Optimistic Message Dissemination

Chen-Da Liu-Zhang[1], Christian Matt[2], and Søren Eller Thomsen*[3]

[1]Lucerne University of Applied Sciences and Arts & Web3 Foundation, Zug, Switzerland
chen-da.liuzhang@hslu.ch
[2]Primev, Steinhausen, Switzerland
christian@primev.xyz
[3]Partisia, Aarhus, Denmark
soren.eller.thomsen@partisia.com

August 1, 2025

Abstract

Message dissemination is a fundamental building block in distributed systems and guarantees that any message sent eventually reaches all parties. State of the art provably secure protocols for disseminating messages have a per-party communication complexity that is linear in the inverse of the fraction of parties that are guaranteed to be honest in the worst case. Unfortunately, this per-party communication complexity arises even in cases where the actual fraction of parties that behave honestly is close to 1. In this paper, we propose an optimistic message dissemination protocol that adopts to the actual conditions in which it is deployed, with optimal worst-case per-party communication complexity. Our protocol cuts the complexity of prior provably secure protocols for 49% worst-case corruption almost in half under optimistic conditions and allows practitioners to combine efficient heuristics with secure fallback mechanisms.

*The work was partly done while the author was at The Alexandra Institute.

Contents

1 Introduction

1.1 Motivation

A basic task in distributed systems is to disseminate a message to all parties in the system. This is often done using a flooding protocol where a party sends the message to all its neighbors, and the neighbors in turn send the message to all their neighbors, and so on. Due to the inherent redundancy in this process, flooding protocols are robust to message loss and random failures of parties. Practical implementations of flooding protocols often optimize the dissemination latency using several heuristics, e.g., preferring to send messages to parties that are geographically closer and have lower latency [LPR10, TM23]. Such heuristic protocols, however, cannot be secure in the presence of Byzantine corruptions: For example, it is possible that all parties geographically close to some honest party are corrupted and eclipse the honest party from the network, even though the corruption fraction in the total network is small.

On the other hand, there are provably secure flooding protocols that guarantee the delivery of messages to all honest parties after a bounded number of steps as long as the overall corruption fraction is below a predetermined threshold [MNT22, LZMM+22, LZMT24]. The inherent downside of these protocols is that they have a higher per-party communication complexity and do not allow for heuristic optimizations. Furthermore, the per-party communication complexity of these protocols is linear in the inverse of the fraction of parties guaranteed to be honest (which was shown to be optimal in [LZMT24]), where the threshold of honest parties needs to be set a priori. This communication complexity arises even when the actual fraction of parties that behave honestly is close to 1. When deploying this type of protocols, the engineers setting the parameters are thus left choosing between a protocol that *always* is secure but *most of the time* has an excessive bandwidth usage, or choosing a protocol that is secure *most of time* and *always* has a low bandwidth.

1.2 Contributions

Consider two thresholds $\gamma_{\text{BC}} \geq \gamma_{\text{WC}}$. We ask whether there is a protocol which is secure as long as at least a γ_{WC} fraction of parties behave honestly, and achieves a strictly better per-party communication complexity when the actual number of honest parties is at least γ_{BC}:

> *Is there flooding protocol that has optimal worst-case per-party communication complexity with at least γ_{WC} honesty-fraction, and a strictly better complexity in the optimistic case with γ_{BC} honesty-fraction?*

We answer this question affirmatively by proposing a flooding protocol with two distinct features: First, our protocol has an optimistic path that is efficient when the actual fraction of honest parties is high. If the actual corruption fraction is high, our protocol employs a fall-back mechanism that guarantees the delivery of messages to all honest parties in this worst-case scenario. Secondly, the optimistic path of our protocol can be instantiated with an arbitrary flooding protocol, including those that are optimized for practical deployment. This allows practitioners to use their favorite flooding protocol together with a secure fallback mechanism. That way, we achieve the best of both worlds: Protocols that take into account practical peculiarities such as physical distance, and provably secure protocols that ensure reliable message delivery even against a Byzantine worst-case adversary.

Applicability of results. Our constructions are built in a black-box manner from existing flooding protocols, and impose no additional assumptions on the fraction of honest parties beyond what these underlying protocols require. Additionally, we assume that all parties have access to a PKI infrastructure and access to a randomness beacon, both of which are readily available in most blockchain systems. As our protocols rely on splitting messages into multiple shares and on cryptographic techniques to guarantee the validity of these shares, our constructions are not suitable for disseminating very short messages. Hence, the protocols are better suited for disseminating blocks than individual transactions.

We assume a synchronous network with point-to-point channels between all parties. The protocol as we describe it requires this synchrony (i.e., knowledge of an upper bound on the delivery time of the point-to-point channels) to guarantee that all honest parties receive all messages in the worst case. The protocol can be slightly modified to guarantee delivery to all honest parties in all cases without synchrony, and only require synchrony to achieve better efficiency in the optimistic case at the expense of slightly more communication in the optimistic case. The protocol can thus be adapted to best suit the synchrony assumption and delivery requirements of the application it is used in.

Warm-up: Asymptotically optimal flooding. As a first step, we present a simple protocol PushPullFlood that achieves asymptotically optimal per-party communication (but is not optimistic). Instead of only "pushing" messages to all parties, it additionally allows parties to "pull" messages from other parties that have already received the message. We will use this mechanism also in our optimistic protocol.

Theorem 1 (PushPullFlood (informal))**.** *Let Π_{Flood} be a flooding protocol and let $\Pi_{FracFlood}$ be a flooding protocol that delivers to only a constant fraction of the honest parties. Then, PushPullFlood$(\Pi_{Flood}, \Pi_{FracFlood})$ makes black-box use of both protocols and is a secure flooding protocol with a per-party communication complexity for sending an l-bit message with security parameter κ proportional to that of $\Pi_{FracFlood}$ distributing l-bit messages plus Π_{Flood} distributing κ-bit messages.*

By instantiating Π_{Flood} and $\Pi_{\text{FracFlood}}$ with appropriate protocols from [LZMM+22] and [LZMT24] respectively, it follows that PushPullFlood can provide flooding with asymptotically optimal per-party communication.

Asymptotically optimal optimistic flooding. We then strengthen the result above and present the first flooding protocol OptimisticFlood that is asymptotically-optimal in the worst-case, but also has improved efficiency in the optimistic case.

Below we use γ_{ACTUAL} to quantify the actual fraction of parties behaving honestly and state an informal version of the theorem we show for OptimisticFlood.

Theorem 2 (OptimisticFlood (informal))**.** *Let $\gamma_{BC} \geq \gamma_{WC} \geq 0$. Further let Π_{BC} be a flooding protocol secure for the optimistic case $\gamma_{\text{ACTUAL}} \geq \gamma_{BC}$ and let Π_{WC} be a flooding protocol secure for the worst-case $\gamma_{\text{ACTUAL}} \geq \gamma_{WC}$. Then OptimisticFlood$(\Pi_{BC}, \Pi_{WC})$ is a secure flooding protocol assuming a γ_{WC} fraction of honesty, with per-party communication complexity and delivery guarantees roughly equal to that of Π_{BC} in the optimistic case, and the sum of both Π_{BC} and Π_{WC} with small overhead in the worst-case.*

A natural choice is to let both Π_{BC} and Π_{WC} be instantiated with asymptotically optimal flooding protocols with a per-party communication complexity of $O(l \cdot \gamma^{-1})$ such as the one from [LZMT24] (where it was also shown that this is a lower bound). By instantiating the corresponding honesty-fraction parameters for the best case to be close to 1, e.g., $\gamma_{\text{BC}} = 0.95$, our protocol shaves off almost a factor of γ_{WC}^{-1} from the communication complexity in the optimistic case. In particular, if $\gamma_{\text{WC}} = 0.5$, it cuts the per-party communication complexity almost in half in the optimistic case compared to using only the worst-case protocol.

Further note that one can also instantiate the optimistic good-case protocol with protocols that do not provide *any* guarantees for the safety of the overall protocol to be guaranteed. In particular, this means that algorithms not designed for Byzantine adversaries aiming to build for example minimum spanning trees such as the Plumtree algorithm [LPR10] (as used in Ethereum [TM23]) may be used optimistically while still having provable worst-case fallback guarantees if the network is under attack.

1.3 Technical Overview

Overview of PushPullFlood. At a high level, the protocol PushPullFlood works by first "pushing" a message to a large fraction of the parties and then allowing the remaining parties that may not have received the message to "pull" the message from those that have received it, similarly to the seminal work of [DGH+87]. The protocol is build modularly by being parametric in two sub protocols: 1) a flooding protocol that must ensure to push a small notification to all parties and 2) a *fractional flooding protocol* that must guarantee to push the message to at least a constant fraction of the honest parties. Combining these, our construction relies on three key techniques to ensure the efficiency of the pull phase:

1. We use a verifiable random function (VRF) to let a party prove that they are allowed to pull the message from another party, similar to [CKMR22]. That is, the pulling party will evaluate the VRF to obtain a seed and use this seed to select whom they will pull from. A party receiving such pull requests will then verify that this seed indeed was the output of the VRF to confirm the validity of the pull request. Thereby, we ensure that honest parties do not need to answer an excessive amount of malicious pull requests while still ensuring that all honest pull requests will be answered. To prevent malicious parties from choosing their VRF keys in such a way that many of them can pull from the same honest party, the VRF is evaluated on an unpredictable value obtained from a randomness beacon, which is assumed to be available to all parties.

2. We split the message into multiple shares using erasure correcting codes while ensuring that a constant fraction of these is sufficient to reconstruct the message, similar to [LZMT24]. By letting the answer to a pull request be such individual share, we ensure that honest parties can send sufficiently many pull requests to be certain to talk to a fraction of parties answering such pull request honestly, without allowing an adversary to exploit this to induce excessive communication.

3. We accompany such shares with membership proofs for a cryptographic accumulator similar to [LZMT24]. This ensures that honest parties are able to recognize which shares belong together, allowing honest parties to reconstruct efficiently.

Together, these techniques ensure that the per-party communication complexity of the pulling phase is asymptotically optimal and thereby improves the efficiency of previous protocols with a similar design. Note that while a certain message length is required for the protocol to be asymptotically optimal, the usage of erasure correcting codes and cryptographic accumulators was demonstrated to be concretely advantageous in terms of per-party communication complexity for a push-based protocol with messages of a length as small as 2 kilobytes in [LZMT24]. In addition to the cryptographic overhead, it must make sense to first send a small notification instead of the full message, so messages should be substantially larger than such a notification. Overall, the protocol is well suited for disseminating blocks in a blockchain system, where each block contains many transactions.

Remark 1. The use of the VRF and the randomness beacon prevent denial-of-service attacks. Often, they are dealt with on the network level by rate-limiting, blocking IP addresses, etc. There are two reasons why such methods are less effective in our setting: First, an adversary may control a large number of nodes, so even a single pull request from every malicious node could overwhelm an honest party. Secondly, a malicious pull request not only incurs incoming traffic to the targeted party, but also forces the party to actively answer the request, leading to additional outgoing communication. Therefore, such attacks are more damaging to our protocols than, e.g., in pure push-based flooding protocol. Nevertheless, if denial-of-service attacks are not a concern, e.g., in a permissioned setting with a limited number of nodes, our protocols can be simplified by omitting the usage of VRFs.

While the idea of using a push-pull based approach is not new [DGH+87, LPR10], to the best of the authors knowledge, this is the first time a provably secure construction with a formal security proof is presented. Additionally, it is the first time erasure correcting codes are used to obtain asymptotically optimal communication complexity in a protocol utilizing pulling.

Overview of OptimisticFlood. We present the first optimistic flooding protocol OptimisticFlood. The protocol is also build modularly and takes two protocols as parameters: 1) A flooding protocol used for the optimistic case and 2) a flooding protocol guaranteed to work even in the worst-case. Using these two, the protocol runs in three phases:

Phase 1: The message is sent using the optimistic case flooding protocol.

Phase 2: The sender estimates whether the message has been received by sufficiently many honest parties by asking a committee of parties whether they have received it.

Phase 3: Depending on the conclusion of Phase 2, one of the following steps is taken:

- If it is concluded that sufficiently many honest parties have received the message, any party that did not receive it is allowed to pull the message from some random peers.
- Otherwise, the sender defaults back to sending the message using the worst-case protocol.

