Asymptotically Optimal Message Dissemination with Applications to Blockchains

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

[1]HSLU and Web3 Foundation, Switzerland
chen-da.liuzhang@hslu.ch
[2]Concordium, Zurich, Switzerland
cm@concordium.com
[3]The Alexandra Institute, Aarhus, Denmark
soeren.eller@alexandra.dk

October 11, 2023

Abstract

Messages in large-scale networks such as blockchain systems are typically disseminated using flooding protocols, in which parties send the message to a random set of peers until it reaches all parties. Optimizing the communication complexity of such protocols and, in particular, the per-party communication complexity is of primary interest since nodes in a network are often subject to bandwidth constraints. Previous flooding protocols incur a per-party communication complexity of $\Omega(l \cdot \gamma^{-1} \cdot (\log(n) + \kappa))$ bits to disseminate an l-bit message among n parties with security parameter κ when it is guaranteed that a γ fraction of the parties remain honest. In this work, we present the first flooding protocols with a per-party communication complexity of $O(l \cdot \gamma^{-1})$ bits. We further show that this is asymptotically optimal and that our protocols can be instantiated provably securely in the usual setting for proof-of-stake blockchains.

To demonstrate that one of our new protocols is not only asymptotically optimal but also practical, we perform several probabilistic simulations to estimate the concrete complexity for given parameters. Our simulations show that our protocol significantly improves the per-party communication complexity over the state-of-the-art for practical parameters. Hence, for given bandwidth constraints, our results allow to, e.g., increase the block size, improving the overall throughput of a blockchain.

*The work was partly done while the author was at NTT Research.
†The work was partly done while the author was at Aarhus University.

Contents

1 Introduction 3

1.1 Contributions . . . 4

1.2 Technical Overview . . . 7

1.3 Related Work . . . 10

2 Model and Preliminaries 11

2.1 Parties, Adversary and Communication Network . . . 12

2.2 Primitives . . . 12

2.3 Flooding . . . 13

2.4 Additional Notation . . . 14

2.5 Bounds . . . 14

3 Per-Party Communication Lower Bound 14

4 Warm Up: Optimal Flooding With Constant Diameter and Linear Neighbors 15

5 Optimal Flooding With Logarithmic Neighborhood and Diameter 17

5.1 Weak Flooding . . . 17

5.2 Analysis of FFlood . . . 18

5.3 Flooding Amplification . . . 24

5.4 Communication Complexity of the Combined Protocol . . . 27

6 Flooding in the Weighted Setting 28

7 Security in the UC Model 30

7.1 The UC Model . . . 30

7.2 Flooding as a UC Functionality . . . 31

7.3 Strong Flooding Implies UC Flooding . . . 31

8 Practicality of ECFlood 32

8.1 Setup for Simulations . . . 33

8.2 Estimating Parameters for FFlood . . . 33

8.3 Estimating Parameters for ECFlood . . . 35

8.4 Comparison to State-of-the-Art . . . 37

8.5 Computational Overhead of using a WSCAS and a ECCS . . . 39

9 Conclusion 39

A Channels in UC 43

1 Introduction

Current blockchain protocols rely on the availability of a multicast network that allows any party to communicate with all other parties in the network, and therefore the security and efficiency of the blockchain protocol are heavily influenced by its underlying multicast network.

In typical blockchain protocols, including Bitcoin [Nak08] and Ethereum [W$^+$14], such multicast networks are efficiently implemented via a flooding protocol, which lets the sender select a set of neighbors randomly and forward the message to these parties, who will forward the messages to another randomly chosen set of neighbors and so on. For an l-bit message and n parties of which only $\gamma \cdot n$ are guaranteed to be honest, current provably secure flooding protocols that follows this design [MNT22, LMM$^+$22], incur $\Omega(l \cdot \gamma^{-1} \cdot (\log(n) + \kappa))$ bits of per-party communication to ensure that the message is delivered to all parties with a probability overwhelming in κ. For practical blockchain systems where messages contain large blocks (e.g., around 1MB), the incurred communication constitutes one of the main bottlenecks. As a consequence, many practical systems rely on heuristics which guarantees low bandwidth consumption rather than provable security. In fact, several works have shown such heuristic approaches to be vulnerable to practical attacks [HKZG15, AZV17, MHG18, TCM$^+$20].

A trivial lower bound for the per-party communication complexity of message dissemination protocol is that for the message to reach all honest parties, some party must forward at least as many bits as the length of the message, i.e. the per-party communication must be $\Omega(l)$. This leaves a gap between the lower bounds and what current provably secure flooding protocols achieve. We bridge this gap by firstly proving that, in fact, a per-party communication complexity of $\Omega(l \cdot \gamma^{-1})$ is necessary, and secondly by providing two highly robust flooding protocols with a per-party communication that matches the lower bound. Our protocols require no setup and are practically efficient even for a small number of parties and message length. Moreover, we show how to extend our protocols to the weighted setting, where each party is assigned a positive weight of a certain resource (such as stake), and the adversary can corrupt any set of parties by accumulating any fraction of the total resource. More details follow below.

Model and security definition. We consider flooding protocols that allow any honest party to deliver their messages to all other parties within a certain time, but do not provide any guarantees when the sender is corrupted. This seemingly weak primitive suffices in particular for most blockchain protocols [GKL15, PS17, DGKR18, PS18, CM19, YMR$^+$19].

Our results are secure for a fixed set of parties connected by point-to-point channels, and where messages sent by honest parties are eventually delivered (i.e., asynchronous channels). To compute the concrete delivery time of the overall flooding protocol, we make use of an upper bound on the channel delays.

For simplicity, our protocols are written with respect to a static adversary that can byzantinely corrupt and control *any* fraction $1 - \gamma$ (this can be, e.g., 99.9%) of the parties at the onset of the protocol. However, the protocols can be modified to the setting of delayed adaptive adversaries, where it takes a certain time from when an adversary decides to corrupt a party until the adversary gains control of this party [MNT22], by regularly letting the parties choose a fresh set of neighbors.

Terminology. We use the term *flooding protocol* to describe a protocol where parties can input a message, and as a result, all other parties will receive the message. We use the term *diameter* of a flooding protocol to describe the maximum number of point-to-point channels some message must pass through before delivery is guaranteed to all parties. We use the term *per-party communication* to describe the maximum number of bits any honest party must send in a flooding protocol to deliver a particular message.

1.1 Contributions

Per-party communication lower bound. First, we present a new lower bound that shows that any flooding protocol that ensures delivery of a message of length l to all honest parties, assuming only γ of them remain honest, must have a per-party communication complexity of at least $\Omega(l \cdot \gamma^{-1})$ bits. Concretely, we show that there exists an adversarial strategy obeying the corruption bound that forces the initial sender of the message to send $\Omega(l \cdot \gamma^{-1})$ bits if delivery must be ensured with an overwhelming probability against this adversary.

Warm up: Optimal flooding with a linear neighborhood and constant diameter. We then present a simple protocol ECCast[1], that requires each party to send messages to all other parties, but it achieves a constant diameter of just 2 with the asymptotically optimal per-party communication complexity. The protocol works by letting the sender of a message divide their message into n (the number of parties) different shares using an erasure-correcting code, and then send a unique share to each party. When a party receives such a share, they forward it to *all* other parties. Once a party receives sufficiently many shares, they are able to reconstruct the original message.

Theorem 1 (ECCast (informal))**.** *For n parties, ECCast ensures flooding with asymptotically optimal per-party communication, a diameter of* 2*, and an overwhelming success probability in κ, for messages of length at least $\Omega(n \cdot (\log(n) + \kappa))$.*

Even though this protocol requires each party to send messages to all other parties, we believe that it has wide applications as it allows one to "balance" the incurred communication among parties, at the cost of doubling the diameter (over the naive protocol in which the sender directly sends the whole message to all parties). In fact, independently and concurrently with our work, Kaklamanis, Yang and Alizadeh [KYA22] use such techniques to speed up the Hotstuff consensus protocol [YMR+19].

Optimal flooding with a logarithmic neighborhood and diameter. We then present the protocol ECFlood,[2] which requires each party to connect to only $O(\gamma^{-1} \cdot (\log(n) + \kappa))$ other parties and use only $O(l \cdot \gamma^{-1})$ of per-party communication.

At a high level, the protocol works by letting the sender of a message divide their message into a number of shares μ. Each of these shares will then be sent to a uniformly random subset of parties of size d. When a party receives such a share, they again forward it to a random subset of parties with size d. Once a party receives sufficiently many shares, they can reconstruct the original message.

Theorem 2 (ECFlood (informal))**.** *For n parties and security parameter κ, there are $\mu = O(\log(n) + \kappa)$ and $d = O(\gamma^{-1})$ such that ECFlood ensures asymptotically optimal flooding with a logarithmic diameter and an overwhelming success probability in κ, for messages of length at least $\Omega((\log(n) + \kappa) \cdot (\log(\log(n)) + \kappa))$.*

Note that ECFlood shaves a factor of $\log(n) + \kappa$ the per-party communication over previous best-known constructions. This is done while keeping both the size of the neighborhood and the diameter at the same level as these previously best-known constructions. We further note

[1] ECCast from the use of Erasure-Correcting codes and each party multicasting messages to all parties.

[2] ECFlood from the use of Erasure-Correcting codes in the flooding protocol.

Table 1: Comparison of flooding for messages of length l among n parties where a γ fraction of the parties is guaranteed to remain honest. We consider the maximum number of neighbors, per-party communication, and maximum distance between sender and any honest party (referenced as diameter). For ECFlood to achieve this asymptotic complexity the length of the message must be $l = \Omega((\log n + \kappa)(\log \log n + \kappa))$ and for ECCast it must be $l = \Omega(n \cdot (\log(n) + \kappa))$.

Protocol	Max. neighbors	Per-party comm.	Diameter
[MNT22, LMM+22]	$O(\gamma^{-1} \cdot (\log(n) + \kappa))$	$O(l \cdot \gamma^{-1} \cdot (\log(n) + \kappa))$	$O(\log(n))$
ECFlood	$O(\gamma^{-1} \cdot (\log(n) + \kappa))$	$O(l \cdot \gamma^{-1})$	$O(\log(n))$
Naive	$n-1$	$l \cdot (n-1)$	1
ECCast	$n-1$	$O(l \cdot \gamma^{-1})$	2

that ECFlood requires no trusted setup but merely relies on an erasure-correcting code and a weak cryptographic accumulation scheme, which can be realized efficiently from standard cryptographic assumptions.

We summarize the properties of ECFlood and ECCast and compare them to other flooding protocols with similar robustness in Table 1, where the "naive" protocol refers to the protocol where the sender simply sends the message to all other parties. Note that both protocols are secure for any message size; we only need sufficiently long messages to achieve optimal communication complexity since for very short messages, the cryptographic primitives add a communication overhead.

Concrete efficiency evaluation of ECFlood. While the theoretical analysis of ECFlood shows that our protocol is asymptotically optimal, we use probabilistic simulations to evaluate its practical efficiency. We compare this to the efficiency of the only approach known to be provably-secure against a byzantine adversary – namely to increase the number of parties that each party forward to make the protocol secure against a byzantine adversary.[3] The main results of our simulations are shown in Figure 1. FFlood(k) is the protocol where each party forwards the message to k parties and the parameter k is increased in order to make the error rate drop.[4]

In the figure, two configurations of ECFlood are included where each has a neighborhood of 200 parties. ECFlood(8) is a configuration optimized for redundancy which has a slightly higher latency than FFlood, whereas ECFlood(20) is a configuration optimized for latency. The latter configuration has a latency that is as good as the latency of FFlood.It is noteworthy that while FFlood needs to increase the per-party communication to obtain a constant failure probability with an increasing number of parties the communication of ECFlood remains constant. Our simulations also show that for both configurations of ECFlood that once the per-party communication reaches a small constant factor our protocol virtually eliminates errors whereas the communication of FFlood needs to be increased linearly in κ to obtain an error rate of $2^{-\kappa}$. Further, to virtually eliminate errors, ECFlood needs a communication redundancy of $\sim$12 (and $\sim$25 for the latency optimized version) whereas FFlood needs a redundancy of ≥ 45. Hence, we conclude that our protocol is not only asymptotically optimal but offers *actual* efficiency advantages over state-of-the-art for practical parameters.

Flooding in the weighted setting. We also consider the setting where parties are publicly assigned a positive weight and any arbitrarily small fraction of the cumulated total weight

[3]For a discussion of why other classical approaches fails in the byzantine setting see Section 1.3.

[4]Further details on the general setup for our simulations can be found in Section 8.

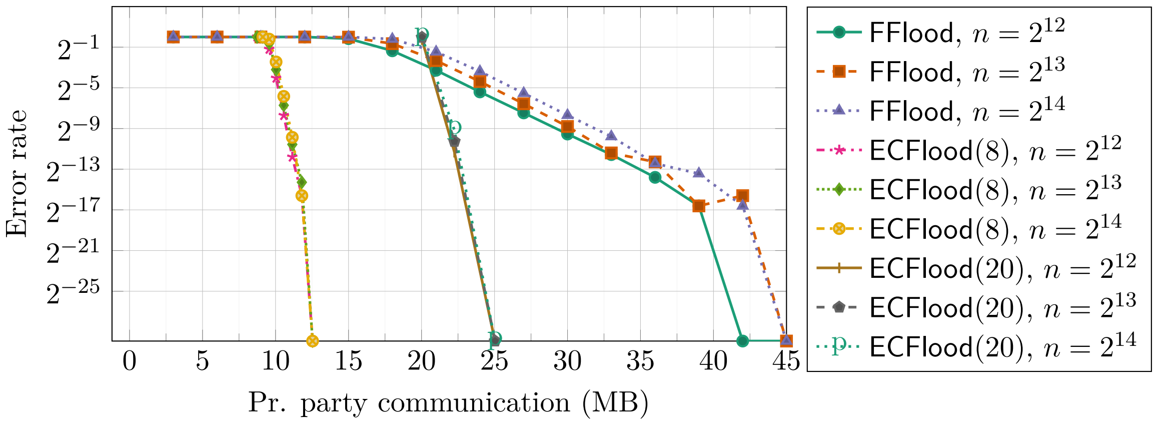

Figure 1: The communication of ECFlood(d) (for $d \in \{8, 20\}$ where d is an internal parameter of the protocol) with a decreasing reconstruction threshold (to lower the error rate), and FFlood(k) with an increasing k (to lower the error rate). The graphs shows the percentages of the simulated executions (on the y-axis which is depicted logarithmicaly) where some party (for a different number of parties n) did not receive the message as a function of the per party communication of the protocol when the protocol sends out a message with size 1 MB, and the accumulators used in ECFlood are implemented via. a Merkle-tree with 256 bit hashes. 100000 simulations of each configuration has been executed for FFlood and for ECFlood. When a point is placed directly on the x-axis this represents the communication needed to make none of the simulations fail.

remains under the control of honest parties. This setting immediately fits the setting of *Proof-of-Stake* protocols [DGKR18, DPS19, CM19], where the stake is publicly available. There are also techniques to estimate the relative computing power in the *Proof-of-Work* setting; we refer to [LMM+22] for an extended discussion.

