bitcoin-core/features/requirements

[3] https://bitinfocharts.com/comparison/bitcoin-transactionfees.html#1y

[4] https://datatracker.ietf.org/doc/html/rfc5321

[5] https://www.statista.com/statistics/547520/e-mail-provider-ranking-consumer-usa

[6] http://www.hashcash.org/

[7] https://ethereum.org/en/developers/docs/evm/

[8] https://eips.ethereum.org/EIPS/eip-721

[9] https://www.investopedia.com/terms/v/veblen-good.asp

[10] https://docs.ens.domains/

[11] https://en.wikipedia.org/wiki/Metcalfe%27s_law

[12] https://www.iacr.org/archive/asiacrypt2001/22480516.pdf

[13] https://www.lifewire.com/what-is-a-subreddit-5271885

[14] https://en.bitcoin.it/wiki/Multi-signature

[15] https://wondernetwork.com/pings

[16] https://www.avalabs.org/whitepapers

[17] http://vldb.org/pvldb/vol14/p458-gong.pdf

[18] https://en.wikipedia.org/wiki/SHA-2

[19] https://electrum.readthedocs.io/en/latest/spv.html

[20] https://en.wikipedia.org/wiki/Zero-knowledge_proof

[21] https://crypto.stanford.edu/ dabo/pubs/papers/BLSmultisig.html

[22] https://pcpartpicker.com/products/internal-hard-drive/#A=2000000000000,2048000000000&f=2

[23] https://pcpartpicker.com/products/internal-hard-drive/#t=0&A=2000000000000,2048000000000

[24] https://etherscan.io/chartsync/chaindefault

[25] https://vitalik.ca/general/2021/04/07/sharding.html

[26] https://en.wikipedia.org/wiki/Internet_Relay_Chat

[27] https://en.wikipedia.org/wiki/Mastodon_(software)

[28] https://mirror.xyz/lensprotocol.eth

[29] https://www.arweave.org

[30] https://sia.tech

[31] https://lbry.com

[32] https://en.wikipedia.org/wiki/Jevons_paradox

[33] https://docs.deso.org/about-deso-chain/readme