If the protocol is executed in a setting fulfilling the optimistic conditions, then the optimistic case protocol ensures delivery to all honest parties and we set the parameters such that the adversary cannot force defaulting back to the worst-case protocol. Hence, the adversary can only induce additional complexity by pulling, which is ensured to be minimal using the same techniques as for the pull-phase of PushPullFlood.

On the other hand, if the protocol is executed in a setting not fulfilling the optimistic conditions, we do not obtain any guarantees from the optimistic case flooding protocol about how many parties receive the message. This is not a problem if the sender in Phase 3 defaults back to the worst-case protocol. However, an adversary may try to convince the sender that the optimistic protocol succeeded even though it did not. We solve this by carefully tweaking the parameters of the protocol to ensure that if the sender concludes that the protocol succeeded, then it is guaranteed that at least a fraction of the honest parties have received the message. By setting the parameters of the erasure correcting codes correspondingly, we ensure that the remaining honest parties receive the message when pulling.

1.4 Related Work

There is an extensive line of work on message dissemination and flooding protocols. While classic epidemic algorithms and gossip protocols [DGH+87, FPRU90, KSSV00, KMG03, DF11] focused mainly on the crash failure setting, a recent line of work [MNT22, LZMM+22, LZMT24, CKMR22] introduced flooding protocols that are resilient against byzantine adversaries. Such protocols follow graph-theoretic techniques such as [KMG03], relying on the fact that the graph induced by the neighbor selection procedure among honest parties remains connected. Most recently [LZMT24] showed that worst-case per-party communication complexity of a flooding protocol is lower-bounded by $\Omega(l \cdot \gamma_{\textsc{Actual}}^{-1})$ where l denotes the length of the message and $\gamma_{\textsc{Actual}}$ denotes the fraction of parties remaining honest. The same work presents a protocol achieving $O(l \cdot \gamma_{\textsc{wc}}^{-1})$ worst-case per-party communication complexity where $\gamma_{\textsc{wc}}$ denotes the worst-case fraction of parties remaining honest and thereby almost matches the lower-bound.

When the protocol PushPullFlood, presented in this paper, is instantiated with suitable parameters, it matches that worst-case bound. Further, the protocol OptimisticFlood instantiated with suitable parameters has a per-party communication complexity of only $O(l \cdot \gamma_{\textsc{bc}}^{-1})$ when $\gamma_{\textsc{Actual}} \geq \gamma_{\textsc{bc}}$ while asymptotically matching the protocol of [LZMT24] in the worst-case (i.e., if $\gamma_{\textsc{bc}} > \gamma_{\textsc{Actual}} \geq \gamma_{\textsc{wc}}$). Thereby, we improve upon the state of the art for provably secure protocols when the actual fraction of honest parties is higher than in the worst-case.

In contrast to this, other lines of work that target efficiency by following heuristic approaches to minimize per-party communication complexity and latency [FOA16, RT19, VT19]. A detailed overview of existing protocols can be found in [LZMT24].

To the best of our knowledge, current flooding protocols secure under Byzantine corruptions focus on the worst-case performance and do not explicitly attempt to improve the efficiency under optimistic conditions. Nevertheless, there is an extensive literature on optimistic protocols for agreement primitives.

Optimistic latency. The work of Abraham, Nayak, Ren and Xiang [ANRX21] considers Byzantine fault-tolerant broadcast and optimizes the *good-case* latency, measured as the number of rounds for all honest parties to commit when the designated broadcaster is honest.

Another traditional line of work investigates protocols that have a number of rounds proportional to the actual number of corruptions f, rather than a known upper bound on the number of corruptions t. In this case, it is known that deterministic broadcast solutions have $\min\{f+2, t+1\}$ rounds of communication [DS83, DRS90]. A long line of works focused on feasibility results, including protocols without setup [DRS82, Rei85, TPS87, BGP92, DRS90, Coa93, GM98, AD15], or protocols with a setup for cryptographic (pseudo-)signatures [PT84, LN24, DKLZ24]. A different line of work optimizes *the delay* of each round (rather than the number of rounds), by making progress as fast as the actual network delay in an optimistic case when the number of corruptions is small [PS18, LZLM+20].

Optimistic communication. The work [CKS23] considered protocols with optimistic communication $O(nf)$ for byzantine agreement, improving upon traditional protocols that achieved $O(nt)$ communication.

2 Model and Preliminaries

2.1 Model

We assume a synchronous network, i.e., that all parties are connected via point-to-point channels which guarantee delivery within a known time $\Delta_{\textsc{Channel}}$. We additionally assume that the actual fraction of honest parties $\gamma_{\textsc{Actual}}$ is at least some worst-case bound on the number of honest parties $\gamma_{\textsc{wc}} \in (0, 1]$. We assume that all parties have access to a PKI (public-key infrastructure). That is, all parties p_i have a public key $\mathtt{pk}_i$ and a secret key $\mathtt{sk}_i$ where the former is known by

Table 1: Overview of commonly used symbols.

Symbol	Meaning	Symbol	Meaning
$\mathcal{P}$	Set of all parties.	γ_{BC}, γ_{WC}	Best/worst case honest parties fraction.
n	Number of parties, $n = \lvert\mathcal{P}\rvert$.	c, T	Committee size and complaint threshold.
κ	Security parameter.	k	Number of received complaints.
l	Message length in bits.	η	Bound on the number of pulling parties.
Δ	Delivery time upper bound.	ζ	Erasure coding scheme.
ψ	Randomness beacon value.	μ, τ	Number of shares and tolerated erasures.
$\mathtt{pk}$, $\mathtt{sk}$	Public and secret key.	s	Share of erasure coding scheme.
r, π^{vrf}	Output and proof of VRF.	α	Cryptographic accumulation scheme.
γ_{ACTUAL}	Actual honesty fraction.	z, π^{acc}	Accumulated value and proof.

all other parties. Additionally, we assume that all parties have access to a randomness beacon, which periodically generates unpredictable random values. These values are updated at regular intervals, referred to as *epochs*, and are accessible to all parties for both the current and past epochs. Such randomness beacons are used in many blockchain protocols, e.g., for leader election in proof-of-stake protocols [DGKR18, Edg25].

Remark 2. When one of our protocols is used to disseminate messages in a blockchain network, the functioning of the blockchain and consequently the randomness beacon and progressing epochs rely on message delivery of the flooding protocol. Since our protocols assume such a randomness beacon with access to epochs, we need to avoid circular dependencies. One way to achieve this is to upgrade an already functioning blockchain with a traditional flooding protocol to use our optimistic flooding protocol: In that case, the randomness beacon is already working and the current epoch is known. Our protocols can thus be used and are guaranteed to reliably deliver messages in the current epoch. This in turn guarantees the blockchain to progress until the next epoch, and so on.

Notation. Let $\mathtt{H}\colon \{0,1\}^* \to \{0,1\}^\kappa$ denote a collision-resistant hash-function and let $\{\!\{a, a, b\}\!\}$ denote a multiset containing the elements a, a, and b. In Table 1, we summarize the notation used in this paper. Note that several symbols are used in the context of our protocols and their meaning will become clear later.

2.2 Primitives

Below we define properties of flooding protocols and introduce basic primitives our protocols will use and briefly discuss how they can be instantiated.

Flooding. We use a property-based definition of flooding as it was shown in [LZMT24] that security w.r.t. this definition implies a secure implementation of the flooding functionality in the UC framework [Can20]. However, unlike previous property based definitions, we include the fraction of parties that must remain honest for the delivery guarantee to apply in the definition. This allows us to capture optimistic protocols that perform better when the actual fraction of honest parties is high.

Definition 1 (Flooding)**.** A *flooding protocol* is a protocol Π executed by parties $\mathcal{P}$, where each party $p \in \mathcal{P}$ can input a message at any time, and as a consequence, parties may get a message as output.

Definition 2 ((γ, Δ)-delivery)**.** We say that a flooding protocol Π has (γ, Δ)-*delivery* for $\gamma \in [0, 1]$ and $\Delta > 0$ if the following holds: When an *honest* party inputs a message m at time t and $\gamma_{\textsc{Actual}} \geq \gamma$, then all other honest parties output m by time $t + \Delta$, except with probability negligible in the security parameter κ.

Additionally, we define the key metric for flooding networks most relevant for this paper namely *per-party communication complexity.*

Definition 3 (Per-Party Communication Complexity)**.** Let Π be a flooding protocol and let $l \in \mathbb{N}$ be the bit-length of a message m. We say that the *per-party communication complexity* of the protocol Π to send a message of length l is bounded by X if for any adversary, the probability that there is an honest party sending more than X bits as a consequence of an honest sender having input m is negligible in the security parameter κ. We write $\texttt{PPCC}(\Pi, l) \leq X$ to denote this.

Note that the delivery time parameter Δ is a crucial metric for flooding protocols. To keep the per-party communication asymptotically optimal and in particular independent of the number of parties n, the delivery time must grow at least logarithmically with n. Our protocols, however, introduce only a small constant overhead beyond the latency of the underlying protocols we use in a black-box manner.

Verifiable Random Function (VRF). In our constructions, we will use a VRF to prevent corrupt parties creating excess network traffic. Below, we define an abstraction of VRF as a pair of two algorithms with the standard properties defined in [GRPV23].

Definition 4 (Verifiable Random Function)**.** A pair $\texttt{vrf}$ is a VRF if it consists of the following two algorithms:

- $\texttt{vrf}.\mathsf{Eval}$: An evaluation algorithm that takes an input $i \in \{0,1\}^*$ and a secret key $\texttt{sk}$ as parameters and outputs an output r and proof π^{vrf}.
- $\texttt{vrf}.\mathsf{Verify}$: A verification algorithm that takes an input i, an output r, a proof π^{vrf}, and a public key $\texttt{pk}$ as parameters and outputs a boolean value $b \in \{\bot, \top\}$.

With the following properties:

Full uniqueness: An adversary cannot find a public key $\texttt{pk}$, an input i, two different outputs $r_1 \neq r_2$, and proofs π_1^{vrf}, π_2^{vrf} s.t. $\texttt{vrf}.\mathsf{Verify}(i, r_1, \pi_1^{\mathsf{vrf}}) = \top$ and $\texttt{vrf}.\mathsf{Verify}(i, r_2, \pi_2^{\mathsf{vrf}}) = \top$.

Full pseudorandomness: An adversary not knowing a secret key $\texttt{sk}$ cannot distinguish the output value r of $\texttt{vrf}.\mathsf{Eval}(i, \texttt{sk})$ (for any input i chosen by the adversary) from a value drawn uniformly at random without the proof π^{vrf}.

For simplicity, we will assume all these properties to hold throughout all our executions and disregard the negligible probability that they do not hold. Note that there exist several simple implementation of such a VRF [GRPV23].

Erasure correcting code scheme. Our protocols will make use of erasure correcting codes. Below, we recap the definition given in [LZMT24].

Definition 5 (Erasure Correcting Code Scheme)**.** Let $\mu \in \mathbb{N}$ be the number of shares, and let $\tau \in \mathbb{N}$ be the number of erasures that are to be tolerated. A pair of algorithms ζ is a (μ, τ)-erasure-correcting-code-scheme (abbreviated (μ, τ)-ECCS) if it consists of two deterministic algorithms:

- $\zeta.\mathsf{Enc}$: An encoding algorithm that takes a message $m \in \{0,1\}^*$ and produces a sequence of shares $s_1, \dots, s_\mu$.
- $\zeta.\mathsf{Dec}$: A decoding algorithm that if a sequence of shares $s'_1, \dots, s'_\mu$ s.t. it holds for at least $\mu - \tau$ of them that $s'_i = s_i$ and for the remaining $s'_i = \bot$ is input, then the original message m is returned.

We will use the notation $\zeta.\mathtt{ShareSize}(l)$ for a function that bounds the size of each share when a message of length l is encoded.

Note that we here assume the algorithms to be deterministic, which we will exploit in our constructions. While this is not the case for all erasure correcting codes, e.g., Reed-Solomon codes [RS60] are deterministic. Using Reed-Solomon codes with the same encoding of messages as in [LZMT24], we obtain $\zeta.\mathtt{ShareSize}(l) = O\left(\frac{l}{\mu-\tau}\right)$.