We leverage the idea of emulation from [LMM+22] to make a general transformation from a flooding protocol that is secure, assuming a fraction of the parties behaves honestly, to a flooding protocol that is secure, assuming that a fraction of publicly assigned weights behaves honestly. We do so by introducing the protocol Flood2WeightedFlood that reduces the task of finding a secure flooding protocol among an actual set of parties to a secure flooding protocol for an emulated set of parties. In more detail, let α_p denote the fraction of total weight assigned to a party p, let there be n parties in total, and let the total fraction of weight given to honest parties be given by $\widetilde{\gamma}$. We observe that by letting each party p emulate $\lceil \alpha_p \cdot n \rceil$ parties, the fraction of these emulated parties that will behave honestly is lower bounded by $\widetilde{\gamma} \cdot 2^{-1}$ and there will be at most $2 \cdot n$ such emulated parties. By instantiating our protocols ECFlood and ECCast such that they are secure assuming only $\widetilde{\gamma} \cdot 2^{-1}$ of the emulated parties remain honest, we thereby allow them work in the weighted setting also. This happens while only increasing the total communication complexity by a factor of at most 4. The per-party communication will, however, in this case, be proportional to the amount of weight each party is assigned. We note that in [LMM+22, Corollary 4], it was shown that it is inherent for the weighted setting that parties with a large fraction of weight must send more messages.

Universal composable security for flooding from delivery guarantees. We prove a general theorem that states that any flooding protocol that guarantees delivery UC [Can20] realizes a flooding functionality. To do so we follow the strategy of Matt et al. [MNT22] by letting the simulator simulate the actual protocol within itself (based on the non-secret inputs of all parties) and adjust the ideal functionality for message delivery to deliver messages according to the simulated actual protocol. Thereby, the security of the protocol is reduced to showing that no messages will be delivered late w.r.t. the delivery guarantee. Our protocols, therefore, benefit from universal composition properties, but for simplicity of presentation, we present them without the overhead of the UC model and focus only on the theorems ensuring that no messages are delivered late.

1.2 Technical Overview

As discussed above, prior works that let each party forward a message to a random subset of neighbors (see, e.g., [MNT22, LMM+22]) need each party to connect to $\Omega(\gamma^{-1} \cdot (\log(n) + \kappa))$ parties to ensure that the message is propagated to all parties with overwhelming probability in the security parameter κ. Intuitively, the term κ is needed to make the probability that an individual party has no honest neighbors negligible. Further, to ensure that the probability that this happens for *any party* is negligible, the additional $\log(n)$ neighbors are needed. However, for each neighbor there is only a γ chance that this neighbor is honest which is why the γ^{-1} factor is necessary.

In fact, it has been shown [KMG03, Section 3.2, p. 6] that the probability that there is an isolated honest party for $n \to \infty$ when $d = \gamma^{-1} \cdot (\log(n \cdot \gamma) + c + o(1))$ for some constant c is given by $1 - e^{-e^{c}}$. It is therefore not possible to decrease the number of needed neighbors for such flooding protocols significantly. In order to further improve the communication complexity, we focus instead on investigating how much we can reduce the number of bits sent by each party.

Per-party communication lower bound. We sketch our lower bound, stating that deliverying an l-bit message requires per-party communication $\Omega(l \cdot \gamma^{-1})$ bits, where γ denotes the fraction of honest parties. To show this, we describe an explicit attack that an adversary corrupting $(1-\gamma)$ fraction of the parties can do to ensure that the sender needs to send $\Omega(l \cdot \gamma^{-1})$ bits with overwhelming probability.

The attack works as follows. Divide the set of parties into γ^{-1} subsets $C_1, \ldots, C_{\gamma^{-1}}$ of size $\gamma \cdot n$ parties each. The adversary chooses an index i uniformly at random and corrupts all parties except the sender s and the parties in C_i. Each corrupted party in any other set $C_j \neq C_i$ ignores all messages that do not come from the sender s or parties in the same set C_j. And moreover, each corrupted party also drops all messages that are sent from C_j to any other outside party $p' \notin C_j \cup \{s\}$.

Intuitively, from the point of view of the honest sender s, it is impossible to identify which C_j is honest, and therefore, *each of these sets* needs to receive full information about the whole message. Moreover, since effectively each set only receives information from the sender s (messages between different sets are ignored or dropped), then the sender needs to transmit the full message to each of these sets, and the total incurred communication is at least $\Omega(l \cdot \gamma^{-1})$.

Optimal per-party communication upper bound. From the feasibility side, we deviate from previous approaches and design our flooding protocol in two steps. First, we consider a so-called *weak* flooding protocol that ensures that for every fixed party, there is a constant probability, that this party receives the message within $O(\log n)$ steps. Secondly, we introduce a compiler that lifts a weak flooding protocol to a (strong) flooding protocol that guarantees delivery to all parties with overwhelming probability.

A weak flooding protocol. Our candidate for a weak flooding protocol is the protocol $\mathsf{FFlood}(d)$ in which every party chooses d random neighbors and forwards each message to those.

As mentioned, [KMG03] showed that for similar protocols, the probability that there is an isolated party for $n \to \infty$ when $d = \gamma^{-1} \cdot (\log(n \cdot \gamma) + c + o(1))$ for some constant c is given by $1 - e^{-e^c}$. This means that one needs to set $d = \Omega(\gamma^{-1} \cdot \log(n \cdot \gamma))$ to have a constant success probability for all parties to receive the message. As a consequence, the expected size of each neighborhood would be $\Omega(\gamma^{-1} \cdot \log(n \cdot \gamma))$, which is too much communication. To overcome this, a novel analysis of the protocol is required.

Obtaining a constant bound on the success probability. Similarly to [LMM+22], we first lower bound the probability of delivering a message to a specific party timely by the probability that this party is reachable within $O(\log n)$ steps from the sender in the communication graph produced by letting only the honest parties forward the message. We then observe that it is sufficient to prove that for an honest sender of a message, there is a constant probability to reach a constant fraction of all honest parties in $O(\log n)$ steps. Since the flooding is completely random, this then implies that any fixed party receives the message with constant probability within $O(\log n)$ steps.

To analyze how many honest parties can be reached in $O(\log n)$ steps, we look at a process where in the first step, the sender sends the message to d random neighbors, and at the ith step, we consider all parties at distance $i - 1$ from the sender each sending the message to d random neighbors. Within each step, we look at the parties in some arbitrary order and consider a party successful, if the d random neighbors of that party contain at least two honest parties that have not been reached, yet. If enough parties are successful, the number of honest parties reached so far increases by some constant > 1 fraction in this step, in which case we consider the step successful. This means after $O(\log n)$ successful steps, we can reach a constant fraction of all honest parties.

We now fix some constant fraction we want to reach. If the targeted constant fraction has already been reached, we can stop. Otherwise, there are enough unreached honest parties left such that for appropriately chosen d independent of n, the probability that at least two of the d neighbors are unreached and honest is at least some positive constant. We note that all constants and parameters need to be set carefully at the end to obtain the desired constant success probability. While the precise calculations are somewhat involved, we intuitively need to set $d = \Theta(\gamma^{-1})$ because when the number of corrupted parties is doubled, this halves the probability that a random neighbor is honest, which can be countered by doubling the number of neighbors.

We next fix some further constant k and look at the first k steps and the remaining ones separately. For the first k steps, we can use a simple union bound to conclude that (for carefully chosen constants) the probability that there exists an unsuccessful party in the process is bounded by a constant. Hence, we have that all the first k steps are successful with constant probability.

It remains to consider the remaining steps. Since there are $O(\log n)$ steps left, it is not sufficient to bound the success probability of each of these steps by a constant. We thus show that the failure probabilities of the remaining steps decrease as the summands in a geometric series to bound the overall failure probability by a constant. We have already established a constant success probability p for each individual party. If there are r parties involved in the current step, the expected number of newly reached honest parties in this step is $2rp$. If we could apply the Chernoff bound to obtain that the actual number is at most a $1 - \delta$ factor away from $2rp$, and thus a constant > 1, except with probability negligible in r, we could conclude the proof, using that r is increasing in every step under the assumption that previous steps are successful. Unfortunately, we cannot directly apply the Chernoff bound here because the success events of the individual parties are not independent: If the first party already reaches many fresh honest parties, there are less left for the next parties, thereby reducing their success probabilities.

We overcome this obstacle by considering a modified experiment as follows: Firstly, when a party reaches more than two fresh honest parties, we only consider two of them as reached and ignore the additional ones. It is clear that this modification can only decrease the probability that there are enough fresh parties in the current step. We then modify the experiment further by *always* adding two additional parties to the set of parties we consider "not fresh and honest". That is, whenever a party is not successful and reaches less than two fresh honest parties, we add one or two extra parties to that set. We do, however, record that this party was not successful. Note that this modification does not increase the success probabilities of any party since removing additional parties from the set of fresh and honest parties can only decrease the success probability of subsequent parties. In this new experiment, the success probabilities of the individual parties are now indeed independent and we can use the Chernoff bound as sketched before to bound the number of successful parties, which concludes the proof.

Flooding amplification. The protocol compiler WeakFlood2Flood splits a message into a number μ of shares using erasure-correcting codes and makes use of a weak flooding protocol to distribute each of these shares independently. Since the shares are created using erasure-correcting codes, it is not necessary for each party to receive all shares to reconstruct the original message. More concretely, consider a reconstruction threshold of $\tau = \xi \cdot \mu$, with constant ξ. Using standard erasure-correcting codes (e.g., Reed-Solomon codes), this can be obtained with a share size of $O(l \cdot \tau^{-1})$, where l is the length of the original message. To achieve a flooding protocol, we then need to ensure that each party receives τ shares.

An apparent attack on such protocol would be for an adversary to try to inject "fake" shares into the set of shares honest parties try to reconstruct the message from. We prevent this by using a cryptographic accumulation scheme, to prove that a particular share is part of the original shares. Such accumulator can be implemented efficiently, e.g., using Merkle trees or signature schemes.

Flooding amplification security proof. We need to show that for appropriately chosen parameters, all parties receive a constant fraction of the shares with overwhelming probability, which allows all parties to reconstruct the original message. We know that the underlying weak flooding protocol ensures that every fixed party receives each individual share with a constant probability. However, even though the honest parties behave independently for flooding the different shares, the events of receiving these shares are not independent. This is because the adversary can, e.g., decide to always deliver some share if another share is delivered, thereby correlating the events. We need to show that the adversary cannot use any correlations to reduce the delivery probability of any of the shares.

To this end, we generalize a result by Maurer, Pietrzak, and Renner [MPR07] to more than two systems. The high-level idea is that we can split the single adversary into smaller adversaries where each of these interacts with a single instance of the weak flooding protocol. Letting the adversaries communicate with each other allows them to jointly emulate an execution of the original adversary and maintain the same advantage in preventing the delivery of the shares. We now inductively reduce the number of messages sent among the adversaries until no communication is needed as follows: The last message is not sent, and the adversary expecting it instead considers all possible messages and behaves as if it received one that maximizes the probability that the delivery guarantee in its flooding instance is violated. Note that this step requires the adversaries to be computationally unbounded, but this is not a problem since security of our weak flooding protocol holds against such adversaries (even though the amplification protocol itself depends on computational assumptions).

Considering the now independent adversaries, we have independent delivery probabilities for the individual shares and can use the Chernoff bound to show that with overwhelming probability, all parties receive sufficiently many shares to reconstruct.

1.3 Related Work

Flooding protocols. Flooding protocols are used to implement so-called multicast networks, which allow a party to distribute a message among a set of parties within some prescribed time. Current flooding protocols (as in Bitcoin [Nak08], Ethereum [W$^+$14], etc.) are typically implemented via a forwarding mechanism, where in order for a party to distribute a message, the party simply selects a random subset of neighbors, who then forward the message to their neighbors and so on.

The security of such a protocol relies on the fact that the graph induced by the neighbor selection procedure among honest parties is connected. Kermarrec, Massoulié and Ganesh [KMG03] showed that when choosing each neighbor with probability ρ in a setting with up to $t = (1-\gamma)\cdot n$ corruptions (out of n parties), it is necessary that $\rho > \frac{\log(n\cdot\gamma)+\kappa}{\gamma\cdot n}$ to ensure that messages are delivered to all honest parties with overwhelming probability in κ.

Matt, Nielsen, and Thomsen [MNT22] formally proved security of such a flooding protocol against a so-called delayed adaptive adversary (where it takes a certain delay for the adversary to gain control over a party) corrupting any fraction of the total number of parties. In a followup work [LMM$^+$22], Liu-Zhang, Matt, Maurer, Rito, and Thomsen gave the first protocol that remains secure in the setting where all parties are publicly assigned a positive weight and the adversary can corrupt parties accumulating up to a constant fraction of the total weight. We adapt the techniques from Liu-Zhang, Matt, Maurer, Rito, and Thomsen and provide a general procedure for obtaining a flooding protocol for the weighted setting from one secure in the none weighted setting. In particular, this allows our protocols to be used in the weighted setting.

The protocols of [MNT22,LMM$^+$22] incur a per-party communication of $O(l\cdot\gamma^{-1}\cdot(\log(n)+\kappa))$ bits, for a message of size l.[5] In contrast, our protocols incur the (asymptotically) optimal per-party communication of $O(l\cdot\gamma^{-1})$.

Coretti, Kiayias, Moore, and Russell [CKMR22] considered the problem of designing a message diffusion mechanism based on the majority of honest stake assumption tailored specifically for the Ouroboros Praos consensus protocol [DGKR18]. However, their flooding protocol achieves a weaker guarantee in that it allows a certain set of honest parties to be eclipsed. In contrast, our work focuses on flooding protocols that guarantee delivery to all honest parties.

Another line of work seeks to improve on the efficiency of flooding protocols for blockchains by applying structured approaches and heuristics [FOA16, RT19, VT19]. However, the behavior of these protocols under byzantine corruptions is not documented, and our focus is on provably secure protocols. We do, therefore, not comment on this line of work further.

Agreement primitives for long messages. A significant line of work is dedicated to building broadcast and Byzantine agreement primitives for long messages for different thresholds, setups, and assumptions, starting from the work of Turpin and Coan [TC84]. Most works in this direction focused on achieving optimal total communication complexity $O(l\cdot n+\text{poly}(n,\kappa))$ (see, e.g., [NRS$^+$20, FH06, GP16, BLZLN22]), for sufficiently large messages. Recently, techniques similar to those we use for our ECCast protocol were used to also achieve low per-party communication

[5] To see this for the work of Matt et al., see [MNT22, Corollary 1, Eq (47)]. To make the failure probability negligible in κ, each party must forward to any other party with probability $\Omega\left(\frac{\log(n)+\kappa}{n\cdot\gamma}\right)$ and hence each party will expectedly have $\Omega(\gamma^{-1}\cdot(\log(n)+\kappa))$ neighbors.

in the context of agreement protocols [NRS+20, LLTW20, YPA+22, GLL+22], and information dispersal protocols [NNT21]. In all these works, however, parties communicate to all other parties (so the neighborhood size is $n-1$). In contrast, we provide ECFlood where each party communicates to only $O(\gamma^{-1} \cdot (\log(n) + \kappa))$ neighbors.