Weak cryptographic accumulator. In our protocols, we will make use of a weak form of cryptographic accumulators. Below we recap the definition from [LZMT24].

Definition 6 (Weak Static Cryptographic Accumulation Scheme)**.** A pair of algorithms α is a *weak static cryptographic accumulation scheme* (abbreviated WSCAS) if it consists of two deterministic algorithms:

- $\alpha.\mathsf{Accumulate}(\{m_1, \dots, m_\lambda\})$: An algorithm for accumulating a set of values $\{m_1, \dots, m_\lambda\}$. It returns an accumulated value z and a sequence of proofs $\pi_1^{\mathsf{acc}}, \dots, \pi_\lambda^{\mathsf{acc}}$ where π_i^{acc} can be used to prove that m_i is in the accumulated value z where each $m_i \in \{0,1\}^*$.
- $\alpha.\mathsf{Verify}(m, \pi^{\mathsf{acc}}, z)$: A function that checks if a proof π^{acc} proves that a message m was in the set of elements used to create the accumulated value z.

With the following properties:

Completeness: All honestly generated proofs are accepted by $\alpha.\mathsf{Verify}$.

Collision-freeness: No polynomial-time adversary can find a set of values $M := \{m_1, \dots, m_\lambda\}$, a value $m' \notin M$, and a proof π^{acc} such that $\alpha.\mathsf{Verify}(m', \pi^{\mathsf{acc}}, z) = \top$ for $z \leftarrow \alpha.\mathsf{Accumulate}(M)$.

We use the notation $\alpha.\mathtt{AccSize}$ for a bound on the size of the accumulated value and $\alpha.\mathtt{ProofSize}(\lambda)$ for a function that bounds the size of each proof as a function of the number of messages accumulated λ.

Note that we here assume deterministic accumulators. Similarly to erasure correcting codes, this is needed for our constructions. One can use Merkle trees [Mer90] to instantiate such a WSCAS, where the accumulated value is the root of the Merkle tree and the proofs are the Merkle proofs for each message. This yields an efficient deterministic scheme with

$$\alpha.\mathtt{AccSize} = O(\kappa) \quad \text{and} \quad \alpha.\mathtt{ProofSize}(\lambda) = O(\log(\lambda) \cdot \kappa). \tag{1}$$

2.3 Mathematical Preliminaries

We use the following generalized Chernoff bounds in our proofs [Doe20, Theorem 1.10.21].

Lemma 1 (Chernoff bound)**.** *Let $X_1, \dots, X_n$ be independent random variables with $X_i \in \{0,1\}$ for all i, and let $\mu^+ \geq E[\sum_{i=1}^n X_i]$ and $\mu^- \leq E[\sum_{i=1}^n X_i]$. We then have for all $\delta \in [0,1]$,*

$$\Pr\left[\sum_{i=1}^n X_i \leq (1-\delta)\mu^-\right] \leq e^{-\frac{\delta^2\mu^-}{2}} \quad \textit{and} \quad \Pr\left[\sum_{i=1}^n X_i \geq (1+\delta)\mu^+\right] \leq e^{-\frac{\delta^2\mu^+}{3}}.$$

3 Warmup: PushPullFlood

We design a new flooding protocol based upon the push-pull paradigm from [DGH+87], by first pushing a message to a fraction of the parties and then letting the remaining parties pull the message from those that have received it. The original push-pull protocol proposed in [DGH+87] relied on periodic pulling but is unfortunately not practical in the Byzantine setting because of two issues:

1. The time period between pull requests must be set. If it is too low, it takes a long time before a message reaches everybody. If it is too high, a lot of excess traffic is created.

2. Byzantine parties may issue an excessive amount of pull request and because honest parties cannot determine whether these request are malicious, they have to answer the requests. Thereby, the bandwidth of honest parties may be exhausted.

We address Issue 1 by letting our new protocol take an existing flooding protocol as a parameter and use this to notify parties that a message is being flooded. Thereby parties can simply issue pull requests when they receive such notification but no message. At first it may seem paradoxical to design a flooding protocol that is parameterized by an existing flooding protocol. However, note that the existing flooding protocol is only used to flood notifications showing that a message has been flooded, and such notifications are therefore significantly shorter than the actual message being flooded. Therefore, previous provably secure flooding protocols [MNT22,LZMM+22] for short messages can be used to instantiate this protocol without blowing up the communication complexity of this new protocol.

A naive way to limit the number of pull requests that honest parties must answer is to have each honest party enforce a local (possibly statistical) cap on the number of requests they respond to. However, to prevent dishonest parties from exhausting this response budget—thereby blocking genuine pull requests—such limits would need to be enforced per sending party. Unfortunately, even if an honest party answers just one pull request from each other party, it would still induce a per-party communication cost of $\Omega(n \cdot (1 - \gamma_{\text{WC}}))$, which is already too high. Instead, a partial solution to Issue 2 is to use a VRF to enforce from which neighbors a party is allowed to pull a message from, similar to how connections are established in [CKMR22]. In more detail, each party uses a VRF to obtain a random seed, which is then used to deterministically sample the set of parties from which it pulls. Parties receiving pull requests can thus verify the VRF proof and ignore illegitimate pull requests. To further prevent malicious parties from biasing their VRF keys to skew the sampling, parties evaluate the VRF on an unpredictable value from the randomness beacon, which is updated every epoch.

To ensure that an honest party pulls from at least one honest party, they must be allowed to pull from at least $\Omega(\kappa)$ neighbors. Thus, if parties are allowed to pull the *entire message* from all their neighbors, adversaries can induce a communication of least $\Omega(\kappa \cdot l \cdot (1 - \gamma_{\text{ACTUAL}}) \cdot n)$ by letting all corrupt parties pull for a message of length l from all their neighbors. To overcome this, we adapt the techniques from [LZMT24] to work for pulling instead of pushing. That is, to reduce the communication complexity of the pull phase, we let parties pull only an erasure correcting share of the original message from each neighbor. As a final ingredient and similarly to [LZMT24], we use a weak cryptographic accumulator to let honest parties recognize which shares belong together and thereby ensure that they can reconstruct the message.

We next describe our push-pull protocol for the Byzantine setting. We describe the pull step in a modular fashion, as we will reuse this step for our optimistic flooding protocol in later sections. We therefore first introduce the protocol Pull and analyze its communication complexity before presenting the protocol PushPullFlood.

3.1 Pulling

The pull-phase of the protocol has the purpose of ensuring that if a constant fraction of the parties already knows a message, then if the remaining parties begin pulling, they are all able to obtain the message with a per-communication complexity of just $O(n \cdot l \cdot \gamma_{\mathrm{WC}}^{-1})$.

The protocol makes use of a VRF and the randomness beacon we assume to be available to all parties (see Section 2.1). We assume that the VRF keys of all parties participating in the protocol are generated independently of the current (and future) values of the randomness beacon. This means in practice, users have to register their VRF keys at least one epoch before participating in the protocol. This prevents malicious parties from repeatedly generating VRF keys to skew the probabilities in their favor.

Protocol $\mathsf{Pull}(\zeta, \alpha)$

The protocol takes the following parameters:

ζ: An (μ, τ)-ECCS.

α: A WSCAS.

Each party $p_i \in \mathcal{P}$ keeps track of a set of shares received for a particular accumulator z, $\mathtt{ReceivedShares}_i[z]$ and a set of received messages $\mathtt{Messages}_i$.
Each party accepts the following three commands:

Pull message: When a party $p_i \in \mathcal{P}$ gets the input (Pull, h) they do the following:

1. Obtain random value ψ from randomness beacon for the current epoch.[a]
2. Use $\mathtt{vrf}.\mathsf{Eval}((\psi, h), \mathtt{sk}_i) = (r, \pi^{\mathsf{vrf}})$ to obtain a random seed r and a proof π^{vrf}. Use the seed to deterministically sample with replacement a multiset $S = \{\!\{p_1, \ldots, p_\mu\}\!\}$ from $\mathcal{P}$ s.t. $|S| = \mu$, and S is distributed uniformly for uniform r.
3. Now, the parties in S are deterministically enumerated $S = \{\!\{p_1, \ldots, p_\mu\}\!\}$. For each party $p_j \in S$ they send $(\mathit{Pull}, r, h, \pi^{\mathsf{vrf}}, j)$ to this party.
4. Whenever a party $p_j \in S$ responds with the requested share and a proof of which accumulator it belongs to $(\mathit{Share}, s_j, \pi^{\mathsf{acc}}, z)$, p_i first check that the proof π^{acc} verifies that the share s_i belongs to the accumulator z. If both checks pass, then p_i adds s_i to $\mathtt{ReceivedShares}_i[z]$.
5. When a party has received sufficiently many shares they will reconstruct the shares to get a message m' which they will add to $\mathtt{Messages}_i$.

Accept pull: When a party $p_i \in \mathcal{P}$ gets the input $(\mathit{AcceptPull}, m)$ they first share the message m into shares $\zeta.\mathsf{Enc}(m) = s_1, \ldots, s_\mu$. Furthermore, they obtain an accumulated value and proofs for each share and its share number $z, \pi_1^{\mathsf{acc}}, \ldots, \pi_\mu^{\mathsf{acc}} = \alpha.\mathsf{Accumulate}(\{(s_j, j) \mid 1 \le j \le \mu\})$.

Afterwards, whenever, a message $(\mathit{Pull}, r, h, \pi^{\mathsf{vrf}}, j)$ is received from a party p_j for the first time, then party p_i checks that p_j should send this message to p_i. This is done by obtaining the random value ψ from randomness beacon for the current epoch, checking that $\mathtt{vrf}.\mathsf{Verify}((\psi, h), r, \pi^{\mathsf{vrf}}, \mathtt{pk}_j) = \top$, and checking that p_j was indeed sampled as the j'th party using the seed r. If this check passes, the party sends the accumulator and share values $(\mathit{Share}, s_j, \pi^{\mathsf{acc}}, z)$ that belong to the hash value h to party p_j.

Get messages: On input ($\mathcal{GetMessages}$) to p_i the party returns the set of messages they have received $\texttt{Messages}_i$.

[a]We here assume for simplicity that all parties agree on what the current epoch is. In practice, this can be achieved by letting the sender of message include the epoch number ν in the message and propagate this epoch number to protocol calls, i.e., the party would in this case get the input $(\mathcal{Pull}, \nu, h)$ instead of $(\mathcal{Pull}, h)$ etc.

We now state and prove the necessary property of this protocol.

Lemma 2. *Let $\delta \in (0,1]$, let ζ be an (μ, τ)-ECCS and let α be a WSCAS. For any $\beta \in (0,1]$ and any message m, if*

1. *at least $\beta \cdot n$ of the parties are honest and have input $(\mathcal{AcceptPull}, m)$,*
2. *the last honest party inputs $(\mathcal{Pull}, \mathtt{H}(m))$ to the protocol $\mathsf{Pull}(\zeta, \alpha)$ at time t,*
3. *and $\tau \geq \mu \cdot (1 - (1-\delta) \cdot \beta)$,*

then the probability that some party has not received the message m by time $t + 2 \cdot \Delta_{\text{CHANNEL}}$ is less than $\gamma_{\text{ACTUAL}} \cdot n \cdot e^{-\frac{\delta^2 \cdot \mu \cdot \beta}{2}}$.

Proof. Consider an honest party p that has not received $(\mathcal{AcceptPull}, m)$ as input. We introduce indicator random variables $X_1, \ldots, X_\mu$ where X_j indicates whether the j'th party from which p requests a share is honest and has already received $(\mathcal{AcceptPull}, m)$ before time t. Since we assume both the erasure correcting code as well as the weak cryptographic accumulator to be deterministic, that party will in this case have generated the same shares and accumulator proofs as other parties. Therefore, p will in this case have received share j and a valid proof by time $t + 2 \cdot \Delta_{\text{CHANNEL}}$. Further, note that if p has received at least $\mu - \tau$ valid shares by time $t + 2 \cdot \Delta_{\text{CHANNEL}}$, then p is able to reconstruct the original message timely by the correctness property of the ECCS and the unforgeability of the WSCAS. Therefore, by the assumption that

$$\tau \geq \mu \cdot (1 - (1-\delta) \cdot \beta) \iff \mu - \tau \leq (1-\delta) \cdot \mu \cdot \beta, \tag{2}$$

we have

$$\begin{aligned} &\Pr[p \text{ has not received message } m \text{ by time } t + 2 \cdot \Delta_{\text{CHANNEL}}] \\ &\leq \Pr\left[\sum_{j=1}^{\mu} X_j \leq \mu - \tau\right] \\ &\leq \Pr\left[\sum_{j=1}^{\mu} X_j \leq (1-\delta) \cdot \mu \cdot \beta\right]. \end{aligned} \tag{3}$$

Parties sample the μ neighbors with replacement using a random seed r obtained from the VRF given the message hash and a fresh value from the randomness beacon. Since we assume the randomness beacon produces unpredictable values, independent from the VRF keys, the X_j are computationally indistinguishable from independent and identically distributed values. We further note that, up to some negligible distinguishing advantage, the expected value of any X_j is given by $\mathrm{E}[X_j] = \beta$, and there Chernoff (Lemma 1)implies

$$\Pr\left[\sum_{j=1}^{\mu} X_j \leq (1-\delta) \cdot \mu \cdot \beta\right] \leq e^{-\frac{\delta^2 \cdot \mu \cdot \beta}{2}}. \tag{4}$$

Noting that there are at most $\gamma_{\text{Actual}} \cdot n$ such parties and using a union-bound together with Equation (3), we get the desired bound

$$\begin{aligned}&\Pr[\text{some honest party has not received message } m \text{ by time } t + 2 \cdot \Delta_{\text{Channel}}]\\ &\leq \gamma_{\text{Actual}} \cdot n \cdot e^{-\frac{\delta^2 \cdot \mu \cdot \beta}{2}}. \end{aligned}$$ $\square$

Communication complexity of pulling. Let us now analyze the communication complexity of the pulling protocol. We first concentrate on the communication complexity induced for a party to pull a message. For each party pulling, there will be μ pulling requests. The pulling requests will each consist of:

1. A tag *Pull* of size $O(1)$,
2. the random seed from the VRF to determine whom to pull from of size $O(\kappa)$,
3. the proof that this seed has been correctly calculated of size $O(\kappa)$,
4. the hash of the message of size $O(\kappa)$,
5. and the index of the requested share of size $O\big(\log(\mu)\big)$.

Therefore, each such pull request will have size

$$O(1) + O(\kappa) + O(\kappa) + O(\kappa) + O\big(\log(\mu)\big) = O\big(\kappa + \log(\mu)\big), \tag{5}$$

and the total communication complexity for such pulling party will be

$$\mu \cdot O\big(\kappa + \log(\mu)\big) = O\big(\mu \cdot (\kappa + \log(\mu))\big). \tag{6}$$

Next, let us analyze the communication complexity of responding to such *valid*[1] pull requests. A response to such pulling request will consist of:

1. A tag *Share* of size $O(1)$,
2. a share of size $\zeta.\texttt{ShareSize}(l)$,
3. the accumulated value of all shares and indexes with size $\alpha.\texttt{AccSize}$,
4. and the proof that the share is a part of the accumulator with size $\alpha.\texttt{ProofSize}(\mu)$.

Therefore, each such response will have size

$$O(1) + \zeta.\texttt{ShareSize}(l) + \alpha.\texttt{AccSize} + \alpha.\texttt{ProofSize}(\mu). \tag{7}$$

The total communication complexity for a party having accepted pulls, will therefore be what is given in Equation (7) multiplied with the number of such valid pull requests the party receives.

By the pseudorandomness property of the VRF, a verifying pull requests can only be established by letting the party knowing their secret key evaluate the VRF.[2] To upper bound

[1]That is, a pull request where the attached VRF-proof proofs that the share should actually be requested from this specific party.

[2]To see that this follows from the pseudorandomness, consider for the sake of contradiction an adversary with a non-negligible probability of evaluating a VRF without knowing a corresponding secret key. This can be used to distinguish an output of the VRF from a uniformly random value non-negligibly by using the adversary capable of evaluating such VRF with a non-negligible probability and if the output matches the challenge, guess that it was produced by the VRF.

the per-party communication complexity let η be an upper bound on the number of secret keys that are used to evaluate the VRF for a particular message. Each VRF evaluation gives rise to μ pull requests. The outputs of each such evaluation from a secret key is guaranteed to be unique by the full uniqueness property which ensures that there are therefore at most $\eta \cdot \mu$ valid pull requests in total for a particular message.

For a particular party p, we introduce an indicator random variable X_i for each of the pull requests $i \in \{1, \ldots \eta \cdot \mu\}$ where $X_i = 1$ if and only if the i'th of these potential pull requests targets p as a valid receiver of the pull request. As the target of a valid pull request is drawn uniformly at random among all parties (up to some negligible distance by the pseudorandomness property of the VRF and the definition of the protocol), we have for any X_i that the expected value is $\mathrm{E}[X_i] = n^{-1}$ and therefore $\mathrm{E}\left[\sum_{i=1}^{\eta \cdot n \cdot \mu} X_i\right] = \eta \cdot \mu \cdot n^{-1}$. Further, because the sampling is done with replacement, we have as in the proof of Lemma 2 that the values X_i are indistinguishable from independent and identically distributed random variables. We can thus apply the Chernoff bound to obtain (up to negligible distance) for any $\delta \in [0, 1]$

$$\Pr\left[\sum_{i=1}^{\eta \cdot n \cdot \mu} X_i \geq (1 + \delta) \cdot \eta \cdot \mu \cdot n^{-1}\right] \leq e^{-\frac{\delta^2 \cdot \eta \cdot \mu}{3 \cdot n}}. \tag{8}$$

Further, using the union bound, we can bound the probability that any party receives more pull requests than what we used in the bound above:

$$\Pr\left[\exists p \text{ receiving more than } (1 + \delta) \cdot \eta \cdot \mu \cdot n^{-1} \text{ pull requests}\right] \leq n \cdot e^{-\frac{\delta^2 \cdot \eta \cdot \mu}{3 \cdot n}}. \tag{9}$$

Letting δ be constant, we note that the probability that any party receives more than $O(\eta \cdot \mu \cdot n^{-1})$ valid pull requests is negligible in κ when $\mu \geq 3 \cdot n \cdot (\log(n) + \kappa) \cdot (\delta^2 \cdot \eta)^{-1}$. Hence, the per-party communication complexity for a party accepting pull requests with these parameters will be

$$O(\eta \cdot \mu \cdot n^{-1} \cdot (\zeta.\texttt{ShareSize}(l) + \alpha.\texttt{AccSize} + \alpha.\texttt{ProofSize}(\mu))). \tag{10}$$

3.2 Push-Pull Flooding

We now present the full protocol that combines the push and pull phases. Before presenting the actual protocol, we introduce a weakened delivery guarantee, that will be used for the push-phase of the protocol.

Fractional delivery. We here introduce a weakened version of the (γ, Δ)-delivery guarantee, namely a version where it is not required that a message is delivered to all parties, but rather only to a fraction of the parties. The idea is that we will use a protocol with this weaker delivery guarantee to spread out messages to a large fraction of the parties before the pulling phase is initiated. We dub this weakened property *fractional delivery* and define it formally below.

Definition 7 ((β, γ, Δ)-fractional delivery)**.** We say that a flooding protocol Π has (β, γ, Δ)-*fractional delivery* for $\beta, \gamma \in [0, 1]$ and $\Delta > 0$ if the following holds: When a message m is input to an *honest* party at time t and $\gamma_{\text{ACTUAL}} \geq \gamma$, then at least a β fraction of honest parties output m by time $t + \Delta$, except with probability negligible in the security parameter κ.

Sometimes, we will refer to a flooding protocol with this property as a *fractional* flooding protocol.

The idea of not delivering messages to all parties was also considered in [CKMR22].[3] However, their work builds a custom consensus protocol on top of the weaker message dissemination

[3]In particular, the F_{sync} functionality of [CKMR22, p. 7] allows a fraction of the parties to be eclipsed in which case the delivery guarantees will not apply.

functionality. In contrast, our work uses the weaker property as a step towards building a flooding protocol that ensures message delivery to *all* parties.

Push and then pull flooding protocol. We now present our flooding protocol that works by first making the parties push out a notification for that a message is about to arrive (the hash of the message) and push out the actual message using a fractional flooding protocol. Afterwards, all parties are allowed to pull for the message if they did not receive the message that they were notified about.

Protocol $\mathsf{PushPullFlood}(\Pi_{\text{Flood}}, \Pi_{\text{FracFlood}}, \zeta, \alpha)$

The protocol takes the following parameters:

$\Pi_{\textbf{Flood}}$**:** a flooding protocol with $(\gamma_{\text{WC}}, \Delta_{\text{Flood}})$-delivery.

$\Pi_{\textbf{FracFlood}}$**:** a flooding protocol with $(\beta, \gamma_{\text{WC}}, \Delta_{\text{FracFlood}})$-fractional delivery.

ζ**:** A ECCS.

α**:** A WSCAS.

Each party p_i keeps track of a set of received messages $\texttt{Messages}_i$ that initially is empty, and runs an instance of the protocols $\mathsf{Pull}(\zeta, \alpha)$, Π_{Flood}, and $\Pi_{\text{FracFlood}}$.
Each party accepts the following two commands:

Send: When party p_i receives input $(\mathcal{Send}, m)$ they:

1. Send a hash of the message $(\mathcal{Hash}, \texttt{H}(m))$ to all parties using Π_{Flood}.
2. Send the message using $\Pi_{\text{FracFlood}}$.
3. Add m to $\texttt{Messages}_i$
4. Input $(\mathcal{AcceptPull}, m)$ to $\mathsf{Pull}(\zeta, \alpha)$.

Get messages: On input $(\mathcal{GetMessages})$ to p_i the party returns the set of messages they have received $\texttt{Messages}_i$.

Additionally, at all times the parties do the following:

1. Whenever a party p_i receives message h in the protocol Π_{Flood}, the party notes down the time t. If there is no message $m' \in \texttt{Messages}_i$ s.t. $\texttt{H}(m') = h$ at time $t + \Delta_{\text{FracFlood}}$, then they issue $(\mathcal{Pull}, h)$ to $\mathsf{Pull}(\zeta, \alpha)$.

2. Whenever a party p_i receives a message m in the protocol $\Pi_{\text{FracFlood}}$, they add m to $\texttt{Messages}_i$ and input $(\mathcal{AcceptPull}, m)$ to $\mathsf{Pull}(\zeta, \alpha)$.

3. Whenever a party p_i receives a message m in $\mathsf{Pull}(\zeta, \alpha)$, they add m to $\texttt{Messages}_i$.

Below, we state and prove the security of $\mathsf{PushPullFlood}$.

Theorem 3. *Let $\mu, \tau, \Delta_{Flood}, \Delta_{FracFlood} \in \mathbb{N}$, let $\beta, \delta \in (0, 1]$, and let α be a WSCAS. If*

1. Π_{Flood} ensures $(\gamma_{\text{WC}}, \Delta_{Flood})$-delivery,

2. $\Pi_{FracFlood}$ ensures $(\beta, \gamma_{\text{WC}}, \Delta_{FracFlood})$-fractional delivery, *3. and ζ is a (μ, τ)-ECCS with*

   (a) $\mu \geq 2 \cdot (\log(n) + \kappa) \cdot (\beta \cdot \gamma_{\text{WC}} \cdot \delta^2)^{-1}$ and

   (b) $\tau \geq \mu \cdot (1 - (1 - \delta) \cdot \beta \cdot \gamma_{\text{WC}})$,

then $\mathsf{PushPullFlood}(\Pi_{\mathit{Flood}}, \Pi_{\mathit{FracFlood}}, \zeta, \alpha)$ ensures $(\gamma_{\text{WC}}, \Delta_{\mathit{Flood}} + \Delta_{\mathit{FracFlood}} + 2 \cdot \Delta_{\textsc{Channel}})$-delivery.

Proof. Assume there are at least $\gamma_{\text{WC}} \cdot n$ honest parties and let m be a message input to some honest party at time t. We let

- A be the event that all honest parties have received m by time $\Delta_{\text{Flood}} + \Delta_{\text{FracFlood}} + 2 \cdot \Delta_{\textsc{Channel}}$,
- B be the event that all honest parties have received $\mathtt{H}(m)$ by time $t + \Delta_{\text{Flood}}$,
- and let C be the event that at least a β fraction of the honest parties have received m by time $t + \Delta_{\text{FracFlood}}$.