Classic randomized epidemic dissemination. Epidemic algorithms or gossip protocols were first considered for data dissemination by Demers et al. [DGH+87], and have been studied extensively since then, see e.g., [FPRU90, KSSV00, KMG03, DF11] (and more).

[FPRU90] showed that if in each period of time, a rumor is forwarded to a random party, then it takes only $\log(n)$ time before the message has reached all parties (thereby the message complexity becomes $O(l \cdot n \cdot \log(n))$. [KSSV00] extended this to show that if additionally parties that have not heard about a message pull for it (and thereby they assume that it is known that a new message is to arrive) the complexity drops to $O(l \cdot \log(\log(n)) \cdot n)$. [DF11] showed that by slightly "steering" the randomness based upon whether or not the most recent party already had received the message, an asymptotically optimal message complexity of $O(l \cdot n)$ can be achieved.

The big difference between this line of work and our work is that no byzantine adversary is considered and none of these protocols are proven secure against a byzantine adversary. A natural approach to obtain a protocol with reduced communication complexity would be to try to adapt the techniques of this line of work to the byzantine setting. This is however highly non-trivial.

Any protocol which allows parties to pull information from other parties faces the problem that an adversary might issue large amounts of false queries and thereby blow up the communication complexity of the protocol (this would, for example, be a problem for the anti-entropy protocol [DGH+87] and the protocol of [KSSV00]). Further, trying to detect the current state of the network and limit the number of redundant messages based upon this (by either or steering whom to send as in [DF11] to or how many parties to send to as in the rumor mongering of [DGH+87]) inherently has the problem of dishonest parties reporting false information. Finally, note that a very minimal requirement for a protocol to possibly guarantee the delivery of a message is that each honest party must communicate with at least one other honest party. To ensure this with an overwhelming probability in κ when a byzantine adversary controls a constant fraction of the parties each party needs at least κ connections, and if the entire message is forwarded over these channels the communication complexity already becomes $\Omega(\kappa \cdot l \cdot n)$ when sending an l bit message among n parties.

Due to the above difficulties, state-of-the-art in the byzantine setting is simply to increase the number of parties each party forwards the message to, to obtain a secure protocol (as in the protocols of [KMG03, MNT22, LMM+22] and FFlood which we compare our protocol to in Figure 1). New insights are therefore needed to obtain an asymptotically optimal communication complexity in the byzantine setting. The main contribution of our work is to give an efficient protocol for the byzantine setting that achieves exactly such optimal communication complexity and per-party communication.

2 Model and Preliminaries

In this section, we define the model, in which we prove our results, specify the primitives our constructions rely on and give suggestions for how to instantiate these primitives. Additionally, we define notation and basic bounds that we will use for our proofs.

2.1 Parties, Adversary and Communication Network

We consider a fixed set of n parties $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$. For simplicity, we assume an adversary that can statically corrupt a set of parties such that only a subset $\mathcal{H} \subseteq \mathcal{P}$ will behave honestly.[6] The corrupted parties are byzantine, i.e. fully controlled by the adversary and can behave in an arbitrary way. We will use h to denote a bound on the size of $\mathcal{H}$. For the remainder of the paper, we will assume that $|\mathcal{H}| \geq h$. We will use $\gamma := \frac{h}{n}$ to denote the fraction of parties guaranteed to be honest. We will use the execution semantics of UC [Can20] with the notion of time from TARDIS [BDD+21] when reasoning about executions of our protocol. That is no global clock is available but all parties is ensured to be activated in each time step.

We assume that all parties are connected via point-to-point channels, but our protocol does not rely on a synchrony assumption and is secure in a fully asynchronous network. We only use Δ_{NET} to quantify the delivery guarantees of our protocol (parties do not need to know this bound). This is the same as in previous works [MNT22, LMM+22]. In Appendix A, we recap how such channels can modeled in UC.

2.2 Primitives

Erasure correcting codes. In our protocols we make use of a special type of weak error-correcting code that is only able to tolerate a certain number of erasures. We refer to these as erasure-correcting codes.

Definition 1 (Erasure Correcting Code Scheme)**.** Let $\mu \in \mathbb{N}$ be the number of shares, and let $\varrho \in \mathbb{N}$ be the number of erasures that are to be tolerated. A pair of algorithms ζ is a (μ, ϱ)-erasure-correcting-code-scheme (abbreviated (μ, ϱ)-ECCS) if it consists of two algorithms:

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

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

Standard Reed-Solomon codes [RS60] can be used to instantiate a (μ, ϱ)-ECCS efficiently in a straightforward manner for arbitrary message lengths. That is, to share a message m with length l into μ shares such that $\mu - \varrho$ shares can be used to reconstruct the original message, we choose the symbol size $a = \lceil \log(\mu) \rceil$ (in bits) and split the message into chunks $c_1, \ldots, c_{\lceil \frac{l}{a \cdot (\mu - \varrho)} \rceil}$ that each has size $a \cdot (\mu - \varrho)$. Because each of these chunks consists of $\mu - \varrho$ symbols, we can use Reed-Solomon encoding to for chunk c_i obtain micro shares $s'_{i,1}, \ldots s'_{i,\mu}$ where any $\mu - \varrho$ of the micro shares can be used to obtain the entire chunk c_i via. Reed-Solomon decoding. We now define shares $s_1, \ldots, s_\mu$ by letting $s_j := s'_{1,j}, \ldots, s'_{\lceil \frac{l}{a \cdot (\mu - \varrho)} \rceil, j}$. Obtaining any $\mu - \varrho$ of such shares allows to reconstruct all chuncks using Reed Solomon decoding. Thereby, the original message can be reconstructed.

Let ζ be this scheme, and we will have that

$$\zeta.\texttt{ShareSize}(l) = O\Big(\frac{l}{\mu - \varrho}\Big). \tag{1}$$

[6]One can extend our protocols to handle so-called delayed adaptive adversaries, using techniques presented in [MNT22].

This implies that the total bitlength of the shares will be $O(l \cdot \frac{\mu}{\mu-\varrho})$. It has been shown that the encoding and decoding of such codes be done in $O(2^a \cdot a^2)$ time [Did09]. This means that the entire encoding and decoding for this scheme can be done in just $O(\mu^2 \cdot (\log(\mu)^2))$ time. For further comments on the practicality of this see Section 8.5.

Weak cryptographic accumulators. We will in the paper make use of a *weak* version of a *static positive* accumulator. *Weak* refers to that we only require collision-freeness and correctness to hold for honestly generated accumulators, *static* refers to that we do not need the set of accumulated values to be dynamically extendable, and *positive* means that we only need to prove membership of an accumulator (in particular we do not need to prove that an element is not a part of the accumulator).

Definition 2 (Weak Static Cryptographic Accumulation Scheme)**.** A *weak static cryptographic accumulation scheme* (abbreviated WSCAS) α consists of two algorithms:

- $\alpha.\mathsf{Accumulate}(\{m_1, \dots, m_\eta\})$: A PPT algorithm for accumulating a set of values $\{m_1, \dots, m_\eta\}$. It returns an accumulated value z and a sequence of proofs $\pi_1, \dots, \pi_\eta$ where π_i can be used to prove that m_i is in the accumulated value z where each $m_i \in \{0,1\}^*$.
- $\alpha.\mathsf{Verify}(m, \pi, z)$: A function that checks if a proof π 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 PPT adversary can find a set of values $M := \{m_1, \dots, m_\eta\}$, a value $m' \notin M$, and a proof π such that $\alpha.\mathsf{Verify}(m', \pi, z) = \top$ for $z \leftarrow \alpha.\mathsf{Accumulate}(M)$ except with negligible probability.

See [BP97] for the original formal definition of collision-freeness and [ÖMBS21] for an overview of accumulator constructions. We use the notation $\alpha.\mathtt{AccSize}$ for a bound on the size of the accumulated value and $\alpha.\mathtt{ProofSize}(\eta)$ for a function that bounds the size of each proof as a function of the number of messages accumulated η.

Because we only require collision-freeness for honestly generated accumulators, a WSCAS scheme can be efficiently instantiated using a regular signature scheme by letting the accumulated value z be the public verification key, and a proof for a message be a signature of that message. For suitable signature schemes:

$$\alpha.\mathtt{AccSize} = O(\kappa) \text{ and } \alpha.\mathtt{ProofSize}(\eta) = O(\kappa). \tag{2}$$

The same complexity can be achieved by basing α on RSA accumulators [BdM93] or bilinear accumulators [Ngu05]. To avoid generating keys or a setup assumption, one can also use Merkle Trees [Mer89] as accumulators, at the cost of slightly increasing the proof size to $\alpha.\mathtt{ProofSize}(\eta) = O(\log(\eta) \cdot \kappa)$.

2.3 Flooding

A flooding protocol allows a set of parties to send messages to each other subject to certain delivery guarantees. Our definition is based on the one presented in [LMM+22] with minor differences.

Definition 3 (Flooding)**.** Let Π be a protocol executed by parties $\mathcal{P}$, where each party $p \in \mathcal{P}$ can input a message at any time, and as a consequence, all parties get a message as output. We say that Π is a *strong* Δ-flooding protocol if when a message m is input to an *honest* party at time t, then by time $t + \Delta$ there is a probability overwhelming in the security parameter κ that all other honest parties output m.

Note that our definition allows a message sent by the adversary to be only received by a subset of honest parties. This is sufficient for most blockchain protocols [GKL15, PS17, DGKR18, PS18, CM19, YMR+19]. If total delivery from dishonest senders is required, one can simply let the honest parties re-distribute the received messages.

Preventing denial-of-service in a flooding network. As noted in [LMM+22], there exists a trivial denial-of-service attack against flooding networks because they allow to flood *any* message. An adversary can simply input a large number of arbitrary messages and as all messages must be propagated by the definition of a flooding network, this will allow an adversary to exhaust the bandwidth of honest parties. A simple solution to this is to only let honest parties forward "valid" messages (w.r.t. some validity predicate determined by upper-level protocols). Such an approach can also be applied to our setting. One can add an extra field of data to the shares sent around by our FFlood protocol, and parties will then verify that this extra data proves that this share should be forwarded. In the context of for example proof-of-stake blockchains, such extra data can for example simply be a signature by the baker of the block and a proof that this baker is allowed to create a block. For clarity of presentation, we have left this out of our presentation as the details of such validity predicates necessarily must be determined by upper-level protocols.

2.4 Additional Notation

We use the notation $\Gamma_s^\lambda(G)$ for the set of neighbors of a party (usually the sender) s at distance at most λ in a graph G. When clear, we omit both s and G for this set and merely write Γ^λ. We write $A \overset{\$}{\leftarrow} \mathcal{D}$ to sample the value A from the distribution $\mathcal{D}$. We let $\mathcal{U}(A)$ denote the uniform distribution on a set A. We denote by $\log x$ the natural logarithm of x. For two random variables X and Y, we will write $X \preceq Y$ if Y stochastically dominates X, i.e. $\Pr[Y \geq k] \geq \Pr[X \geq k]$ for all k.

2.5 Bounds

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 $\mu \leq E[\sum_{i=1}^n X_i]$. We then have for all $\delta \in [0, 1]$,*

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

3 Per-Party Communication Lower Bound

In this section, we present and prove a new lower bound that states that any protocol must have a per-party communication of at least $\Omega(l \cdot \gamma^{-1})$ when sending a message of length l.

Theorem 3. *For any Δ, any protocol that is Δ-flooding protocol must have per party communication complexity $\Omega(l \cdot \gamma^{-1})$ when sending a message of length l.*

Proof. Let $\Delta \in \mathbb{N}$. For the sake of contradiction let us assume that there exists a protocol Π with $o(l \cdot \gamma^{-1})$ per-party communication complexity when sending a message of length l.

We select a party s as the initial sender and reason about an execution of the protocol where a message of length l is input to s against a specific adversarial strategy. Before the protocol execution starts the adversary divides the set of parties without the sender $\mathcal{P} \setminus \{s\}$ into sets $C_1, \ldots, C_{\lfloor \gamma^{-1} \rfloor}$ such that for all i we have that $C_i \geq n \cdot \gamma - 1$. At random the adversary now chooses a $j \in \{1, \ldots, \lceil \gamma^{-1} \rceil\}$ and corrupts all sets C_i where $i \neq j$. This is possible because for any z we have

$$|C_z \cup \{s\}| \geq \gamma \cdot n. \tag{3}$$

In particular, this holds for C_j, and therefore the corruption threshold is not exceeded. The adversary now lets the corrupted parties in each of the sets C_i execute the protocol Π with the following modifications:

- When a party $p \in C_i$ receives a message from a party $p' \notin C_i \cup \{s\}$, the party p ignores the message and acts as if they had not received it at all.
- Whenever the protocol dictates that a party $p \in C_i$ should send a message to party $p' \notin C_i \cup \{s\}$, the message is dropped and not send.

Now note that from the perspective of the sender, s it is impossible to distinguish which C_i is honest and which is corrupted, as it cannot be distinguished from the sender side whether a message is dropped on the sending side or the receiving side, and by Equation (3), it could be for any i that each set C_i is actually the only set of honest. If there exists a k such that C_k has a constant probability of receiving less than l bits from the sender, then there is at least $\lfloor \gamma^{-1} \rfloor^{-1}$ probability (because the non-corrupted set where selected uniformly at random) that $k = j$ and hence the protocol would fail with a non-negligible probability. Therefore, the protocol must, with overwhelming probability, let the sender send each set of parties at least l bits as they would otherwise not be able to deliver the message to all honest parties with overwhelming probability. Therefore, with overwhelming probability at least $l \cdot \lfloor \gamma^{-1} \rfloor = \Omega(l \cdot \gamma^{-1})$ bits are sent by the sender. This contradicts that a protocol with $o(l \cdot \gamma^{-1})$ per party communication complexity exists. □

4 Warm Up: Optimal Flooding With Constant Diameter and Linear Neighbors

In this section, we present our protocol ECCast and show that it is a flooding protocol with a maximum per-party communication of $O(l \cdot \gamma^{-1})$, a total communication complexity of $O(l \cdot \gamma^{-1} \cdot n)$, and a diameter of 2.

Our protocol ECCast is parameterized by an erasure correcting code scheme that shares a message into n shares and a cryptographic accumulator. When a sender wishes to send, they will share the message into n shares and send a unique share to each party. When a party receives such a share, they will forward the share they receive to *all* other parties. This will ensure that each party ends up receiving as least as many shares as there are honest parties. Therefore, the only thing that can prevent honest parties from reconstructing the original message is if they try to reconstruct from some shares that were not sent by the original sender. To prevent this, we use the cryptographic accumulator.

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

The protocol is parameterized by, a (n, ϱ)-ECCS ζ for some $\varrho \in \mathbb{N}$, and a cryptographic accumulator α. Each party $p_i \in \mathcal{P}$ keeps track of a set of shares received for a particular accumulator z, $\texttt{ReceivedShares}_i[z]$. Additionally, each party p_i keeps track of a set of received messages $\texttt{Received}_i$.

Initialize: Initially, each party p_i sets $\texttt{ReceivedShares}_i := \varnothing$, and $\texttt{Received}_i := \varnothing$.