By the law of total probability and the assumptions on Π_{Flood} and $\Pi_{\text{FracFlood}}$, we have that

$$\Pr[A] \geq \Pr[A \mid B \cap C] \cdot \Pr[B \cap C] \geq \Pr[A \mid B \cap C] \cdot (1 - \text{negl}(\kappa)). \tag{11}$$

Further, as $\Pr[A \mid B \cap C] = 1 - \Pr[\neg A \mid B \cap C]$, it is sufficient to prove that $\Pr[\neg A \mid B \cap C] \leq \text{negl}(\kappa)$. Note that C ensures that a β fraction of the honest parties, i.e., at least a fraction $\beta' := \beta \cdot \gamma_{\text{WC}}$, has received m in the protocol $\Pi_{\text{FracFlood}}$ by time $t + \Delta_{\text{FracFlood}}$, and thus has input $(\mathcal{AcceptPull}, m)$ to $\mathsf{Pull}(\zeta, \alpha)$ by then. Further note that B ensures that by time $t + \Delta_{\text{Flood}} + \Delta_{\text{FracFlood}}$, all honest parties either have received m or have input $(\mathcal{Pull}, \mathtt{H}(m))$ to $\mathsf{Pull}(\zeta, \alpha)$. Finally note that we have $\tau \geq \mu \cdot (1 - (1 - \delta) \cdot \beta')$ by assumption. Hence, the preconditions for Lemma 2 are fulfilled, which gives us that:

$$\Pr[\neg A \mid B \cap C] \leq \gamma_{\textsc{Actual}} \cdot n \cdot e^{-\frac{\delta^2 \cdot \mu \cdot \beta \cdot \gamma_{\text{WC}}}{2}} \leq n \cdot e^{-\frac{\log(n) + \kappa}{2}} \leq \text{negl}(\kappa). \tag{12}$$

□

Communication complexity of pushing and pulling. We bound the per-party communication complexity $\mathtt{PPCC}(\mathsf{PushPullFlood}(\Pi_{\text{Flood}}, \Pi_{\text{FracFlood}}, \zeta, \alpha)$ with the variables instantiated according to Theorem 3, choosing the number of shares as small as possible while fulfilling $\mu \geq 2 \cdot (\log(n) + \kappa) \cdot (\beta \cdot \gamma_{\text{WC}} \cdot \delta^2)^{-1}$. The per-party communication complexity of $\mathsf{PushPullFlood}$ will be the sum of the following:

1. The per-party communication complexity from the flooding protocol used to send the hash of the message.
2. The per-party communication complexity from the fractional flooding protocol used to send the message itself.
3. The per-party communication complexity induced by the pulling phase.

The former two terms depends on the sub-protocols that $\mathsf{PushPullFlood}$ is instantiated with (i.e., Π_{Flood} and $\Pi_{\text{FracFlood}}$) and we therefore go on to analyze the size of the last term: The per-party communication complexity of the pulling phase. In the worst-case a party will first pull and afterwards receive the message and thereby accepts pulling requests.[4] The communication

[4]This can easily be mitigated without compromising the security of the protocol by simply not allowing any party to both pull and accept pulls.

induced by a party pulling was already established in Equation (6) and the communication complexity for a party to answer pull-request was established in Equation (10) given an upper bound on how many parties evaluate the VRF. It is therefore sufficient to bound the number of parties that evaluate the VRF for the specific message. The fractional flooding protocol guarantees that at least $\beta \cdot \gamma_{\mathrm{WC}} \cdot n$ will not evaluate the VRF for the hash of this message as they are honest and have timely received a message whose hash matches the notification. Therefore, at most $(1-\beta \cdot \gamma_{\mathrm{WC}}) \cdot n$ evaluate the VRF. Further, for a suitable ECCS (such as the Reed-Solomon construction described in [LZMT24] with $\zeta.\texttt{ShareSize}(l) = O\left(\frac{l}{\mu-\tau}\right)$) and $\tau = \Theta(\mu \cdot (1-\beta \cdot \gamma_{\mathrm{WC}}))$ we have

$$\zeta.\texttt{ShareSize}(l) = O\big(l \cdot (\mu - \mu \cdot (1-\beta \cdot \gamma_{\mathrm{WC}}))^{-1}\big) = O\big(l \cdot (\mu \cdot \beta \cdot \gamma_{\mathrm{WC}})^{-1}\big). \tag{13}$$

Specializing Equation (10) to this setting, letting $\mu = \Theta((\log(n)+\kappa)\cdot(\beta \cdot \gamma_{\mathrm{WC}})^{-1})$ to ensure that the probability is overwhelming in κ and combining this with the previous analysis of the communication induced by each such request, we get

$$\begin{aligned}
&O((1-\beta \cdot \gamma_{\mathrm{WC}}) \cdot \mu \cdot (O(1) + \zeta.\texttt{ShareSize}(l) + \alpha.\texttt{AccSize} + \alpha.\texttt{ProofSize}(\mu)))\\
&= O\big((1-\beta \cdot \gamma_{\mathrm{WC}}) \cdot \mu \cdot (O(1) + O\big(l \cdot (\mu \cdot \beta \cdot \gamma_{\mathrm{WC}})^{-1}\big) + O(\kappa) + O\big(\kappa \cdot \log\big((\log(n)+\kappa)\cdot(\beta \cdot \gamma_{\mathrm{WC}})^{-1}\big)\big)\big)\big)\\
&= O\big(l \cdot (\beta \cdot \gamma_{\mathrm{WC}})^{-1} + \big((\log(n)+\kappa)\cdot(\beta \cdot \gamma_{\mathrm{WC}})^{-1}\big) \cdot \kappa \cdot \log\big((\log(n)+\kappa)\cdot(\beta \cdot \gamma_{\mathrm{WC}})^{-1}\big)\big)\\
&= \tilde{O}\big(l \cdot (\beta \cdot \gamma_{\mathrm{WC}})^{-1} + \kappa^2 \cdot (\beta \cdot \gamma_{\mathrm{WC}})^{-1}\big)
\end{aligned} \tag{14}$$

Using the same instantiation of variables for a pulling party (i.e. (Equation (6))), we get that the communication for a pulling party is bounded by

$$\begin{aligned}
O(\mu \cdot (\kappa + \log(\mu))) &= O((\log(n)+\kappa)\cdot(\beta \cdot \gamma_{\mathrm{WC}})^{-1} \cdot (\kappa + \log((\log(n)+\kappa)\cdot(\beta \cdot \gamma_{\mathrm{WC}})^{-1})))\\
&= \tilde{O}(\kappa^2 \cdot (\beta \cdot \gamma_{\mathrm{WC}})^{-1})
\end{aligned} \tag{15}$$

We note that asymptotically this does not add more to the per-party communication complexity than what is already induced by accepting pull. Using the above bounds on the individual parts of per-party communication required for accepting pulls and for pushing we get that the per-party communication complexity for the entire protocol is bounded by

$$\begin{aligned}
&\texttt{PPCC}(\mathsf{PushPullFlood}(\Pi_{\mathrm{Flood}}, \Pi_{\mathrm{FracFlood}}, \zeta, \alpha), l)\\
&\qquad \le \texttt{PPCC}(\Pi_{\mathrm{Flood}}, \kappa) + \texttt{PPCC}(\Pi_{\mathrm{FracFlood}}, l) + \tilde{O}\big(l \cdot (\beta \cdot \gamma_{\mathrm{WC}})^{-1} + \kappa^2 \cdot (\beta \cdot \gamma_{\mathrm{WC}})^{-1}\big).
\end{aligned} \tag{16}$$

Hence, if the fractional flooding algorithm ensures to deliver the message to a constant fraction of the parties, then PushPullFlood only gives an overhead compared to fractional flooding that is asymptotically optimal in the message length for messages of length $l = \tilde{\Omega}(\kappa^2 \cdot (\beta \cdot \gamma_{\mathrm{WC}})^{-1})$.

Therefore, to ensure that the entire flooding protocol is an asymptotically optimal flooding protocol, it is only left to instantiate the fractional flooding algorithm in a suitable manner. However, as the flooding protocol presented in [LZMT24], ECFlood, was already proven to be asymptotically optimal and any flooding protocol also is a fractional flooding protocol, it follows immediately (by the above discussion of the communication complexity of Pull), that PushPullFlood is an asymptotically optimal flooding protocol as well when instantiated with ECFlood and a suitable choice flooding algorithm for the notifications such as FFlood from [LZMT24].

4 OptimisticFlood

In this section, we present our optimistic flooding protocol OptimisticFlood that has an optimistic path such that under certain conditions it is guaranteed to have a communication complexity that is much lower than in worst-case scenarios.

4.1 Protocol

Protocol intuition. Our protocol is parameterized by two flooding protocols: 1) a best-case flooding protocol that only works if some best-case conditions are fulfilled, 2) a worst-case flooding protocol that is ensured to work in all remaining cases. The best-case conditions will be that at least a fraction γ_{BC} of the parties remain honest throughout the execution.

A first skeleton of an optimistic flooding protocol relying on two such existing flooding protocols could look like the following:

1. Run a best-case protocol to disseminate the actual message.
2. Check if the best-case protocol succeeds. If not, default to sending the entire message using the worst-case protocol.

While the skeleton reads fairly straightforward, it is easier said than done to reliably detect if the best-case protocol fails while still tolerating a small fraction of corrupted parties. Because flooding protocols only ensure the delivery of the message when the initial sender is honest, a first step towards this could be to let the initial sender ask all parties if they have received the message. We refer to a party reporting that they have not received the message as a *complaint*. Based on the answers given by the parties, a decision must be taken on whether we default back to use the worst-case flooding protocol and use this to send the entire message. The decision procedure and the following actions should account for the following two cases depending on the actual fraction of parties being honest γ_{Actual}:

$\gamma_{\text{bc}} \leq \gamma_{\text{Actual}}$: In this case, the best-case protocol is guaranteed to deliver the message to all honest parties and therefore the worst-case protocol should not be executed, independently of the actions of the malicious parties. In particular, these up to $n \cdot (1 - \gamma_{\text{BC}})$ malicious parties may complain about not having received the message even though they have.

$\gamma_{\text{wc}} \leq \gamma_{\text{Actual}} < \gamma_{\text{bc}}$: In this case, we have no guarantees from the best-case flooding protocol about how many parties have received the message and the adversary can choose this arbitrarily (by delivering the message to specific parties). In particular, it may be that the adversary does not deliver the message to $n \cdot (1 - \gamma_{\text{BC}})$ parties.

Note that from the sender's point of view, it will be impossible to distinguish which of the two above cases they are in as an adversary can make the views appear exactly the same for the sender. So how do we ensure that $n \cdot (1 - \gamma_{\text{BC}})$ parties cannot force the execution of the worst-case protocol in the first case while ensuring that all honest parties receive the message in the second case?

To balance this we introduce a subsequent pull-phase in case we decide that the best-case protocol "succeeded" allowing parties not having received the message to pull similar to the pull-phase from the protocol PushPullFlood presented in Section 3. That is, instead of requiring that no honest party complains to conclude a success, we only require that not more than a fraction of the parties complain in order to conclude that the best-case protocol succeeded. Concretely, we introduce a threshold T for how many complaints we will accept and still conclude that the best-case protocol "succeeded". We will set this threshold such that $(1 - \gamma_{\text{BC}}) \cdot n$ parties cannot produce enough complaints to conclude that the protocol failed if it did not, but still it should ensure that a sufficient fraction of the honest parties have received the message to ensure that pull requests will be responded to appropriately. Thereby, we can use the protocol Pull without prohibitively high communication.

What is left is only to combat the impracticality of letting the sender ask *all* parties. We do this by letting the sender sample a subset of the parties and ask this subset of parties about whether or not they have received the message. This allows the sender to statistically conclude whether or not the best-case protocol succeeded.

Protocol description. Below, we present our protocol for optimistic flooding.

Protocol $\mathsf{OptimisticFlood}(\Pi_{\text{BC}}, \Pi_{\text{WC}}, c, T, \zeta, \alpha)$

The protocol has the following parameters:

$\Pi_{\mathbf{bc}}$**:** A flooding protocol that should work in the best-case ensuring delivery within Δ_{BC} time when at least a γ_{BC} fraction of the parties remains honest.

$\Pi_{\mathbf{wc}}$**:** A flooding protocol that should work in the worst-case ensuring delivery within Δ_{WC} time when at least a γ_{WC} fraction of the parties remains honest.

c**:** The size of the subset the sender should ask for complaints.

T**:** A threshold that decides how many complaints are acceptable.

ζ**:** A ECCS.

α**:** A WSCAS.

Each party p_i keeps track of a set of received messages $\texttt{Messages}_i$ that initially is empty, and runs an instance of the protocols $\mathsf{Pull}(\zeta, \alpha)$, Π_{Flood}, and $\Pi_{\text{FracFlood}}$.

Send: When a party s receives input (Send, m) they do the following:

1. The sender s sends $(\mathit{Message}, m)$ to all parties using Π_{BC}. We let the time that this happens be denoted t_{INIT}.
2. At time $t_{\text{INIT}} + \Delta_{\text{BC}}$[a] the sender uniformly at random (with repetition) samples a committee of parties $C = \{p_1, \ldots, p_c\} \subseteq \mathcal{P}$. For each party $p \in C$ the sender sends a direct message using an authenticated channel $(\mathit{Received?}, \texttt{H}(m), t_{\text{INIT}} + \Delta_{\text{BC}})$.
3. At time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}}$[b], the sender counts how many unique complaints they have received from valid committee members for $\texttt{H}(m)$. We let the count be denoted by k, and based on this the sender does one of following two things:
   (a) If $k > T$, then the sender sends the original message $(\mathit{Message}, m)$ to all parties using Π_{WC}.
   (b) Otherwise if $k \leq T$, the sender initializes a pull-phase by signing and sending $(\mathit{PullPhaseBegun}, \texttt{H}(m))$ to all parties using Π_{WC}.

Get messages: On input $(\mathit{GetMessages})$ to p_i, the party returns the set of messages they have received $\texttt{Messages}_i$.

Additionally, at all times the parties do the following:

- When party p_i receives a message $(\mathit{Received?}, h, t)$ over an authenticated channel from a party s, the party checks if there is any message $m \in \texttt{Messages}_i$ s.t. $\texttt{H}(m) = h$ which has been received before time t.[c] If no such message exists, then they send $(\mathit{Complaint}, h)$ to s over the authenticated channel.

- When a party p_i receives $(\mathit{PullPhaseBegun}, h)$ over Π_{WC} at time t they do the following:

1. If there is any message $m \in \texttt{Messages}_i$ s.t. $\texttt{H}(m) = h$, then they input $(\mathcal{AcceptPull}, m)$ to $\mathsf{Pull}(\zeta, \alpha)$.
2. Otherwise, if no such message exists, then they input $(\mathcal{Pull}, h)$ to $\mathsf{Pull}(\zeta, \alpha)$ at time $t + \Delta_{\mathrm{WC}}$.[d]

- When party p_i receives a message m in either the best-case protocol, the worst-case protocol or the pulling protocol, they add it to $\texttt{Messages}_i$ and store the time they received it together with the message.

[a]This timing ensures that the best-case protocol have had sufficient time to deliver the message.

[b]At this time it is ensured that all honest parties' complaints have reached the sender.

[c]Note that it is necessary to require that the message have been received before time t to ensure that the number of complaints from the set of parties sampled accurately reflects the share of parties that have actually received the message. If this condition was not enforced, an adversary could choose to deliver the message to all the parties part of the committee once the sender sends the $\mathcal{Received?}$-message to them.

[d]The reason that the party does not immediately input the message to the pull protocol, is that it must be ensured that sufficiently many parties have already input an accept of the message to the pulling protocol.

Note that for the provably secure protocols, we often have $\Delta_{\mathrm{BC}}, \Delta_{\mathrm{WC}} = O\big(\log(n) \cdot \Delta_{\mathrm{CHANNEL}}\big)$. Hence, the direct communication steps that happen using a channel are comparatively "cheap" time-wise.

It is also worth noting that instead of using just two different flooding protocols, one could consider using three different protocols. A natural example of this would be to use one worst-case protocol for short messages (notifications), one worst-case protocol for long-messages, and one best-case protocol for long messages.

The described version of the protocol requires a synchrony assumption on the channels for safety (i.e., guaranteeing message delivery) in the worst case to ensure that sufficiently many complaints reach the sender in time. In the optimistic case, it does not require synchrony. However, if we instead changed the protocol to require a certain number of "confirmations" from parties having received the message instead of complaints, the protocol would achieve safety without relying on synchrony. On the other, it would still only achieve the best-case communication complexity under synchronous conditions and require slightly more communication in the optimistic case to send the confirmations. This allows fine-tuning the protocol for the specific settings it is deployed in.

4.2 Correctness

In this section, we prove the correctness of the protocol $\mathsf{OptimisticFlood}$ under relevant conditions .We start out by stating that if the actual fraction honest parties γ_{ACTUAL} is bigger than the best-case threshold γ_{BC}, then for certain parameters, we achieve the delivery guarantees of the best-case protocol.

Lemma 3 (Best-case correctness)**.** *Let $c \in \mathbb{N}$ be the size of the committee, $T \in \mathbb{N}$ be the complaint threshold, Π_{WC} a protocol, ζ a ECCS, α be a WSCAS, and $\delta \in (0, 1]$. If*

1. *Π_{BC} has $(\gamma_{BC}, \Delta_{BC})$-delivery*
2. *and $T \geq (1 + \delta) \cdot (1 - \gamma_{BC}) \cdot c$*

then $\mathsf{OptimisticFlood}(\Pi_{BC}, \Pi_{WC}, c, T, \zeta, \alpha)$ has $(\gamma_{BC}, \Delta_{BC})$-delivery and if $\gamma_{\mathrm{ACTUAL}} \geq \gamma_{BC}$ and the sender is honest, then the probability that Step 3a is activated is less than $e^{-\frac{\delta^2 \cdot (1-\gamma_{BC}) \cdot c}{3}}$.

The intuition for Condition 2 is that it corresponds to requiring that under best-case conditions, the threshold for the number of complaints is set sufficiently large such that only dishonest parties in the committee cannot make the sender default back to using the worst-case protocol. We now proceed with the proof.

Proof. We note that because the entire message is immediately input to Π_{BC}, the delivery guarantees for Π_{BC} directly apply and hence $\mathsf{OptimisticFlood}(\Pi_{\mathrm{BC}}, \Pi_{\mathrm{WC}}, c, T, \zeta, \alpha)$ has $(\gamma_{\mathrm{BC}}, \Delta_{\mathrm{BC}})$-delivery.

We now bound the probability that Step 3a is activated for an honest sender. Let the time that the honest sender sends a message m be denoted t_{INIT} and note that by the above, all honest parties have received the message at time $t_{\mathrm{INIT}} + \Delta_{\mathrm{BC}}$. Hence, no honest party will send back $(\mathit{Complaint}, \mathtt{H}(m))$ to the sender. It is therefore sufficient to show that the number of corrupted parties in the committee is at most T with overwhelming probability.

To do so, we let $X_1, \ldots, X_c$ denote indicator variables s.t. $X_i = 1$ if and only if party p_i of the committee C in Step 2 is corrupted and note that for the actual number of complaints k, it holds that $\sum_{i=1}^{c} X_i \geq k$. Further, note that

$$\mathrm{E}\left[\sum_{i=1}^{c} X_i\right] = (1 - \gamma_{\mathrm{ACTUAL}}) \cdot c \leq (1 - \gamma_{\mathrm{BC}}) \cdot c, \tag{17}$$

and that the variables are identically and independently distributed as the honest sender samples the committee at random *with repetition*. Hence, using the Chernoff bound(Lemma 1), we conclude that

$$\Pr\left[\sum_{i=1}^{c} X_i \geq (1+\delta) \cdot (1 - \gamma_{\mathrm{BC}}) \cdot c\right] \leq e^{-\frac{\delta^2 \cdot (1-\gamma_{\mathrm{BC}}) \cdot c}{3}}. \tag{18}$$

Therefore, by Condition 2, we have $\Pr\left[k > T\right] \leq e^{-\frac{\delta^2 \cdot (1-\gamma_{\mathrm{BC}}) \cdot c}{3}}$. □

Next, we state that when the parameters of the protocol are instantiated carefully, then the protocol also ensures delivery assuming just the worst-case bound on the number of honest parties. It is worth noting that the theorem only makes assumptions about the worst-case protocol. Hence, message delivery is ensured independently of which best-case protocol is deployed. In particular, this allows to use protocols that based on for example heuristics about practice.

Lemma 4 (Worst-case correctness)**.** *Let $\tau, \mu, T, c \in \mathbb{N}$, let Π_{BC} be a protocol, let α be a WSCAS, and let $\delta_1, \delta_2, \beta \in (0, 1]$. If*

1. *Π_{WC} has $(\gamma_{WC}, \Delta_{WC})$-delivery,*
2. *$T \leq (1 - \delta_1) \cdot (\gamma_{WC} - \beta) \cdot c$,*
3. *$\mu \geq 2 \cdot (\log(n) + \kappa) \cdot (\beta \cdot \delta_2^2)^{-1}$,*
4. *$c \geq \frac{\kappa}{\delta_1^2 \cdot (\gamma_{WC} - \beta)}$,*
5. *$\tau \geq \mu \cdot (1 - (1 - \delta_2) \cdot \beta)$,*
6. *and ζ is a (μ, τ)-ECCS,*

then $\mathsf{OptimisticFlood}(\Pi_{BC}, \Pi_{WC}, c, T, \zeta, \alpha)$ has $(\gamma_{WC}, \Delta_{BC} + 4 \cdot \Delta_{\mathrm{CHANNEL}} + 2 \cdot \Delta_{WC})$-delivery. Further, if the sender is honest, then the probability that there are less than $\beta \cdot n$ honest parties that have received the message and Step 3b is activated is negligible in κ.

The intuition for β is that it corresponds to a threshold for the fraction of parties that are honest and must have had the message delivered in case the sender does not receive sufficiently many complaints to default back to the worst-case protocol.

Proof. Let s be an honest sender that sends a message m at time t_{INIT}. Further, let $\theta \in [0, \gamma_{\text{ACTUAL}}]$ be the fraction of parties that are honest and have received the message m at time $t_{\text{INIT}} + \Delta_{\text{BC}}$. Because we have no guarantees about how the best-case protocol performs assuming only $\gamma_{\text{ACTUAL}} \geq \gamma_{\text{WC}}$, we have no guarantees about the value of θ. Instead, we make a case distinction whether $\theta > \beta$ or not:

$\theta > \beta$**:** For this case, we again make a case distinction based upon whether or not the actual number of complaints collected by the sender k is above the threshold T:

$k > T$**:** In this case the sender will enter Step 3a at time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}}$ and flood the entire message using the worst-case protocol Π_{WC}. Hence, by Condition 1, it is guaranteed that with overwhelming probability, all parties have learned the message at the latest at time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}} + \Delta_{\text{WC}}$.