Send: When p_i receives (Send, m) they share the message m into shares $\zeta.\mathsf{Enc}(m) = s_1, \ldots, s_n$. Furthermore, they obtain an accumulated value and proofs for each share and its share number $z, \pi_1, \ldots, \pi_n = \alpha.\mathsf{Accumulate}(\{(s_j, j) \mid 1 \leq j \leq n\})$. For $1 \leq j \leq n$, the party now sends $(\mathit{Forward}, s_j, j, \pi_j, z)$ to party p_j using the point-to-point channel between them. Finally, they add m to $\texttt{Received}_i$.

Get Messages: When p_i receives $(\mathit{GetMessages})$ they return $\texttt{Received}_i$.

When party p_i receives a tuple (T, s, j, π, z) over a point-to-point channel where $\alpha.\mathsf{Verify}((s, j), \pi, z) = \top$ they add (s, j) to $\texttt{ReceivedShares}_i[z]$. Furthermore, p_i does the two following checks:

- If $|\texttt{ReceivedShares}_i[z]| \geq n - \varrho$, then they
  1. Obtain a sequence of shares $s_1, \ldots, s_n$ by letting $s_j = s$ if $(s, j) \in \texttt{ReceivedShares}_i[z]$ and otherwise sets $s_j = \bot$ (i.e. if no such pair is in $\texttt{ReceivedShares}_i[z]$).
  2. Decode the shares and add the recovered message to the set of received messages, $\texttt{Received}_i := \texttt{Received}_i \cup \{\zeta.\mathsf{Dec}(s_1, \ldots, s_n)\}$.
- If $T = \mathit{Forward}$, it is the first time they receive (T, s, j, π, z), and $j = i$, then they send $(\mathit{Receive}, s, j, \pi, z)$ to all parties over their respective point-to-point channels.

Below, we state and prove that ECCast is flooding protocol.

Theorem 4. *Let $\varrho \geq n \cdot (1 - \gamma)$, let ζ be a (n, ϱ)-ECCS, and let α be a WSCAS, then the protocol $\mathsf{ECCast}(\zeta, \alpha)$ is a strong $(2 \cdot \Delta_{\text{NET}})$-flooding protocol.*

Proof. We consider an execution with a PPT adversary $\mathcal{A}$, where a message m is input to some honest party s at time t, and let the accumulated value that is sent out be z. The delivery guarantees for the underlying point-to-point channels ensures that at latest at time $t + \Delta_{\text{NET}}$ any honest party p_i will have received a $(\mathit{Forward}, s_i, i, r, \pi_i, z)$ s.t. $\alpha.\mathsf{Verify}((s, i), \pi_i, z) = \top$ (by correctness of the WSCAS). This implies that this is the latest point any honest party will forward $(\mathit{Receive}, s_i, i, \pi, z)$ to all other parties. By the delivery guarantees of the point-to-point channels, these messages will be delivered at the latest at time $t + 2 \cdot \Delta_{\text{NET}}$. Hence, if p_i is an honest party, then the size of $\texttt{ReceivedShares}_i[z]$ will be at least $n \cdot \gamma = n - n \cdot (1 - \gamma) \geq n - \varrho$. Therefore any honest party p_i will be able to reconstruct the original message at the latest at time $t + 2 \cdot \Delta_{\text{NET}}$ unless there are some tuple (s_j, j) and π where $\alpha.\mathsf{Verify}((s, j), \pi, z) = \top$ and where s_j is not equal to an original share sent out by the sender s. To bound the probability that this happens, we construct another PPT adversary $\mathcal{A}'$ that emulates the execution of the protocol against adversary $\mathcal{A}$ in order to break the collision-freeness of the WSCAS. Whenever one of the emulated parties receives a tuple with (s_j, j) and π where $\alpha.\mathsf{Verify}((s, j), \pi, z) = \top$ and where s_j is not equal to the original share sent out by the sender, the adversary $\mathcal{A}'$ successfully finds a collision. Hence, this can only happen with a negligible probability.

□

Communication complexity of ECCast. We now analyze the communication complexity of $\mathsf{ECCast}(\zeta, \alpha)$ (for ζ and α instantiated as suggested by Theorem 4) when a message of length l is input. The neighborhood of each party is n as all parties will talk to all other parties. The per-party communication is given by the size of the neighborhood times the size of the tuple sent over each point-to-point channel. As each tuple consists of a bit (*Forward* or *Receive*), a share, a sequence number of the share, an accumulator proof, and an accumulated value, we have that the communication for each party is given by

$$n \cdot (1 + \zeta.\mathtt{ShareSize} + \log(n) + \alpha.\mathtt{ProofSize} + \alpha.\mathtt{AccSize}).$$

If we instantiate the ECCS with Reed-Solomon codes, we get that the size of each share is bounded by $\zeta.\mathtt{ShareSize} = O(l \cdot (\gamma \cdot n)^{-1})$. By using an efficient WSCAS α with size of the accumulated value and proof $O(\kappa)$ (see Equation (2)), we get that the communication of each party is bounded by

$$n \cdot (1 + O(l \cdot (\gamma \cdot n^{-1}) + \log(n) + O(\kappa) + O(\kappa)) = O(l \cdot \gamma^{-1} + n \cdot (\log(n) + \kappa)).$$

This is optimal when $l = \Omega(n \cdot (\log(n) + \kappa))$ by Theorem 3.

5 Optimal Flooding With Logarithmic Neighborhood and Diameter

We show a flooding protocol with (asymptotically) optimal communication complexity, in two steps. First, we define a weaker notion, denoted *weak flooding* and propose an instantiation of it. Then we show how to lift the security guarantees from a weak flooding protocol to achieve a full-fledged flooding protocol.

5.1 Weak Flooding

A weak flooding protocol is a flooding protocol that, instead of being guaranteed to deliver all messages to all parties, only ensures that there is a lower bound on the probability that each party receives a message.

Definition 4 (Weak Flooding)**.** Let Π be a protocol 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. We say that Π is a *weak* (Δ, ξ)-flooding protocol if at any time t when a message m is input to some honest party, then it must be that for any $p_i \in \mathcal{H}$

$$\Pr[p_i \text{ receives } m \text{ at latest at time } t + \Delta] \geq \xi.$$

Protocol description. We now describe a simple flooding protocol, in which each party samples a random set of d neighbors for some parameter d and relays all new messages to all their neighbors.

Protocol FFlood(d)

Each party $p_i \in \mathcal{P}$ keeps track of a set of relayed messages $\texttt{Relayed}_i$ which will also be used to keep track of which messages party p_i has received.

Initialize: Initially, each party p_i sets $\texttt{Relayed}_i := \varnothing$ and samples a uniform random set $N_i \subseteq \mathcal{P}, |N_i| = d$ of d neighbors.

Send: When p_i receives (Send, m), they forward the message to all parties in N_i. Finally, they set $\texttt{Relayed}_i := \texttt{Relayed}_i \cup \{m\}$.

Get Messages: When p_i receives $(\mathit{GetMessages})$, they return $\texttt{Relayed}_i$.

When party p_i receives message m on a point-to-point channel where $m \notin \texttt{Relayed}_i$, party p_i continues as if they had received (Send, m).

We prove the following theorem in Section 5.2. Note that we explicitly quantify over the number of parties n after the existential quantification of the success probability bound, to highlight that the probability is independent of the number of parties.

Theorem 5. *There exists $\xi \in (0, 1]$ such that for any $n \geq 50 \cdot \gamma^{-1}$ there is a $d = O(\gamma^{-1})$ and $\Delta = O(\log(n) \cdot \Delta_{\text{NET}})$ such that the protocol FFlood(d) is a weak (Δ, ξ)-flooding protocol. The security also holds against computationally unbounded adversaries.*

Previous analyses of related protocols [KMG03, MNT22] only considered the probability to deliver the message to *all honest parties*, and for this required $d = \Omega(\gamma^{-1} \cdot \log(\gamma \cdot n))$ to obtain a constant probability. Our analysis instead proves that $d = O(\gamma^{-1})$ is sufficient to guarantee that *any fixed party* receives the message with constant success probability.

5.2 Analysis of FFlood

In order to prove that FFlood is a weak flooding protocol, we have to prove that for any party, the probability that this party receives a specific message is constant. We do so by re-using the honest sending process from [LMM+22]. The idea of the honest sending process, which we recap below ,is to let it mimic the communication graph induced by the sending message where only the honest parties participate in the distribution of the message, and the adversary only delivers the message on the point-to-point channels at the latest point in time possible.

Definition 5 (The honest sending process)**.** Let $s \in \mathcal{H}$ be an honest party, let d be a neighborhood size, and let $\lambda \in \mathbb{N}$ be a distance. We let the *honest sending process*, $\mathsf{HSP}(s, d, \lambda)$, be a random process that returns a directed graph $G = (\mathcal{V}, E)$ defined by the following random procedure:

1. Initially, $E := \varnothing$. Furthermore, we keep track of a set $\texttt{Flipped} := \varnothing$ that consists of nodes that have already had their outgoing edges decided, and a first-in-first-out queue $\texttt{ToFlip} := \{(s, 0)\}$ of nodes and their distance from p that are to have their edges decided.
2. The process proceeds with taking out the first element of $\texttt{ToFlip}$ until $\texttt{ToFlip} == \varnothing$. Let the element that is taken out of $\texttt{ToFlip}$ be denoted by (p', i) and do the following:
   (a) Let N be a uniformly random subset of $\mathcal{P}$ with d elements, and set $N := N \cap \mathcal{H}$.
   (b) Update the set of edges $E := E \cup \{(p', p'') \mid p'' \in N\}$ and let $\texttt{Flipped} := \texttt{Flipped} \cup \{p'\}$.
   (c) If $i + 1 < \lambda$, for all $p'' \in N \setminus (\texttt{Flipped} \cup \texttt{ToFlip})$, add $(p'', i + 1)$ to $\texttt{ToFlip}$.

3. Finally, return $G = (\mathcal{H}, E)$.

The following lemma relates the probability of a party being in the neighborhood of the sender in the graph produced by the honest sending process with FFlood being a weak flooding protocol. The proof is analogous to the one of [LMM+22, Lemma 5 on p. 16] and therefore omitted.

Lemma 2. *Let $\lambda \in \mathbb{N}$ be a distance, let $d \in \mathbb{N}$, and let $\Delta := \lambda \cdot \Delta_{NET}$. Further, let $s_{min} \in \mathcal{H}$ and $p_{min} \in \mathcal{H}$ s.t. when $G \xleftarrow{\$} \mathit{HSP}(s_{min}, d, \lambda)$ then $\Pr[p_{min} \in \Gamma^{\lambda}_{s_{min}}(G)]$ is minimized over all such $s, p \in \mathcal{H}$. If*

$$\xi \leq \Pr[p_{min} \in \Gamma^{\lambda}_{s_{min}}(G)],$$

then $\mathit{FFlood}(d)$ is a weak (Δ, ξ)-flooding protocol.

Next, we lower bound for any party the probability that this party is in a logarithmic neighborhood of the sender in the honest sending process.

Lemma 3. *Let $\alpha, \delta \in [0, 1]$. Further, let $\phi \in \mathbb{R}$ be an expected expansion factor, $d, k \in \mathbb{N}$, and let $\lambda := \frac{\log(\alpha \cdot |\mathcal{H}|)}{\log((1-\delta)\cdot\phi)}$. Finally, let $s \in \mathcal{H}$ and let $G \xleftarrow{\$} \mathit{HSP}(s, d, \lambda)$. If $n \geq 11 \cdot \gamma^{-1}$, $\alpha < 1/3$,*

$$\phi \leq 2 - 2(d+1) \cdot \left(\frac{11}{11-\gamma} \cdot (1 - (1-3\alpha)\cdot\gamma)\right)^{d-1}, \tag{4}$$

and

$$(1-\delta)\cdot\phi \geq \frac{3}{2}, \tag{5}$$

then for any party $p \in \mathcal{H}$

$$\Pr[p \in \Gamma^{\lambda}_{s}(G)] \geq \frac{\alpha \cdot |\mathcal{H}| - 1}{|\mathcal{H}|} \cdot \left(1 - \frac{d^k - 1}{d-1}(d+1) \cdot \left(\frac{11}{11-\gamma} \cdot (1-(1-3\alpha)\cdot\gamma)\right)^{d-1} - \frac{e^{-\frac{\delta^2 \phi \cdot ((1-\delta)\cdot\phi)^k}{20}}}{1 - e^{-\frac{\delta^2 \phi \cdot ((1-\delta)\cdot\phi)^k}{4}}}\right).$$

Proof. We let Γ^{λ} be the set of neighbors of the sender s at a distance at most λ in G. Note that since G is here the result of the honest sending process, all nodes in Γ^{λ} are honest. We note that the probability that p is in the close neighborhood of the sender is lower bounded by the probability that this is the case *and* the sender has a large honest neighborhood $|\Gamma^{\lambda}| \geq \alpha \cdot |\mathcal{H}|$:

$$\Pr\left[p \in \Gamma^{\lambda}\right] \geq \Pr\left[p \in \Gamma^{\lambda} \cap \alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right] = \Pr\left[p \in \Gamma^{\lambda} \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right] \cdot \Pr\left[\alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right]. \tag{6}$$

We next bound these two probabilities individually.

We first bound the probability that party p is in the set of neighbors, i.e., $p \in \Gamma^{\lambda}$ given the set of neighbors is "large", $\alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|$. For this, we use that all honest parties have an equal probability of appearing inside Γ^{λ} (except the sender, who is always there) and the law of total probability.

$$\begin{aligned}
&\Pr\left[p \in \Gamma^{\lambda} \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right] \\
&= \sum_{a=\lceil \alpha \cdot |\mathcal{H}| \rceil}^{|\mathcal{H}|} \Pr\left[p \in \Gamma^{\lambda} \mid |\Gamma^{\lambda}| = a\right] \cdot \Pr\left[|\Gamma^{\lambda}| = a \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right] \\
&\geq \sum_{a=\lceil \alpha \cdot |\mathcal{H}| \rceil}^{|\mathcal{H}|} \frac{a-1}{|\mathcal{H}|} \cdot \Pr\left[|\Gamma^{\lambda}| = a \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right] \\
&\geq \frac{\alpha \cdot |\mathcal{H}| - 1}{|\mathcal{H}|} \cdot \sum_{a=\lceil \alpha \cdot |\mathcal{H}| \rceil}^{|\mathcal{H}|} \Pr\left[|\Gamma^{\lambda}| = a \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right] \\
&= \frac{\alpha \cdot |\mathcal{H}| - 1}{|\mathcal{H}|}.
\end{aligned} \tag{7}$$

We now bound the probability that the neighborhood of the sender is “large”, i.e., the event $\alpha \cdot |\mathcal{H}| \geq |\Gamma^\lambda|$. To do so, we use some ideas of a proof by Matt et al. [MNT22, Lemma 3 on p. 23-25], but look at a more specific setting and extend this with a new result about the expansion of constant out-degree graphs. Further, we only reuse parts of the proof as we are only interested in bounding the neighborhood of the sender and not the neighborhoods of all parties.

We define $D \coloneqq (|\Gamma^\lambda| < \alpha \cdot |\mathcal{H}|)$, i.e., the event that the sender does not reach at least a α-fraction of the honest nodes within λ steps. We have that

$$\Pr\left[\alpha \cdot |\mathcal{H}| \leq |\Gamma^\lambda|\right] = 1 - \Pr[D], \tag{8}$$

and hence it is sufficient to bound $\Pr[D]$. We let θ^ν be the set of honest parties that are at exactly distance ν from the sender, and define the parties at exactly distance 0 to be the sender, $\theta^0 \triangleq \{s\}$. We now define a series of events. First, we define the events that the number of honest neighbors at distance $\nu + 1$ is strictly more than $(1 - \delta) \cdot \phi$ times the number of neighbors at distance ν:

$$B_\nu \coloneqq \left(|\theta^{\nu+1}| > (1 - \delta) \cdot \phi \cdot |\theta^\nu|\right).$$

Furthermore, we define the event that there are at least $\alpha \cdot |\mathcal{H}|$ neighbors within distance ν:

$$C_\nu \coloneqq \left(|\Gamma^\nu| \geq \alpha \cdot |\mathcal{H}|\right).$$

As convenient notation we let $A_\nu \coloneqq B_\nu \vee C_\nu$ for $\nu = 0, \ldots, \lambda - 1$. Now,

$$\lambda = \frac{\log(\alpha \cdot |\mathcal{H}|)}{\log((1 - \delta) \cdot \phi)} \quad \Longleftrightarrow \quad ((1 - \delta) \cdot \phi)^\lambda = \alpha \cdot |\mathcal{H}|.$$

Therefore, if $B_0, \ldots, B_{\lambda-1}$ hold, then

$$|\Gamma^\lambda| = \sum_{\nu=0}^{\lambda} |\theta^\nu| \geq |\theta^\lambda| \geq ((1 - \delta) \cdot \phi)^\lambda \cdot |\theta^0| = \alpha \cdot |\mathcal{H}|.$$

Furthermore, if just some C_ν holds, then we get that $|\Gamma^\lambda| \geq \alpha \cdot |\mathcal{H}|$. Therefore, we have that

$$\left(\bigwedge_{\nu=0}^{\lambda-1} A_\nu \Longrightarrow \neg D\right) \quad \Longleftrightarrow \quad \left(D \Longrightarrow \bigvee_{\nu=0}^{\lambda-1} \neg A_\nu\right).$$

Hence, we get the following:

$$\begin{aligned}
\Pr[D] &\leq \Pr\left[\bigcup_{i=0}^{\lambda-1} \neg A_i\right] \\
&\leq \sum_{i=0}^{\lambda-1} \Pr\left[\neg A_i \,\middle|\, \bigcap_{j<i} A_j\right] \\
&= \sum_{i=0}^{\lambda-1} \Pr\left[\neg B_i \cap \neg C_i \,\middle|\, \bigcap_{j<i} A_j\right] \\
&\leq \sum_{i=0}^{\lambda-1} \Pr\left[\neg B_i \,\middle|\, \bigcap_{j<i} A_j \cap \neg C_i\right] \\
&= \sum_{i=0}^{\lambda-1} \Pr\left[\neg B_i \,\middle|\, \bigcap_{j<i} B_j \cap \neg C_i\right].
\end{aligned} \tag{9}$$

We now state and prove a bound on the individual probabilities inside the sum.

Claim 1 (Fast expansion to small fraction). *For $n \geq 11 \cdot \gamma^{-1}$, $\alpha < 1/3$, $\phi \leq 2 - 2(d+1) \cdot \left(\frac{11}{11-\gamma} \cdot (1-(1-3\alpha)\cdot\gamma)\right)^{d-1}$, and for any $\nu \in \{1, \dots, \lambda - 1\}$, we have*

$$\Pr\left[\neg B_\nu \;\middle|\; \bigcap_{j<\nu} B_j \cap \neg C_\nu\right] \leq \min\left\{e^{-\frac{\delta^2 \phi \cdot ((1-\delta)\cdot\phi)^\nu}{4}}, d^\nu \cdot (d+1) \cdot \left(\frac{11}{11-\gamma} \cdot (1-(1-3\alpha)\cdot\gamma)\right)^{d-1}\right\}.$$

Proof. We look at the probability space where $\bigcap_{1 \leq j < \nu} B_j \cap \neg C_\nu$ holds. We define the number of honest parties that are reachable at a distance ν to be $r := |\theta^\nu|$ and let the number of honest parties that have not been reached so far be $U := \mathcal{H} \setminus \Gamma^\nu$.

Now order the parties in θ^ν in some arbitrary way and let N_i be the set of neighbors chosen by the i-th party in θ^ν. Consider the following process: $S_0 := \varnothing$, and for $i > 0$, we let $N_i' := N_i \cap (U \setminus S_{i-1})$ and consider two cases: If $|N_i'| < 2$ (i.e., the i-th party has not chosen at least two "new" parties as neighbors), then $S_i = S_{i-1} \cup N_i'$ and $X_i := 0$. Otherwise, let s_1, s_2 be two uniformly random elements in N_i', define $S_i = S_{i-1} \cup \{s_1, s_2\}$, and $X_i := 1$. Note that

$$|\theta^{\nu+1}| \geq |S_r| \geq 2 \sum_{i=1}^{r} X_i.$$

Observing that adding more elements to S_i decreases the probability of X_{i+1}, we define the following related process that always adds two elements in each step: Let $\tilde{S}_i$, $\tilde{N}_i'$, and $\tilde{X}_i$ be defined analogous to S_i, N_i', and X_i, except that if $|N_i \cap (U \setminus \tilde{S}_{i-1})| < 2$, then we add one or two random elements from $U \setminus \tilde{S}_{i-1}$ to $\tilde{N}_i'$ such that we *always* have $|\tilde{N}_i'| = 2$. We then have

$$\sum_{i=1}^{r} \tilde{X}_i \leq \sum_{i=1}^{r} X_i \leq \frac{|\theta^{\nu+1}|}{2}. \tag{10}$$

Furthermore, the $\tilde{X}_i$ are independent. We now upper bound the probability that $\tilde{X}_i = 0$. Let $F_i := (\mathcal{P} \setminus \mathcal{H}) \cup \Gamma^\nu \cup \tilde{S}_{i-1}$. Note that, assuming $\neg C_\nu$, i.e., $|\Gamma^\nu| < \alpha \cdot |\mathcal{H}|$, we have

$$\begin{aligned} |F_i| &< (n - |\mathcal{H}|) + \alpha \cdot |\mathcal{H}| + 2r \\ &\leq n - |\mathcal{H}| + \alpha \cdot |\mathcal{H}| + 2\alpha \cdot |\mathcal{H}| \\ &= n - (1-3\alpha) \cdot |\mathcal{H}|. \end{aligned}$$

Further note that

$$\Pr\left[\tilde{X}_i = 0\right] \leq \Pr[N_i \subseteq F_i] + \Pr[\exists n^* \in N_i : N_i \setminus \{n^*\} \subseteq F_i].$$

We now bound these two probabilities individually.

For the first one, note that, for $\alpha < 1/3$, the probability that one uniformly chosen neighbor is in F_i is at most

$$\frac{n - (1-3\alpha) \cdot |\mathcal{H}|}{n} \leq 1 - (1-3\alpha) \cdot \gamma.$$

We are interested in the probability that d uniformly chosen neighbors without repetition are all in F_i. This is upper bounded by the probability to choose with repetition, since after choosing one neighbor in F_i without repetition, there is one element less in F_i available. Hence,

$$\Pr[N_i \subseteq F_i] \leq (1 - (1-3\alpha)\cdot\gamma)^d.$$

The second probability can be upper bounded by fixing one out of the d chosen neighbors (that could be $n^* \notin F_i$), and then bounding the probability that the remaining $d-1$ neighbors chosen out of $n-1$ remaining parties are all in F_i. Since there are d ways of fixing one of the neighbors, we can use the union bound to obtain

$$\Pr[\exists n^* \in N_i : N_i \setminus \{n^*\} \subseteq F_i] \leq d \cdot \left(\frac{n-(1-3\alpha)\cdot|\mathcal{H}|}{n-1}\right)^{d-1}.$$

Since we assume $n \geq 11 \cdot \gamma^{-1}$, we have $n-1 \geq n - \frac{n\cdot\gamma}{11} = n \cdot \frac{11-\gamma}{11}$. Hence, the probability above can be upper bounded by

$$d \cdot \left(\frac{11}{11-\gamma} \cdot \frac{n-(1-3\alpha)\cdot|\mathcal{H}|}{n}\right)^{d-1} \leq d \cdot \left(\frac{11}{11-\gamma} \cdot (1-(1-3\alpha)\cdot\gamma)\right)^{d-1}.$$

Note that

$$(1-(1-3\alpha)\cdot\gamma)^d \leq \left(\frac{11}{11-\gamma} \cdot (1-(1-3\alpha)\cdot\gamma)\right)^{d-1}.$$

We can therefore conclude that

$$\Pr\left[\tilde{X}_i = 0\right] \leq (d+1) \cdot \left(\frac{11}{11-\gamma} \cdot (1-(1-3\alpha)\cdot\gamma)\right)^{d-1}.$$

Let

$$p := 1-(d+1) \cdot \left(\frac{11}{11-\gamma} \cdot (1-(1-3\alpha)\cdot\gamma)\right)^{d-1} \leq \Pr\left[\tilde{X}_i = 1\right].$$

Then, for $\phi \leq 2p$, we have $\mathrm{E}[\sum_{i=1}^r \tilde{X}_i] \geq rp \geq r\phi/2$. Thus, Equation (10) and the Chernoff bound imply

$$\Pr[\neg B_\nu] = \Pr[|\theta^{\nu+1}| \leq (1-\delta)\cdot\phi\cdot|\theta^\nu|] \leq \Pr\left[\sum_{i=1}^r \tilde{X}_i \leq (1-\delta)\cdot\frac{r\phi}{2}\right] \leq e^{-\frac{\delta^2 r\phi}{4}}.$$

Furthermore, $\bigcap_{j<\nu} B_j$ ensures that

$$r = |\theta^\nu| \geq ((1-\delta)\cdot\phi)^\nu.$$

Therefore,

$$\Pr[\neg B_\nu] \leq e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^\nu}{4}}.$$

Further note that B_ν is true when all $\tilde{X}_i = 1$. Thus,

$$\Pr[\neg B_\nu] \leq \Pr\left[\bigcup_{i=1}^r \tilde{X}_i = 0\right] \leq r\cdot(1-p) \leq d^\nu \cdot (1-p),$$

and the claim follows. □

Equation (9) and the claim we have just proven imply for any $k \in \mathbb{N}$,

$$\begin{aligned}\Pr[D] &\leq \sum_{\nu=0}^{k-1} d^\nu \cdot (d+1) \cdot \left(\frac{11}{11-\gamma} \cdot (1-(1-3\alpha)\cdot\gamma)\right)^{d-1} + \sum_{\nu=k}^{\lambda-1} e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^\nu}{4}} \\ &= \frac{d^k-1}{d-1}(d+1) \cdot \left(\frac{11}{11-\gamma} \cdot (1-(1-3\alpha)\cdot\gamma)\right)^{d-1} + \sum_{\nu=k}^{\lambda-1} e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^\nu}{4}}.\end{aligned}$$

We further have

$$\sum_{\nu=k}^{\lambda-1} e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^\nu}{4}} = \sum_{\nu=0}^{\lambda-1-k} e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^{\nu+k}}{4}} = \sum_{\nu=0}^{\lambda-1-k} \left(e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^k}{4}}\right)^{((1-\delta)\cdot\phi)^\nu}.$$

Due to assumption (5), we have $(1-\delta)\cdot\phi \geq \frac{3}{2}$. Hence,

$$((1-\delta)\cdot\phi)^\nu \geq \left(\frac{3}{2}\right)^\nu \geq \nu + \frac{1}{5}.$$

Further note that $e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^k}{4}} < 1$. Therefore:

$$\sum_{\nu=0}^{\lambda-1-k} \left(e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^k}{4}}\right)^{((1-\delta)\cdot\phi)^\nu} \leq e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^k}{4}\cdot\frac{1}{5}} \cdot \sum_{\nu=0}^{\infty} \left(e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^k}{4}}\right)^\nu = \frac{e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^k}{20}}}{1-e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^k}{4}}}.$$

Hence, it follows that

$$1-\Pr[D] \geq 1 - \frac{d^k-1}{d-1}(d+1)\cdot\left(\frac{11}{11-\gamma}\cdot(1-(1-3\alpha)\cdot\gamma)\right)^{d-1} - \frac{e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^k}{20}}}{1-e^{-\frac{\delta^2\phi\cdot((1-\delta)\cdot\phi)^k}{4}}}. \tag{11}$$

The desired bound on the probability for late delivery now follows from Equations (6) to (8) and (11). □

Using Lemmas 2 and 3, we now prove our main theorem for FFlood. For completeness we restate it below.

Theorem 5. *There exists $\xi \in (0,1]$ such that for any $n \geq 50\cdot\gamma^{-1}$ there is a $d = O(\gamma^{-1})$ and $\Delta = O(\log(n)\cdot\Delta_{\mathrm{NET}})$ such that the protocol FFlood(d) is a weak (Δ,ξ)-flooding protocol. The security also holds against computationally unbounded adversaries.*

Proof. We let $\delta := \frac{1}{10}$, $\phi := \frac{5}{3}$, and $\alpha := \frac{1}{33}$. Then, $(1-\delta)\cdot\phi = \frac{3}{2}$. Hence, Equation (5) is fulfilled. Furthermore,

$$1-(1-3\alpha)\cdot\gamma = 1 - \frac{10}{11}\cdot\gamma = \frac{11-10\gamma}{11}.$$

Hence,

$$\frac{11}{11-\gamma}(1-(1-3\alpha)\cdot\gamma) = \frac{11-10\gamma}{11-\gamma} = \frac{11-\gamma-9\gamma}{11-\gamma} = 1 - \frac{9\gamma}{11-\gamma} < 1 - \frac{9\gamma}{11}.$$

Note that since $1-x \leq e^{-x}$, we have $\log(1-x) \leq -x$ for all $x \in (0,1)$, and consequently $\log(1-x)/x \leq -1$. Hence, $(1-x)^{1/x} \leq e^{-1} < \frac{1}{2}$. Setting $x := \frac{9\gamma}{11c}$ for some positive constant c yields

$$\left(\frac{11}{11-\gamma}(1-(1-3\alpha)\cdot\gamma)\right)^{\frac{11c}{9\gamma}} = \left(\frac{11-10\gamma}{11-\gamma}\right)^{\frac{11c}{9\gamma}} < \left(1-\frac{9\gamma}{11}\right)^{\frac{11c}{9\gamma}} < \frac{1}{2^c}. \tag{12}$$

This implies that for a sufficiently large constant c and $d = \frac{11c}{9\gamma} + 1 = O(\gamma^{-1})$, Equation (4) is also satisfied.