$k \leq T$**:** In this case the sender enters Step 3b at time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}}$. Hence, the delivery guarantees of the worst-case flooding protocol Π_{WC} ensures that with overwhelming probability, all parties will have received $(\mathit{PullPhaseBegun}, \mathtt{H}(m))$ before time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}} + \Delta_{\text{WC}}$. Now, this implies that more than $\beta \cdot n$ parties are honest and have input $(\mathit{AcceptPull}, \mathtt{H}(m))$ to $\mathsf{Pull}(\zeta, \alpha)$ by this time. Furthermore, it is guaranteed that all remaining honest parties will have input $(\mathit{Pull}, \mathtt{H}(m))$ before time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}} + 2 \cdot \Delta_{\text{WC}}$. Additionally, Conditions 5 and 6 ensure that the final precondition for Lemma 2 is fulfilled. Hence, using Condition 3, the probability that at time $t_{\text{INIT}} + \Delta_{\text{BC}} + 4 \cdot \Delta_{\text{CHANNEL}} + 2 \cdot \Delta_{\text{WC}}$, there is some party who has not received the message is less than

$$\gamma_{\text{ACTUAL}} \cdot n \cdot e^{-\frac{\delta_2^2 \cdot \mu \cdot \beta}{2}} \leq n \cdot e^{-\frac{\delta_2^2 \cdot \mu \cdot \beta}{2}} \leq n \cdot e^{-\frac{\log(n)+\kappa}{2}} \leq \text{negl}(\kappa). \tag{19}$$

$\theta \leq \beta$**:** First, note (similarly to the previous case) that if for the actual number of complaints k it holds that $k > T$, then the sender enters Step 3a in which case the probability that all parties have received the message at time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}} + \Delta_{\text{WC}}$ is overwhelming in the security parameter (by Condition 1). Therefore, it is sufficient to show that the probability that $k \leq T$ is negligible in the security parameter. To show this, let $X_1, \ldots, X_c$ be indicator variables s.t. X_i indicates if the committee member i is honest and has not received the message by time $t_{\text{INIT}} + \Delta_{\text{BC}}$. Now, note that any honest party that is part of the committee and has not received the message by time $t_{\text{INIT}} + \Delta_{\text{BC}}$ will send $(\mathit{Complaint}, \mathtt{H}(m))$ at latest at time $t_{\text{INIT}} + \Delta_{\text{BC}} + \Delta_{\text{CHANNEL}}$, which means that the sender will receive it at most Δ_{CHANNEL} time later. Hence, we have that $\sum_{i=1}^{c} X_i \leq k$. By Condition 2, it is therefore to sufficient to show that

$$\Pr\left[\sum_{i=1}^{c} X_i \leq (1 - \delta_1) \cdot (\gamma_{\text{WC}} - \beta) \cdot c\right] \leq \text{negl}(\kappa). \tag{20}$$

Now note that for any i

$$\Pr[X_i = 1] = \gamma_{\text{ACTUAL}} - \theta \geq \gamma_{\text{WC}} - \theta \geq \gamma_{\text{WC}} - \beta. \tag{21}$$

Hence, we have $\text{E}\left[\sum_{i=1}^{c} X_i\right] \geq (\gamma_{\text{WC}} - \beta) \cdot c$. Further, because the sampling of the committee is done with replacement, we can apply the Chernoff bound (Lemma 1), which when using

$c \geq \frac{\kappa}{\delta_1^2 \cdot (\gamma_{\mathrm{WC}} - \beta)}$ (Condition 4) gives us:

$$\Pr\left[\sum_{i=1}^{c} X_i \leq (1-\delta_1)\cdot(\gamma_{\mathrm{WC}}-\beta)\cdot c\right] \leq e^{-\frac{\delta_1^2\cdot(\gamma_{\mathrm{WC}}-\beta)\cdot c}{2}} \leq e^{-\frac{\kappa}{2}} \leq \mathrm{negl}(\kappa). \tag{22}$$

Hence, the total failure probability will be bounded by the three negligible probabilities from above. Therefore, with overwhelming probability all parties will have received the message before time $t_{\text{INIT}} + \Delta_{\mathrm{BC}} + 4 \cdot \Delta_{\text{CHANNEL}} + 2 \cdot \Delta_{\mathrm{WC}}$. □

Finally, combining Lemmas 3 and 4, we can conclude that for suitable parameters the protocol OptimisticFlood performs well in both the best-case and the worst-case.

Corollary 1. *Let $\tau, T, c \in \mathbb{N}$ be the size of the committee, let $T \in \mathbb{N}$ be the complaint threshold, let α be a WSCAS, and let $\delta_1, \delta_2, \beta, \delta \in (0,1]$. If*

1. *Π_{BC} has $(\gamma_{BC}, \Delta_{BC})$-delivery,*
2. *Π_{WC} has $(\gamma_{WC}, \Delta_{WC})$-delivery,*
3. *$T \geq (1+\delta)\cdot(1-\gamma_{BC})\cdot c$*
4. *$T \leq (1-\delta_1)\cdot(\gamma_{WC}-\beta)\cdot c$,*
5. *$\mu \geq 2\cdot(\log(n)+\kappa)\cdot(\beta\cdot\delta_2^2)^{-1}$,*
6. *$c \geq \frac{\kappa}{\delta_1^2\cdot(\gamma_{WC}-\beta)}$*
7. *$\tau \geq \mu\cdot(1-(1-\delta_2)\cdot\beta)$,*
8. *and ζ is a (μ,τ)-ECCS,*

*then OptimisticFlood$(\Pi_{BC}, \Pi_{WC}, c, T, \zeta, \alpha)$ has both $(\gamma_{BC}, \Delta_{BC})$-delivery **and** $(\gamma_{WC}, \Delta_{BC} + 4\cdot\Delta_{\text{CHANNEL}} + 2\cdot\Delta_{WC})$-delivery. Additionally, when $\gamma_{\text{ACTUAL}} \geq \gamma_{BC}$, then the probability that Step 3a is activated for an honest sender is less than $e^{-\frac{\delta^2\cdot(1-\gamma_{BC})\cdot c}{3}}$.*

4.3 Communication Complexity

We now show that OptimisticFlood does not only ensure fast message delivery but also incurs only a minimal overhead in terms of per-party communication complexity compared to sending the message using just the best-case protocol and worst-case protocol respectively.

Let OptimisticFlood$(\Pi_{\mathrm{BC}}, \Pi_{\mathrm{WC}}, c, T, \zeta, \alpha)$ be instantiated with variables as stated in Corollary 1. First, we analyze the communication complexity of the individual steps of the protocol to send a message m of length l. We will use that hashes will be of size κ, the time can be represented using κ bits, and the VRF proofs will have size $O(\kappa)$. We will let the parameters of Corollary 1 be instantiated to give the smallest communication complexity possible. is, we will let δ, δ_1, δ_2 be small constants, and let $\beta = \Theta(\gamma_{\mathrm{WC}})$. Thereby we will have $\mu = \Theta\big((\log(n)+\kappa)\cdot\gamma_{\mathrm{WC}}^{-1}\big)$, $\tau = \Theta\big(\mu\cdot(1-\gamma_{\mathrm{WC}})\big)$, and $c = \Theta(\kappa\cdot\gamma_{\mathrm{WC}}^{-1})$. Using Reed-Solomon codes for our ECCS and a Merkle Tree as our accumulator, we will have:

$$\begin{aligned}
&\zeta.\mathtt{ShareSize}(l) = O\big(l\cdot(\mu-\tau)^{-1}\big) = O\big(l\cdot(\mu\cdot\gamma_{\mathrm{WC}})^{-1}\big),\\
&\alpha.\mathtt{AccSize} = \kappa,\\
&\alpha.\mathtt{ProofSize} = O(\kappa\cdot\log(\mu)) = O\big(\kappa\cdot\log\left((\log(n)+\kappa)\cdot\gamma_{\mathrm{WC}}^{-1}\right)\big)
\end{aligned} \tag{23}$$

Additionally, we will assume that $1 - \gamma_{\mathrm{BC}} = O(\gamma_{\mathrm{WC}})$. Note that this is not a very strong assumption as it essentially states that those that the fraction parties that are being malicious in the best case is no more than constant times the fraction of parties being honest in the worst case. We note that this holds for what we consider realistic deployment scenarios such as when $\gamma_{\mathrm{WC}} = 1/2$ and $\gamma_{\mathrm{BC}} = 19/20$.

Communication of Step 1: The communication induced by this step is the communication for using the best-case protocol to send the message:

$$\mathtt{PPCC}(\Pi_{\mathrm{BC}}, l). \tag{24}$$

Communication of Step 2: First, the sender sends a message $(\mathit{Received?}, \mathtt{H}(m), t_{\mathrm{INIT}} + \Delta_{\mathrm{BC}})$ to c parties. This message has size $O(\kappa)$. Similarly, each of these parties will at most respond with the message $(\mathit{Complaint}, \mathtt{H}(m))$ which is also bounded by $O(\kappa)$. The sender being the party doing the most communication, the per-party-communication complexity will therefore be bounded by the following:

$$O(c \cdot \kappa) = O(\kappa^2 \cdot \gamma_{\mathrm{WC}}^{-1}). \tag{25}$$

Communication of Step 3a: In this step the entire message is sent using the worst-case protocol, which induces a per-party communication complexity of:

$$\mathtt{PPCC}(\Pi_{\mathrm{WC}}, l). \tag{26}$$

Communication of Step 3b: First, we note that the sender will send the hash of the message using the worst-case protocol which induces a per-party communication complexity of $\mathtt{PPCC}(\Pi_{\mathrm{WC}}, \kappa)$. Additionally, this will make all parties begin the pulling protocol which per-party communication complexity are bounded by the sum of Equations (6) and (10). We now specialize (Equation (6)), i.e., the per-party communication for a pulling party, to this setting:

$$\begin{aligned} &O(\mu \cdot (\kappa + \log(\mu))) \\ &= O((\log(n) + \kappa) \cdot \gamma_{\mathrm{WC}}^{-1} \cdot (\kappa + \log((\log(n) + \kappa) \cdot \gamma_{\mathrm{WC}}^{-1}))) \\ &= \tilde{O}(\kappa^2 \cdot \gamma_{\mathrm{WC}}^{-1}) \end{aligned} \tag{27}$$

Let us now analyze the protocol with a case distinction on the execution based on the actual fraction of honest parties γ_{ACTUAL}. Note that Steps 1 and 2 are always executed with the same bound on the communication complexity independent of the number of honest parties and it is only the complexity of Step 3 that differs in the cases below.

$\gamma_{\mathrm{ACTUAL}} \geq \gamma_{\mathrm{BC}}$: As the probability that Step 3a is activated is negligible in the security parameter by Corollary 1, it is sufficient to concentrate on the communication complexity of Step 3b. Note that because $\gamma_{\mathrm{ACTUAL}} \geq \gamma_{\mathrm{BC}}$, it must be that all honest parties received the message at time $t_{\mathrm{INIT}} + \Delta_{\mathrm{BC}}$ (by the delivery guarantees of Π_{BC}). Further, because all honest parties have received the message before the pull-phase begins, only dishonest parties will actually pull and there will be at most $(1 - \gamma_{\mathrm{BC}}) \cdot n$ of these.

By Equation (10), the per-party communication complexity for answering pull requests will be bound by

$$\begin{aligned}
&O\Big((1-\gamma_{\text{BC}})\cdot n\cdot\mu\cdot n^{-1}\cdot(\zeta.\texttt{ShareSize}(l)+\alpha.\texttt{AccSize}+\alpha.\texttt{ProofSize}(\mu))\Big)\\
&=O\Big((1-\gamma_{\text{BC}})\cdot\mu\cdot(\zeta.\texttt{ShareSize}(l)+\alpha.\texttt{AccSize}+\alpha.\texttt{ProofSize}(\mu))\Big)\\
&=O((1-\gamma_{\text{BC}})\cdot\mu\cdot((l\cdot(\mu\cdot\gamma_{\text{WC}})^{-1})+\kappa+(\kappa\cdot\log((\log(n)+\kappa)\cdot\gamma_{\text{WC}}^{-1}))))\\
&=O((1-\gamma_{\text{BC}})\cdot\gamma_{\text{WC}}^{-1}\cdot l+(1-\gamma_{\text{BC}})\cdot(\log(n)+\kappa)\cdot\gamma_{\text{WC}}^{-1}\cdot(\kappa\cdot\log((\log(n)+\kappa)\cdot\gamma_{\text{WC}}^{-1}))) \qquad (28)\\
&=O(l+(\log(n)+\kappa)\cdot(\kappa\cdot\log((\log(n)+\kappa)\cdot\gamma_{\text{WC}}^{-1})))\\
&=\tilde{O}(l+\kappa^2).
\end{aligned}$$

In then pen ultimate step of the reduction we used that $1-\gamma_{\text{BC}}=O(\gamma_{\text{WC}})$. For the total per-party communication for this case we thus obtain the bound

$$\mathsf{PPCC}(\mathsf{OptimisticFlood}(\Pi_{\text{BC}},\Pi_{\text{WC}},c,T,\zeta,\alpha),l)\leq\mathsf{PPCC}(\Pi_{\text{BC}},l)+\mathsf{PPCC}(\Pi_{\text{WC}},\kappa)+\tilde{O}(l+\kappa^2\cdot\gamma_{\text{WC}}^{-1}) \tag{29}$$

Hence, for long messages, this implies that if $\gamma_{\text{ACTUAL}}\geq\gamma_{\text{BC}}$, the communication complexity of OptimisticFlood reduces to only basically sending the message using the best-case protocol and an additional complexity that is only linear in the length of the message which is anyway asymptotically optimal. Further, note that the additional complexity linear in the length of the message requires the adversary to *actively* try to induce more communication by sending pull requests even though all parties have received the message.