Therefore, Lemma 3 implies that for some $\lambda = O(\log(n))$, any sender s and receiver p (in particular the sender and receiver that minimize the probability for the event below), and for $G \xleftarrow{\$} \mathsf{HSP}(s,d,\lambda)$,

$$\Pr[p \in \Gamma_s^\lambda(G)] \geq \frac{\alpha \cdot |\mathcal{H}| - 1}{|\mathcal{H}|} \cdot \Bigg(1 - \frac{d^k - 1}{d - 1}(d+1) \cdot \left(\frac{11}{11-\gamma} \cdot (1 - (1 - 3\alpha) \cdot \gamma)\right)^{d-1} - \frac{e^{-\frac{\delta^2 \phi \cdot ((1-\delta)\cdot\phi)^k}{20}}}{1 - e^{-\frac{\delta^2 \phi \cdot ((1-\delta)\cdot\phi)^k}{4}}}\Bigg).$$

We note that $n \geq 50 \cdot \gamma^{-1} \implies h \geq 50$ and therefore

$$\frac{\alpha \cdot |\mathcal{H}| - 1}{|\mathcal{H}|} = \alpha - \frac{1}{|\mathcal{H}|} \geq \alpha - \frac{1}{h} \geq \frac{1}{33} - \frac{1}{50} \geq \frac{1}{100},$$

which is constant. Furthermore, for $k = 20$, we have $\delta^2\phi \cdot ((1-\delta)\cdot\phi)^k \geq 50$, and thus

$$\frac{e^{-\frac{\delta^2 \phi \cdot ((1-\delta)\cdot\phi)^k}{20}}}{1 - e^{-\frac{\delta^2 \phi \cdot ((1-\delta)\cdot\phi)^k}{4}}} \leq \frac{e^{-\frac{5}{2}}}{1 - e^{-\frac{25}{2}}} \leq \frac{1}{10}.$$

Finally, Equation (12) implies that for a sufficiently large constant c and $d = \frac{11c}{9\gamma} + 1 = O(\gamma^{-1})$, we have

$$\frac{d^k - 1}{d - 1}(d+1) \cdot \left(\frac{11}{11-\gamma} \cdot (1 - (1 - 3\alpha) \cdot \gamma)\right)^{d-1} \leq \frac{1}{10}.$$

Hence, for $\xi := \frac{1}{125}$ and $\Delta := \Delta_{\text{NET}} \cdot \lambda = O(\log(n) \cdot \Delta_{\text{NET}})$, Lemma 2 implies that $\mathsf{FFlood}(d)$ is a weak (Δ, ξ)-flooding protocol. $\square$

5.3 Flooding Amplification

We present a compiler that amplifies delivery guarantees of a weak flooding protocol to full-fledged flooding. The protocol WeakFlood2Flood is parameterized by a *weak* flooding protocol, an erasure correcting code scheme (ECCS), and a cryptographic accumulator. The idea of the protocol is that when a sender wishes to send a message, they divide the message into multiple shares using the ECCS. The sender will then send each of these shares using the weak flooding protocol. Each receiver will receive a set of shares and try to reconstruct the original message from this. Intuitively, if everybody receives sufficiently many of the original shares within the given time, then the only thing that can prevent an honest party from reconstructing the message sent by the sender is if an adversary managed to inject some “false shares” into the set of shares an honest party tries to reconstruct their message from. To prevent this from happening, the sender will create an accumulated value of all shares, and then instead of sending out only the share, they will send out the share, an accumulated value, and a proof that this share belongs to this accumulated value. On the receiving end, honest parties will group shares by the accumulated value they belong to and only try to reconstruct from shares that belong to the same accumulated value. Hence, an adversary will have to break the collision-free property of the accumulator scheme in order to inject such false shares.

It is only left to ensure that all parties receive sufficiently many of the original shares. We will ensure this by instantiating WeakFlood2Flood with a weak flooding protocol FFlood, such that each party is guaranteed to receive a constant fraction of the shares if a message is split into sufficiently many shares and set the parameters of the ECCS accordingly. For the security proof of our amplification, we need the weak flooding protocol to be secure against computationally unbounded adversaries, which is the case for the protocol we presented above, see Theorem 5.

Protocol $\mathsf{WeakFlood2Flood}(\Pi, \zeta, \alpha)$

The protocol is parameterized by a weak flooding protocol Π, a (μ, ϱ)-ECCS ζ for some $\mu, \varrho \in \mathbb{N}$, and a cryptographic accumulator α.
Each party $p_i \in \mathcal{P}$ keeps track of a set of shares received for a particular accumulator z, $\texttt{ReceivedShares}_i[z]$. Additionally, each party p_i keeps track of a set of received messages $\texttt{Received}_i$.

Initialize: Initially, each party p_i sets $\texttt{ReceivedShares}_i \coloneqq \varnothing$, and $\texttt{Received}_i \coloneqq \varnothing$. Furthermore, μ independent instances $\Pi_1, \ldots, \Pi_\mu$ of the weak flooding protocol are initialized.

Send: When p_i receives (Send, m) they 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 $(z, \pi_1, \ldots, \pi_\mu) = \alpha.\mathsf{Accumulate}(\{(s_i, i) \mid 1 \leq i \leq \mu\})$. Now, the party inputs the message (s_j, π_j, z) to Π_j for $1 \leq j \leq \mu$. Finally, they add m to $\texttt{Received}_i$.

Get Messages: When p_i receives $(\mathit{GetMessages})$ they return $\texttt{Received}_i$.

When party p_i receives a tuple (s, π, z) in Π_j where $\alpha.\mathsf{Verify}((s, j), \pi, z) = \top$, they add (s, j) to $\texttt{ReceivedShares}_i[z]$. Furthermore, p_i checks if $|\texttt{ReceivedShares}_i[z]| \geq \mu - \varrho$. If that is the case, p_i ignores further messages with this accumulated value z and does the following:

1. Obtain a sequence of shares $s_1, \ldots, s_\mu$ by letting $s_j = s$ if $(s, j) \in \texttt{ReceivedShares}_i[z]$ and otherwise sets $s_j = \perp$ if no such pair is in $\texttt{ReceivedShares}_i[z]$.
2. Decode the shares and add the recovered message to the set of received messages, $\texttt{Received}_i \coloneqq \texttt{Received}_i \cup \{\zeta.\mathsf{Dec}(s_1, \ldots, s_\mu)\}$.

Security of WeakFlood2Flood. We now state and prove the security of WeakFlood2Flood.

Theorem 6. *Let $\xi \in (0, 1]$, let $\Delta \in \mathbb{N}$, and let Π be a weak (Δ, ξ)-flooding protocol with security against computationally unbounded adversaries. Further, let $\delta \in (0, 1]$, let $\mu \in \mathbb{N}$, let $\varrho \geq \mu \cdot (1 - (1 - \delta) \cdot \xi)$, let ζ be a (μ, ϱ)-ECCS, and let α be a WSCAS. The probability in an execution with a PPT adversary $\mathcal{A}$ that a message sent using the protocol WeakFlood2Flood(Π, ζ, α) is not delivered within time Δ to all honest parties is less than*

$$|\mathcal{H}| \cdot e^{-\frac{\delta^2 \cdot \xi \cdot \mu}{2}} + \mathrm{negl}(\kappa).$$

Proof. Consider an execution with a PPT adversary $\mathcal{A}$ in which a message is sent by an honest sender s at time t by flooding shares $s_1, \ldots, s_\mu$ together with an accumulated value z and the corresponding proofs π_j. Further consider an adversary $\mathcal{A}'$ on the collision freeness of the accumulator that emulates the full protocol execution with adversary $\mathcal{A}$. Whenever a party in the emulation receives a tuple (s_j, π_j, z), the adversary $\mathcal{A}'$ checks whether π_j is valid, z matches the accumulator value generated by the honest sender, and s_j is not one of the honestly generated shares. In that case, $\mathcal{A}'$ has found a collision, which we assume is only possible with negligible probability.

On the other hand, if an honest party p receives at least $(1 - \delta) \cdot \xi \cdot \mu \geq \mu - \varrho$ of the original shares with valid proofs and no different shares for the same accumulator value z, then by properties of the (μ, ϱ)-ECCS, p is able to reconstruct the original message. It is therefore sufficient to bound the probability that there exists an honest party that does not receive at least $(1-\delta)\cdot\xi\cdot\mu$ shares from the sender within time Δ.

For each honest party $p_i \in \mathcal{H}$ we introduce random indicator variables $S_{i,1}, S_{i,2}, \ldots, S_{i,\mu}$ where $S_{i,j}$ indicates whether or not party p_i received share s_j by time $t+\Delta$. Further, we introduce a variable that denotes how many shares party p_i receives from honest parties $S_i = \sum_{j=1}^{\mu} S_{i,j}$ at latest at time $t+\Delta$. Note that the $S_{i,j}$ are not necessarily independent, since the adversary can, e.g., decide to on purpose increase the delivery probability for one of the shares if another share is delivered. It is, however, not possible for the adversary to decrease the delivery probability for some share by correlating the executions for different shares since all honest parties use independent randomness for the different instances of Π. This is formalized in the following claim that is a generalization of [MPR07, Lemma 4] and follows the same proof idea.

Claim 2. *For an adversary $\mathcal{A}_{i,j}$ interacting with the jth instance of the weak flooding protocol Π_j, we denote by $S_{i,j}^{\mathcal{A}_{i,j}}$ the event that the honest party p_i receives the message sent in Π_j by time $t+\Delta$. As above, $S_{i,j}$ denotes the corresponding event in an interaction of a single adversary $\mathcal{A}$ with the overall protocol $\mathsf{WeakFlood2Flood}(\Pi, \zeta, \alpha)$. Further let $S_i = \sum_{j=1}^{\mu} S_{i,j}$ and $S_i^* \coloneqq \sum_{j=1}^{\mu} S_{i,j}^{\mathcal{A}_{i,j}}$. Then, for every i, there exist independent, computationally unbounded adversaries $\mathcal{A}_{i,1}, \ldots, \mathcal{A}_{i,\mu}$ such that S_i stochastically dominates S_i^*. That is, we have for any $k \in \mathbb{N}$,*

$$\Pr[S_i \leq k] \leq \Pr[S_i^* \leq k].$$

Proof. We fix some arbitrary i and first consider the adversaries $\mathcal{A}'_{i,0}, \mathcal{A}'_{i,1}, \ldots, \mathcal{A}'_{i,\mu}$ that can exchange messages between each other and operate as follows: The adversary $\mathcal{A}'_{i,0}$ emulates $\mathcal{A}$, but does not directly interact with any protocol. Instead, when $\mathcal{A}$ wants to interact with protocol instance Π_j, $\mathcal{A}'_{i,0}$ sends a message to $\mathcal{A}'_{i,j}$ with the instruction to interact accordingly. The result of this interaction is then passed back from $\mathcal{A}'_{i,j}$ to $\mathcal{A}'_{i,0}$. To allow all adversaries to keep track of the total number of exchanged messages, $\mathcal{A}'_{i,0}$ always sends a dummy message to all other $\mathcal{A}'_{i,j}$, so that all $\mathcal{A}'_{i,j}$ receive the same number of messages from $\mathcal{A}'_{i,0}$. To further remove any variance on timing and the total number of messages sent between the adversaries, we let $\mathcal{A}'_{i,0}$ send (dummy) messages at every activation, continuing until some upper bound on the total number of activations needed to complete the whole protocol execution. Note that the overall behavior is identical to an execution of $\mathsf{WeakFlood2Flood}(\Pi, \zeta, \alpha)$ with $\mathcal{A}$, and therefore, the probabilities of all events in the two experiments are identical.

Assume $\mathcal{A}'_{i,0}$ sends ℓ messages to all parties. We now modify all adversaries as follows: $\mathcal{A}'_{i,0}$ does not send the ℓth message to any adversary, and $\mathcal{A}'_{i,j>0}$ does not expect the last message, but instead considers the set of all possibly received messages, and behaves as if it received a message that minimizes the probability that p_i receives the message in Π_j by time $t+\Delta$ (note that this step requires $\mathcal{A}'_{i,j}$ to be computationally unbounded). By doing so, we have reduced the number of messages exchanged with each $\mathcal{A}'_{i,j}$ such that for every j, the probability that p_i receives the message in Π_j is not increased. This ensures that for all k,

$$\Pr[S_i \leq k] \leq \Pr\left[\sum_{j=1}^{\mu} S_{i,j}^{\mathcal{A}'_{i,j}} \leq k\right]$$

We now inductively continue, until we arrive at adversaries $\mathcal{A}_{i,0}, \mathcal{A}_{i,1}, \ldots, \mathcal{A}_{i,\mu}$ that do not exchange any messages. They are therefore independent and $\mathcal{A}_{i,0}$ is not needed anymore. □

By the above claim, it is sufficient to bound the probability for the event that $S_i^* \leq (1-\delta)\cdot\xi\cdot\mu$ in order to bound the probability that party p_i receives less than $(1-\delta)\cdot\xi\cdot\mu$ shares by time

$t+\Delta$. Because each Π_j is a weak (ξ,Δ)-flooding protocol, we have $\Pr[S^*_{i,j}=1]\geq\xi$ for all i,j. By linearity of expectation, this implies

$$\mathrm{E}[S^*_i]\geq\xi\cdot\mu.$$

Note that the $S^*_{i,j}$ for $j=1,\ldots,\mu$ are independent because they are events in independent protocol instances interacting with independent adversaries. We can therefore apply the Chernoff bound to obtain that for any $\delta\in[0,1)$,

$$\Pr\left[S^*_i\leq(1-\delta)\cdot\xi\cdot\mu\right]\leq e^{-\frac{\delta^2\cdot\xi\cdot\mu}{2}}.$$

Finally, the probability that there *exists any honest party* that does not receive at least $(1-\delta)\cdot\xi\cdot\mu$ shares is

$$\begin{aligned}&\Pr\left[\exists p_i\in\mathcal{H},\ \text{s.t. } p_i \text{ receives less than } (1-\delta)\cdot\xi\cdot\mu \text{ shares}\right]\\ &\leq\sum_{p_i\in\mathcal{H}}\Pr\left[S_i\leq(1-\delta)\cdot\xi\cdot\mu\right]\leq|\mathcal{H}|\cdot e^{-\frac{\delta^2\cdot\xi\cdot\mu}{2}},\end{aligned}$$

by the union bound. □

It is noteworthy that WeakFlood2Flood inherits the delivery guarantee of the weak flooding protocol that it is instantiated with. Next, we state a direct corollary of the above theorem, stating that for appropriate parameters, WeakFlood2Flood is a strong flooding protocol.

Corollary 1. *Let $\xi\in(0,1]$, let $\Delta\in\mathbb{N}$, and let Π be a weak (Δ,ξ)-flooding protocol. Further, let $\mu\geq8\cdot\frac{\log(n)+\kappa}{\xi}$, let $\varrho\geq\mu\cdot\left(1-\frac{\xi}{2}\right)$, let ζ be a (μ,ϱ)-ECCS, and let α be a WSCAS. The protocol WeakFlood2Flood(Π,ζ,α) is a strong Δ-flooding protocol.*

Proof. We note that $|\mathcal{H}|\leq n$ and use Theorem 6 instantiated with $\delta:=\frac{1}{2}$ to obtain that the probability that some honest party does not obtain a message sent by an honest party within time Δ is at most

$$|\mathcal{H}|\cdot e^{-\frac{\delta^2\cdot\xi\cdot\mu}{2}}+\mathrm{negl}(\kappa)\leq n\cdot e^{-\log(n)-\kappa}+\mathrm{negl}(\kappa)=e^{-\kappa}+\mathrm{negl}(\kappa)\leq\mathrm{negl}(\kappa).$$ □

5.4 Communication Complexity of the Combined Protocol

We consider the combined protocol

$$\mathsf{ECFlood}(d,\zeta,\alpha):=\mathsf{WeakFlood2Flood}(\mathsf{FFlood}(d),\zeta,\alpha),$$

instantiated with a (μ,ϱ)-ECCS ζ, and a WSCAS scheme α. Note that Corollary 1 and Theorem 5 imply that for $\mu\geq8\cdot\frac{\log(n)+\kappa}{\xi}$, $\varrho\geq\mu\cdot\left(1-\frac{\xi}{2}\right)$, $n\geq50\cdot\gamma^{-1}$, $d=O(\gamma^{-1})$, and $\Delta=O(\log(n)\cdot\Delta_{\mathrm{NET}})$, ECFlood$(d,\zeta,\alpha)$ is a strong Δ-flooding protocol.

To analyze the (per-party) communication complexity, consider the case where a single message of length l is input to an honest party. First note that the honest sender produces μ shares and sends these together with the sequence number of the share, an accumulator proof, and an accumulated value using FFlood(d). Hence, the size of each of these messages is bounded by $\zeta.\mathtt{ShareSize}(l)+\log(\mu)+\alpha.\mathtt{ProofSize}(\mu)+\alpha.\mathtt{AccSize}$. The protocol FFlood$(d)$ uses a neighborhood of size d for every message and every honest party sends each message at most once to their neighbors. Furthermore, no other messages related to this message is sent by any honest party, unless the adversary breaks the collision freeness of the accumulator and manages to inject additional shares, which is only possible with negligible probability. Hence, each honest party sends at most $\mu \cdot d$ messages (except with negligible probability) and the per-party communication complexity is upper bounded by

$$\mu \cdot d \cdot (\zeta.\texttt{ShareSize}(l) + \log(\mu) + \alpha.\texttt{ProofSize}(\mu) + \alpha.\texttt{AccSize}). \tag{13}$$

Using Reed-Solomon codes and since we can set $\varrho := \lceil \mu \cdot (1 - \frac{\xi}{2}) \rceil$, Equation (1) implies that the share size can be bounded by

$$\zeta.\texttt{ShareSize} = O\Big(\frac{l}{\mu - \varrho}\Big) = O\Big(\frac{l}{\mu \cdot \xi}\Big) = O\Big(\frac{l}{\mu}\Big).$$

Furthermore, using efficient accumulators (see Equation (2)) that have accumulator and proof sizes of $O(\kappa)$ bits, setting $\mu = O(\frac{\log(n)+\kappa}{\xi})$, $d = O(\gamma^{-1})$, and using that ξ is just a constant, we obtain from Equation (13) that the per-party communication complexity is bounded by

$$\begin{aligned} &O\big((\log(n) + \kappa) \cdot \gamma^{-1} \cdot (l \cdot (\log(n) + \kappa)^{-1} + \log(\log(n) + \kappa) + \kappa)\big) \\ &\qquad = O\big(\gamma^{-1} \cdot (l + (\log(n) + \kappa) \cdot (\log(\log(n)) + \kappa))\big), \end{aligned} \tag{14}$$

and the total communication complexity is at most n times that.

Note that for

$$l = \Omega\big((\log(n) + \kappa) \cdot (\log(\log(n)) + \kappa)\big),$$

this simplifies to

$$O(l \cdot \gamma^{-1}),$$

which is optimal by Theorem 3.

6 Flooding in the Weighted Setting

Model. We consider the setting where parties are assigned a fraction of the total weight, and assume that the assigned weights are public. We let W_p denote the weight assigned to party p, and let $\alpha_p := \frac{W_p}{\sum_{p \in \mathcal{P}} W_p}$ i.e., the fraction of the total weight assigned to party p. The adversary can corrupt any subset of the parties such that the remaining set of honest parties together constitutes more than a $\widetilde{\gamma} \in (0, 1]$ fraction of the total weight. That is, $\sum_{p \in \mathcal{H}} \alpha_p \geq \widetilde{\gamma}$, and all parties have a non-zero positive weight i.e. $\forall p \in \mathcal{P}, W_p > 0$.

Transformation. We provide a general transformation for a flooding protocol in the equal-weights setting to the weighted setting, leveraging ideas from [LMM+22]. The main idea of our transformation is to let each party *emulate* a number of parties in another flooding protocol. We use the same emulation function as [LMM+22] where each weighted party $p \in \mathcal{P}$ emulates $\lceil \alpha_p \cdot n \rceil$ non-weighted parties. For each party $p \in \mathcal{P}$, we define a set of parties that this party emulates as $\texttt{E}(p) := \{p_i \mid i \in \mathbb{N} \wedge i \leq \lceil \alpha_p \cdot n \rceil\}$. Note that because all parties have a non-zero weight, all parties emulate at least one party, i.e., for any party $p \in \mathcal{P}$ we have $\texttt{E}(p) \neq \varnothing$. For convenience, we introduce notation for the set of emulated parties, $\mathcal{P}_{\texttt{E}} = \bigcup_{p \in \mathcal{P}} \texttt{E}(p)$, the total number of emulated parties $n_{\texttt{E}} = |\mathcal{P}_{\texttt{E}}|$, the set of emulated parties that are emulated by honest players $\mathcal{H}_{\texttt{E}} = \bigcup_{p \in \mathcal{H}} \texttt{E}(p)$ and the number of honestly emulated parties $h_{\texttt{E}} = |\mathcal{H}_{\texttt{E}}|$. Following [LMM+22], we note that

$$n_{\texttt{E}} = \sum_{p \in \mathcal{P}} \lceil \alpha_p \cdot n \rceil \leq \sum_{p \in \mathcal{P}} \alpha_p \cdot n + 1 = 2 \cdot n, \tag{15}$$

and

$$h_{\mathtt{E}} = \sum_{p \in \mathcal{H}} \lceil \alpha_p \cdot n \rceil \geq \widetilde{\gamma} \cdot n. \tag{16}$$

When defining a strong flooding protocol in Section 2.3, we were not explicit about the set of parties a flooding protocol has to provide guarantees for, as all of our previous flooding protocols have simply worked for the same set of assumed parties $\mathcal{P}$. Below, this will not be the case, as the flooding protocols we discuss will work for a different set of parties. Hence, we will make these sets explicit by using the phrase that "a protocol is a flooding protocol for a set of parties".

Protocol Flood2WeightedFlood(Π)

The protocol is parameterized by a protocol Π that is a flooding protocol for $\mathcal{P}_{\mathtt{E}}$.
Each party $p \in \mathcal{P}$ starts a process for each of their emulated parties, and lets these processes participate in the protocol Π.

Initialize: Initially, each party p initialize all of their emulated parties $\mathtt{E}(p)$ in Π.

Send: When p receives (Send, m) they pick $p_i \in \mathtt{E}(p)$ and forward (Send, m) to p_i in Π.

Get Messages: When p receives $(\mathit{GetMessages})$ they pick $p_i \in \mathtt{E}(p)$ forward $(\mathit{GetMessages})$ to p_i in Π, and return the set of messages returned to p_i.

Below we prove that if Flood2WeightedFlood is instantiated with a strong flooding protocol for $n_{\mathtt{E}}$, then Flood2WeightedFlood will be a strong flooding protocol.

Theorem 7. *Let $\Delta \in \mathbb{N}$. If Π is a strong Δ-flooding protocol for $\mathcal{P}_{\mathtt{E}}$ under the assumption that at least $\widetilde{\gamma} \cdot n$ of them behaves honestly, then Flood2WeightedFlood(Π) is a strong Δ-flooding protocol for $\mathcal{P}$.*

Proof. Let m be a message that is input to some honest party at time t. Since all parties emulate at least one party, this implies that the message will also be input to some honest emulated party in Π at time t. Because Π is a strong Δ-flooding protocol for $\mathcal{P}_{\mathtt{E}}$ when $\widetilde{\gamma} \cdot n$ parties are honest (Equation (16) ensures that this is actually the case), then there is an overwhelming probability in κ that all emulated parties receive m before $t + \Delta$. As each honest party emulates at least one party, this implies that all honest parties will also receive the message with a probability that is overwhelming in Π. □

Realising a strong flooding protocol for $\mathcal{P}_{\mathtt{E}}$. It may seem like Theorem 7 allows us to easily translate the protocols presented in Section 5 to the weighted setting. However, even though the protocols in these sections work for any set of parties, they make channels, which are only assumed for the actual set of parties $\mathcal{P}$ and not the emulated set of parties $\mathcal{P}_{\mathtt{E}}$. To use these protocols blackbox, we need to show how to establish channels between the emulated parties.

We note that channels for the emulated set of parties can easily be established from channels between the original set of parties. One way to do this is by simply prepending $(p_e, p_{e'})$ to any message that an emulated party p_e wishes to send to another emulated party p'. When a party receives such a message on a normal channel, they will take it as an input on the emulated channel between the emulated parties p_e and $p_{e'}$.

Communication complexity analysis. The analysis in Section 5.4 also applies when the protocol is transformed to work for the weighted setting because $n_{\mathtt{E}} = O(n)$ (Equation (15)) and the fraction of honest emulated parties $\frac{h_{\mathtt{E}}}{n_{\mathtt{E}}} = O(\tilde{\gamma})$ (by Equations (15) and (16)). The only thing that changes is that all messages will have identifiers for emulated parties prepended. The size of such identifiers is bounded by $O(\log(n))$. When this is threaded through the analysis using the same parameters as in Section 5.4, we see that for a suitable d, ζ and α, the communication complexity of the $\mathsf{Flood2WeightedFlood}(\mathsf{ECFlood}(d, \zeta, \alpha))$ is bounded by $O(\gamma^{-1} \cdot n \cdot (l + (\log(n) + \kappa)^2))$, which is optimal when $l = \Omega((\log(n) + \kappa)^2)$. It is, however, worth noting that for our particular protocol, it is not necessary to keep the messages delivered to different emulated parties separate. In particular, $\mathsf{Flood2WeightedFlood}(\mathsf{ECFlood}(d, \zeta, \alpha))$ would have the same guarantees if any message sent from an emulated party of party p_i to an emulated party of p_j is simply delivered to *all* emulated parties of p_j. In that case, the communication complexity of $\mathsf{Flood2WeightedFlood}(\mathsf{ECFlood}(d, \zeta, \alpha))$ would be optimal under the same constraints as $\mathsf{ECFlood}(d, \zeta, \alpha)$.

7 Security in the UC Model

The Universal Composable (UC) model by Cannetti [Can20] is by many considered the golden standard for security for cryptographic protocols, because security in this model ensures that the protocol remains secure independently of the context it is deployed in. In this section, we formalize a theorem that informally says that any protocol that is a flooding protocol w.r.t. the property-based definition (Definition 3) is a UC secure implementation of a flooding network. The theorem is basically a generalization of the proof ideas that appear in [MNT22] when proving that their flooding protocol implements the ideal functionality.

Before stating and proving the actual theorem, we first summarize the necessary details of the UC framework to understand the remainder of the section and present an ideal functionality for flooding.

7.1 The UC Model

UC [Can20] is a framework that provides precise execution semantics for distributed protocols using Turing machines.

Security. The intended behavior of a protocol in the UC framework is defined by comparing the execution of distributed protocol to an execution of an ideal functionality. The ideal functionality can informally be thought of as a trusted third party, which behavior defines the intended behavior of the distributed protocol. Security is defined by comparing the two executions, and if for all attacks there exists a program that translates an attack on the protocol to an attack on the ideal functionality, which renders the attack to not be an attack by definition. Below, we recap the formal definition of security in the UC model.

Definition 6 (Secure Implementation)**.** Let Π be a protocol, $\mathcal{F}$ an ideal functionality, and $\approx$ mean that the statistical distance between two distributions is negligible in the security parameter κ. The protocol Π is said to securely implement $\mathcal{F}$ if

$$\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z}, \mathtt{EXEC}(\mathcal{Z}, \mathcal{A}, \Pi) \approx \mathtt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{F}) \qquad (17)$$

where $\mathtt{EXEC}$ denotes the random variable defined as the binary output that the environment $\mathcal{Z}$ outputs after having executed protocol Π with adversary $\mathcal{A}$.

For further details on the UC model, we refer to [Can20].

Time. There is no build in notion of time in the UC model, which is needed when describing the ideal functionality for a flooding network. We therefore use the notion of time from TARDIS [BDD+21]. That is, to let a use a global ticker to ensure that all parties are activated in each time step, without imposing specific assumptions about the exact time being available to parties.

For further details about how this notion of time can be used in connection with flooding networks, we refer to [MNT22].

7.2 Flooding as a UC Functionality

We base our flooding functionality on the flooding functionality from [MNT22] but do not consider pre-corrupted parties (a corruption type specific to the model of delayed adversaries used in their work) nor relay messages as this is not needed for most blockchains protocols (see Section 2.3).

Functionality $\mathcal{F}_{\mathsf{Flood}}^{\Delta}$

The functionality is parameterized by a set of parties $\mathcal{P}$ and a delivery guarantee Δ.

It keeps track of a set of messages for each party `Mailbox`. These sets contain the messages that each party will receive after fetching. Additionally, it keeps track of the set of parties that has been corrupted by the adversary `Corrupted`.

Initialize: Initially, $\texttt{Corrupted} := \varnothing$ and $\texttt{Mailbox}[p_i] := \varnothing$ for all $p_i \in \mathcal{P}$.

Send: After receiving (Send, m) from p_i it leaks $(\mathit{LeakSend}, p_i, m)$ to the adversary.

Get Messages: After receiving $(\mathit{GetMessages})$ from p_i it outputs $\texttt{Mailbox}[p_i]$ to party p_i and $(\mathit{LeakGet}, p_i, \texttt{Mailbox}[p_i])$ to the adversary.

Set Message: After receiving $(\mathit{SetMessage}, m, p_i)$ from the adversary, the functionality sets $\texttt{Mailbox}[p_i] := \texttt{Mailbox}[p_i] \cup \{m\}$.

At any time after all parties have been initialized the functionality automatically enforces the following: For any message m that is input to an honest party p_i at some time t, it is ensured for any honest party $\forall p_j \in \mathcal{P} \setminus \texttt{Corrupted}$ that by time $t + \Delta$ they have received the message i.e., $m \in \texttt{Mailbox}[p_j]$.
The property are ensured by the functionality automatically[a] making the minimal possible additional calls with $\mathit{SetMessage}$.

[a]The global clock used to check the time ensures that the functionality is invoked at least once per time step, and therefore such automatic checks are possible.

7.3 Strong Flooding Implies UC Flooding

We are now ready to state the main result of this section. The theorem informally says that any flooding protocol that for UC executions is a strong flooding protocol (w.r.t. Definition 3) also securely implements $\mathcal{F}_{\mathsf{Flood}}$.