$\gamma_{\text{WC}}\leq\gamma_{\text{ACTUAL}}<\gamma_{\text{BC}}$: In this case, we do not know whether Step 3a or Step 3b are activated. Hence we make a case distinction between the two:

Step 3a is activated: In this case, the per-party communication complexity of will simply be the sum of the per-party communication complexity of the individual steps:

$$\begin{aligned}
&\mathsf{PPCC}(\mathsf{OptimisticFlood}(\Pi_{\text{BC}},\Pi_{\text{WC}},c,T,\zeta,\alpha),l)\\
&\qquad\leq\mathsf{PPCC}(\Pi_{\text{BC}},l)+O(\kappa^2\cdot\gamma_{\text{WC}}^{-1})+\mathsf{PPCC}(\Pi_{\text{WC}},l).
\end{aligned}\tag{30}$$

Step 3b is activated: By Lemma 4, it is given that the probability that less than $\beta\cdot n$ honest parties have received the message is negligible in the security parameter and to compute the per-party communication complexity of this step we can thus assume that $\beta\cdot n$ honest parties have received the message. Hence, at most $(1-\beta)\cdot n$ parties pull for the message. Using (Equation (10)) for this setting we get

$$\begin{aligned}
&O\Big((1-\beta)\cdot\mu\cdot(\zeta.\texttt{ShareSize}(l)+\alpha.\texttt{AccSize}+\alpha.\texttt{ProofSize}(\mu))\Big)\\
&=O\Big(\mu\cdot((l\cdot(\mu\cdot\gamma_{\text{WC}})^{-1})+\kappa+(\kappa\cdot\log((\log(n)+\kappa)\cdot\gamma_{\text{WC}}^{-1})))\Big)\\
&=O(l\cdot\gamma_{\text{WC}}^{-1}+(\log(n)+\kappa)\cdot\gamma_{\text{WC}}^{-1}\cdot(\kappa\cdot\log((\log(n)+\kappa)\cdot\gamma_{\text{WC}}^{-1})))\\
&=\tilde{O}(l\cdot\gamma_{\text{WC}}^{-1}+\kappa^2\cdot\gamma_{\text{WC}}^{-1}).
\end{aligned}\tag{31}$$

Summing up the communication complexity of the individual steps, and using the communication complexity of a pulling party is given by Equation (27) the communication complexity for this case will therefore be bounded

$$\begin{aligned}
&\mathsf{PPCC}(\mathsf{OptimisticFlood}(\Pi_{\text{BC}},\Pi_{\text{WC}},c,T,\zeta,\alpha),l)\\
&\qquad\leq\mathsf{PPCC}(\Pi_{\text{BC}},l)+\mathsf{PPCC}(\Pi_{\text{WC}},\kappa)+\tilde{O}(l\cdot\gamma_{\text{WC}}^{-1}+\kappa^2\cdot\gamma_{\text{WC}}^{-1}).
\end{aligned}\tag{32}$$

We note that independent of which case, for sufficiently long messages, the overhead associated with running OptimisticFlood is only proportional to the extra cost to first try running the best-case protocol and then a minimal additional overhead that matches the asymptotically optimal bound by [LZMT24].

Below, we summarize these results as a theorem.

Theorem 4. *Let OptimisticFlood$(\Pi_{BC}, \Pi_{WC}, c, T, \zeta, \alpha)$ be instantiated with variables as stated in Corollary 1 while minimizing the communication complexity, let α be a implemented by a merkle tree, and let ζ be a (μ, τ)-ECCS be implemented with Reed-Solomon codes.*

If $\gamma_{\textsc{Actual}} \geq \gamma_{BC}$, then

$$\begin{aligned}&\mathsf{PPCC}(\mathit{OptimisticFlood}(\Pi_{BC}, \Pi_{WC}, c, T, \zeta, \alpha), l) \\ &\qquad \leq \mathsf{PPCC}(\Pi_{BC}, l) + \mathsf{PPCC}(\Pi_{WC}, \kappa) + \tilde{O}(l + \kappa^2 \cdot \gamma_{WC}^{-1}), \end{aligned} \tag{33}$$

Further, if $\gamma_{\textsc{Actual}} \geq \gamma_{WC}$, then

$$\begin{aligned}&\mathsf{PPCC}(\mathit{OptimisticFlood}(\Pi_{BC}, \Pi_{WC}, c, T, \zeta, \alpha), l) \\ &\qquad \leq \mathsf{PPCC}(\Pi_{BC}, l) + \mathsf{PPCC}(\Pi_{WC}, l) + \tilde{O}(l \cdot \gamma_{WC}^{-1} + \kappa^2 \cdot \gamma_{WC}^{-1}). \end{aligned} \tag{34}$$

For the best-case, we emphasize that there is only an asymptotic overhead compared to running only the best-case protocol that is directly linear in the message length for messages of length $l = \tilde{\Omega}(\kappa^2 \cdot \gamma_{\mathrm{WC}}^{-1})$. Hence, as noticed in Section 1.2, this allows the protocol to shave off a factor of γ_{WC} when instantiated with asymptotically optimal flooding protocols.

Finally, we note that this protocol is also optimistically responsive in the network delay if the best case protocol is optimistically responsive. I.e. if there is a high fraction of honest parties, then OptimisticFlood propagates the message with the actual delivery time of the best-case protocol.

5 Conclusion

In this work, we presented two new protocols for message dissemination based on a push-pull mechanism. Both are asymptotically optimal in terms of per-party communication complexity. The protocol OptimisticFlood has an even better communication complexity in the best-case, where the fraction of honest parties is high. Furthermore, OptimisticFlood is designed modularly such that it remains provably secure when instantiated with a heuristically optimized best-case protocol with high practical efficiency. This improves the state of the art in theoretical research on message dissemination protocols and at the same time provides a protocol with practical efficiency gains.

Acknowledgements. This work was supported by the Ethereum Foundation, grant number FY24-1527.

References

[AD15] Ittai Abraham and Danny Dolev. Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 605–614. ACM Press, June 2015.

[ANRX21] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: a complete categorization. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *40th ACM PODC*, pages 331–341. ACM, July 2021.

[BGP92] Piotr Berman, Juan A Garay, and Kenneth J Perry. Optimal early stopping in distributed consensus. In *Distributed Algorithms: 6th International Workshop, WDAG'92 Haifa, Israel, November 2–4, 1992 Proceedings 6*, pages 221–237. Springer, 1992.

[Can20] Ran Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020.

[CKMR22] Sandro Coretti, Aggelos Kiayias, Cristopher Moore, and Alexander Russell. The generals' scuttlebutt: Byzantine-resilient gossip protocols. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 595–608. ACM Press, November 2022.

[CKS23] Shir Cohen, Idit Keidar, and Alexander Spiegelman. Make Every Word Count: Adaptive Byzantine Agreement with Fewer Words. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems (OPODIS 2022)*, volume 253 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:21, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Coa93] Brian A Coan. Efficient agreement using fault diagnosis. *Distributed computing*, 7:87–98, 1993.

[DF11] Benjamin Doerr and Mahmoud Fouz. Asymptotically optimal randomized rumor spreading. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part II*, volume 6756 of *LNCS*, pages 502–513. Springer, Berlin, Heidelberg, July 2011.

[DGH$^+$87] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In Fred B. Schneider, editor, *6th ACM PODC*, pages 1–12. ACM, August 1987.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Nielsen and Rijmen [NR18], pages 66–98.

[DKLZ24] Giovanni Deligios, Ivana Klasovita, and Chen-Da Liu-Zhang. Optimal early termination for dishonest majority broadcast. Cryptology ePrint Archive, Report 2024/1656, 2024.

[Doe20] Benjamin Doerr. *Probabilistic Tools for the Analysis of Randomized Optimization Heuristics*, pages 1–87. Springer International Publishing, Cham, 2020.

[DRS82] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. 'Eventual' is earlier than 'Immediate'. In *23rd FOCS*, pages 196–203. IEEE Computer Society Press, November 1982.

[DRS90] Danny Dolev, Ruediger Reischuk, and H Raymond Strong. Early stopping in byzantine agreement. *Journal of the ACM (JACM)*, 37(4):720–741, 1990.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[Edg25] Ben Edgington. Upgrading ethereum: 2.9.2 randomness, 2025. https://eth2book.info/capella/part2/building_blocks/randomness/.

[FOA16] Muntadher Fadhil, Gareth Owenson, and Mo Adda. A bitcoin model for evaluation of clustering to improve propagation delay in bitcoin network. In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pages 468–475, 2016.

[FPRU90] Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Structures & Algorithms*, 1(4):447–460, 1990.

[GM98] Juan A Garay and Yoram Moses. Fully polynomial byzantine agreement for n> 3 t processors in t+ 1 rounds. *SIAM Journal on Computing*, 27(1):247–290, 1998.

[GRPV23] Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. Verifiable Random Functions (VRFs). RFC 9381, August 2023.

[JL24] Marc Joye and Gregor Leander, editors. *EUROCRYPT 2024, Part III*, volume 14653 of *LNCS*. Springer, Cham, May 2024.

[KMG03] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. Parallel Distributed Syst.*, 14(3):248–258, 2003.

[KSSV00] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *41st FOCS*, pages 565–574. IEEE Computer Society Press, November 2000.

[LN24] Julian Loss and Jesper Buus Nielsen. Early stopping for any number of corruptions. In Joye and Leander [JL24], pages 457–488.

[LPR10] João Leitão, José Pereira, and LuÍs Rodrigues. *Gossip-Based Broadcast*, pages 831–860. Springer US, Boston, MA, 2010.

[LZLM$^+$20] Chen-Da Liu-Zhang, Julian Loss, Ueli Maurer, Tal Moran, and Daniel Tschudi. MPC with synchronous security and asynchronous responsiveness. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 92–119. Springer, Cham, December 2020.

[LZMM$^+$22] Chen-Da Liu-Zhang, Christian Matt, Ueli Maurer, Guilherme Rito, and Søren Eller Thomsen. Practical provably secure flooding for blockchains. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 774–805. Springer, Cham, December 2022.

[LZMT24] Chen-Da Liu-Zhang, Christian Matt, and Søren Eller Thomsen. Asymptotically optimal message dissemination with applications to blockchains. In Joye and Leander [JL24], pages 64–95.

[Mer90] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, New York, August 1990.

[MNT22] Christian Matt, Jesper Buus Nielsen, and Søren Eller Thomsen. Formalizing delayed adaptive corruptions and the security of flooding networks. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 400–430. Springer, Cham, August 2022.

[NR18] Jesper Buus Nielsen and Vincent Rijmen, editors. *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*. Springer, Cham, April / May 2018.

[PS18] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Nielsen and Rijmen [NR18], pages 3–33.

[PT84] Kenneth J Perry and Sam Toueg. An authenticated byzantine generals algorithm with early stopping. Technical report, Cornell University, 1984.

[Rei85] Rüdiger Reischuk. A new solution for the byzantine generals problem. *Information and Control*, 64(1-3):23–42, 1985.

[RS60] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[RT19] Elias Rohrer and Florian Tschorsch. Kadcast: A structured approach to broadcast in blockchain networks. In *AFT*, pages 199–213. ACM, 2019.

[TM23] Louis Thibault and Dan Marzec. The hitchhiker's guide to p2p overlays in ethereum, 2023. Accessed: 2024-10-18.

[TPS87] Sam Toueg, Kenneth J Perry, and TK Srikanth. Fast distributed agreement. *SIAM Journal on Computing*, 16(3):445–457, 1987.

[VT19] Huy Vu and Hitesh Tewari. An efficient peer-to-peer bitcoin protocol with probabilistic flooding. In Mahdi H. Miraz, Peter S. Excell, Andrew Ware, Safeeullah Soomro, and Maaruf Ali, editors, *Emerging Technologies in Computing*, pages 29–45, Cham, 2019. Springer International Publishing.