Theorem 8. *Let $\Delta \in \mathbb{N}$ and let Π be a protocol. If Π is a Δ-flooding protocol, then Π securely implements $\mathcal{F}_{\mathsf{Flood}}^{\Delta}$.*

To prove this we exploit that the functionality has no secrecy and therefore the full protocol can be simulated in black-box manner by simply forwarding the inputs that are leaked to the simulator from the functionality.

Proof. We are to prove Equation (17). To do so, we let $\mathcal{A}$ be an adversary and show the existence of a simulator $\mathcal{S}$ by constructing it explicitly. We define the simulator $\mathcal{S}$ similarly to the simulator in [MNT22, proof of Lemma 5]. That is, we let $\mathcal{S}$ emulate an execution of Π inside itself. In more detail, for any party $p \in \mathcal{P}$ it spawns a process that act as this party according to protocol Π, spawns a process that acts as the adversary $\mathcal{A}$, and spawns an environment $\mathcal{Z}'$. The simulator $\mathcal{S}$ controls the environment and has the following behavior:

- Whenever the simulator receives $(\mathit{LeakSend}, p, m)$ from the ideal functionality $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$ the simulator inputs (Send, m) via the environment $\mathcal{Z}'$ to the process for p.
- Whenever the simulator receives $(\mathit{LeakGet}, p, M)$ from the ideal functionality $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$ the simulator inputs $(\mathit{GetMessages})$ via the environment $\mathcal{Z}'$ to the process for p.
- Whenever a party p receives a message m in the emulated execution of Π the simulator inputs $(\mathit{SetMessage}, m, p)$ to the ideal functionality $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$.
- Whenever the emulated adversary within the simulator output something to the emulated environment, the simulator forwards this to the real environment outside the simulator.

Having defined the simulator, it is left to argue that for any environment $\mathcal{Z}$ we have

$$\mathtt{EXEC}(\mathcal{Z}, \mathcal{A}, \Pi) \approx \mathtt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{F}).$$

Observe that using the same randomness, the execution of Π in the real world and inside the head of the simulator will provide the same outputs as the same $\mathit{SetMessage}$ and $\mathit{GetMessages}$ inputs are given. Therefore, the only time that the outputs of to the environment will be different is when the ideal functionality automatically inserts messages to the mailbox of some party, because the simulator failed to do so timely. However, this only happens when the emulated protocol within the simulator fails to deliver messages to the emulated messages timely. Because the protocol Π is a Δ-flooding protocol, it in particular also ensures delivery within Δ for $\mathcal{A}$ and $\mathcal{Z}'$. Hence, it is ensured that the probability that this happens is negligible in κ. □

A direct corollary of Theorem 8 is that ECFlood securely implements $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$.

8 Practicality of ECFlood

In Section 5.4, it was shown that parameters could be set such that ECFlood theoretically constitutes an asymptotically optimal flooding. We instantiated many variables to constants required by our analysis, but these are most likely not instantiated optimally, and nor are our analyses themselves optimal.

To provide a guideline for how to instantiate the parameters of our protocol for practical performance and to compare its practical efficiency to the efficiency of state-of-the-art, we made probabilistic simulations of the protocol executions that explore how the message complexity of our protocols is affected by adjusting parameters, which we report on in the following sections. Finally, in Section 8.5, we report on initial experiments that suggests that the computational overhead associated with the WSCAS and ECCS are by no means hindrance for practical adoption of the protocol.

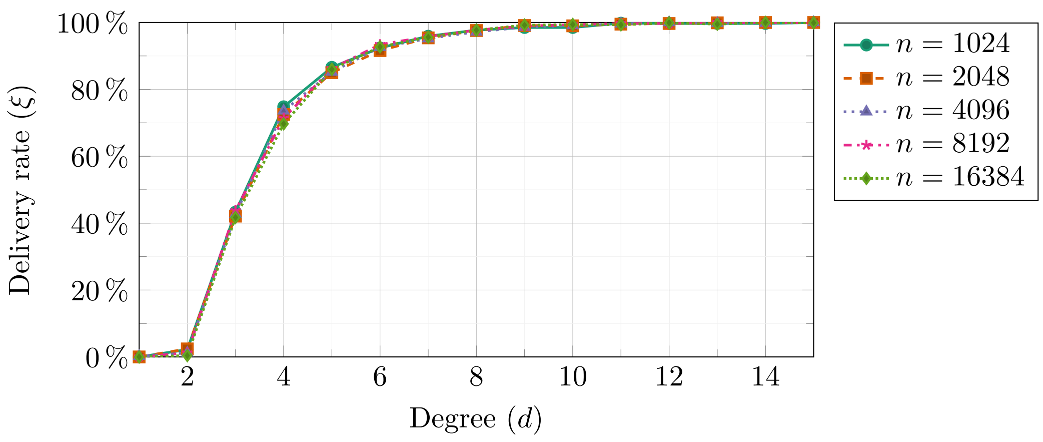

Figure 2: Results for simulations of FFlood(d) for different values of d and n. The delivery rate for a fixed party different from the sender is plotted as a function of changing the expected degree d.

8.1 Setup for Simulations

We implemented a probabilistic experiment among n nodes where $\frac{n}{2}$ of them behave honestly and will forward any received message according to our protocol if they receive a message. We consider not forwarding any messages the worst-case adversarial behavior. So the remaining $\frac{n}{2}$ parties, which we consider corrupt, will not participate in forwarding any messages sent to them. That is, an initial party will distribute several messages by selecting a set of d uniformly random neighbors (for varying values of d) for each message. These will then again forward each message to a random set of neighbors. This continues until no honest party receives the message for the first time. For simulations of FFlood, a single message is initially input, whereas, for ECFlood, several messages will be input corresponding to the number of shares a message is divided into in the protocol.

We have repeated our simulations 1000 times for each set of parameters except for Figure 1 where more simulations are done in order to make more accurate estimates of the error probabilities.

Below, we report on various statistics from our simulations.

8.2 Estimating Parameters for FFlood

In Section 5.4 we showed that the communication complexity of WeakFlood2Flood instantiated with FFlood(d) is directly proportional to $\frac{d}{\xi}$, if FFlood is a weak (Δ, ξ) flooding protocol for some Δ and ξ.

Initially, let us ignore how Δ is affected by changing the degree d and let us focus on finding estimates of $\frac{d}{\xi}$ for different degrees. A simple way to estimate the maximum ξ for which FFlood(d) is a ξ-weak flooding algorithm is to select a party different from the sender and count the rate with which this party receives a message throughout many executions. The results of this approach for different values of n and d are in Figure 2.

Note that the graphs for different values of n are extremely close to each other. Hence, our simulations confirm that the delivery rate of FFlood(d) (and thereby how well it acts as a weak flooding algorithm) is truly independent of the number of parties n. In particular, this holds even for very small values of d (and n).

It is striking that even for $d = 3$, where an honest party in expectation forwards the message to just 1.5 other honest parties, the probability that a particular party receives the message is about ∼45%. To understand this behavior, we collected additional statistics on the fraction of parties reached in each execution of the protocol. This data is presented in Figure 3.

Interestingly, it seems that for all the different degrees plotted; there is an initial drop before the curves reach a plateau, where they stay for a while. Our interpretation of this phenomenon is that there is quite a high probability that the initial sender or their close neighbors talk only to dishonest parties, which stops the propagation. However, once a critical mass of parties has been reached, it becomes very unlikely that they all select only dishonest parties or parties that have already received the message as their neighbors (which is what is required to make the propagation of the message stop). So even for $d = 3$, where on average each party only talks to 1.5 honest parties, there is more than 50% chance that the message will spread to more than 75% of the parties.

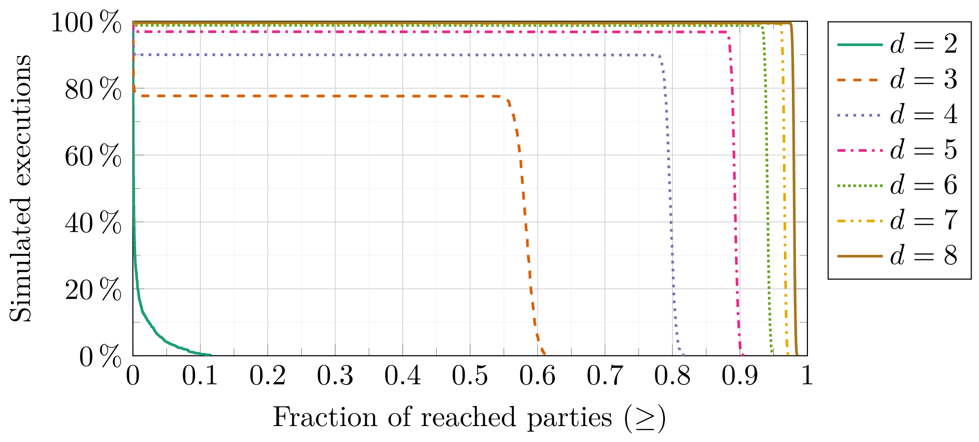

Figure 3: Results for simulations of $\mathsf{FFlood}(d)$ for different values of d and a fixed $n = 8192$. The graphs show the percentages of the simulated executions (on the y-axis) where at least a certain fraction of parties received the message (on the x-axis).

With estimates on ξ, we are also able to estimate the size of the factor $\frac{d}{\xi}$ that is multiplied by the message length in the communication complexity. In Figure 4, we plot this. Our simulations indicates that the best value for d in order to get the lowest overall communication complexity is ∼4 where $\frac{d}{\xi}$ will be just above 5.

We now turn our attention to how the Δ parameter of the weak flooding algorithm $\mathsf{FFlood}(d)$ is affected by varying the value d. To provide an upper bound on Δ, we recorded the maximum number of hops from the sender to any other party that receives the message in each execution. Note that the number of hops times the maximum delay on the point-to-point channels directly translates to an upper bound on the delivery time for a message in a real execution of the protocol. In Figure 5, we plot the maximum number of hops across all the simulations (for a single set of parameters) for the various number of parties n and degrees d. We include only those degrees that are of interest w.r.t. an overall low communication complexity.

We note that during the execution, no message was delivered in more than 32 hops for expected degrees at least 4. For degrees at least 5, this maximum number of observed hops drops to 23, and for degrees larger than 10, no message is delivered in more than 10 hops.

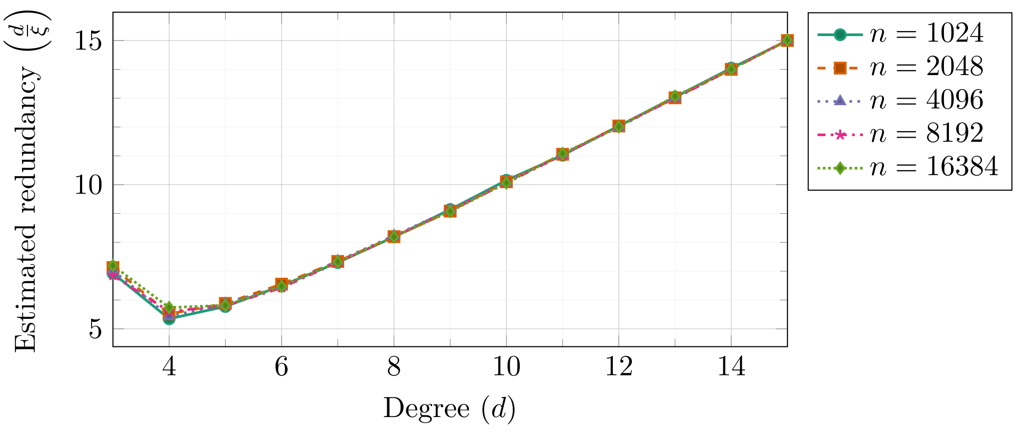

Figure 4: Results for simulations of $\mathsf{FFlood}(d)$ for different values of d and n. The graphs show the degree normalized by the observed delivery rate as a function of the degree.

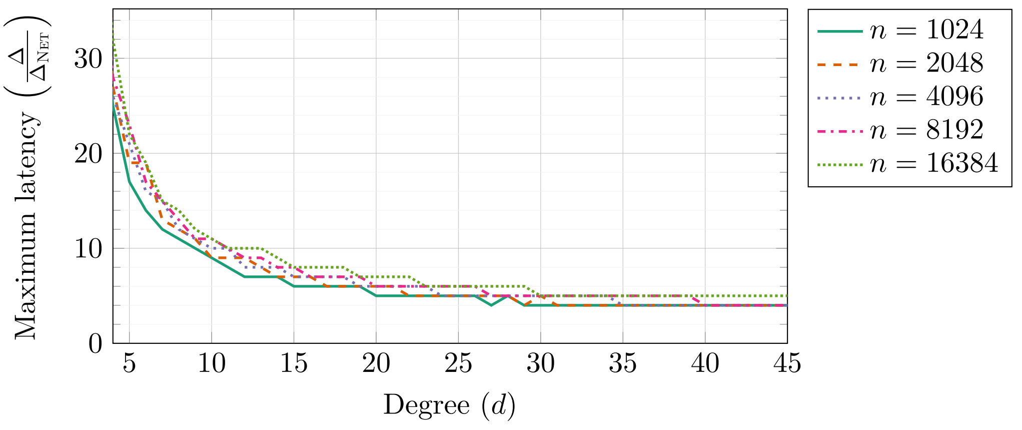

Figure 5: Results for simulations of $\mathsf{FFlood}(d)$ for different values of d and n. The graphs show the maximal latency from the sender to any party that receives the message for different values of d.

8.3 Estimating Parameters for ECFlood

In the previous section, we established that $\mathsf{FFlood}(d)$ constitutes the "best" weak flooding network when d is approximately 4. However, this does not necessarily translate to the protocol $\mathsf{WeakFlood2Flood}$ instantiated with $\mathsf{FFlood}(d)$. The reason is that $\mathsf{WeakFlood2Flood}$ also needs to be instantiated with an (μ, ϱ)-ECCS, and the redundancy of the ECCS scheme is proportional to $(\mu - \varrho)^{-1}$. For $\mathsf{WeakFlood2Flood}$ to be secure, we need that all parties receive at least $\mu - \varrho$ shares. In expectation, all parties receive $\xi \cdot \mu$ when instantiated with a weak ξ-flooding protocol. But for a secure combined protocol, we need that the probability that some parties receive significantly less than $\xi \cdot \mu$ shares is small.

In this section, we provide guidelines for how to select d, μ, and ϱ for a ζ that is (μ, ϱ)-ECCS, such that $\mathsf{ECFlood}(d \cdot n^{-1}, \zeta)$ is both secure and has a small overall factor multiplied to $n \cdot l$ in the communication complexity. We will abuse notation slightly and ignore the WSCAS scheme that is also a parameter of $\mathsf{ECFlood}$, as we will not do any simulations w.r.t. the efficiency of such scheme. Sometimes we will also leave out the ECCS of the notation and treat the number of shares explicitly.

To estimate how to set ϱ of the ECCS, we record the minimum fraction of shares received by any party in any of the simulations (using the same parameters), and plot how this minimum fraction of received shares across all simulations, β, is affected by a varying number of shares μ and degrees d. The results are in Figure 6.

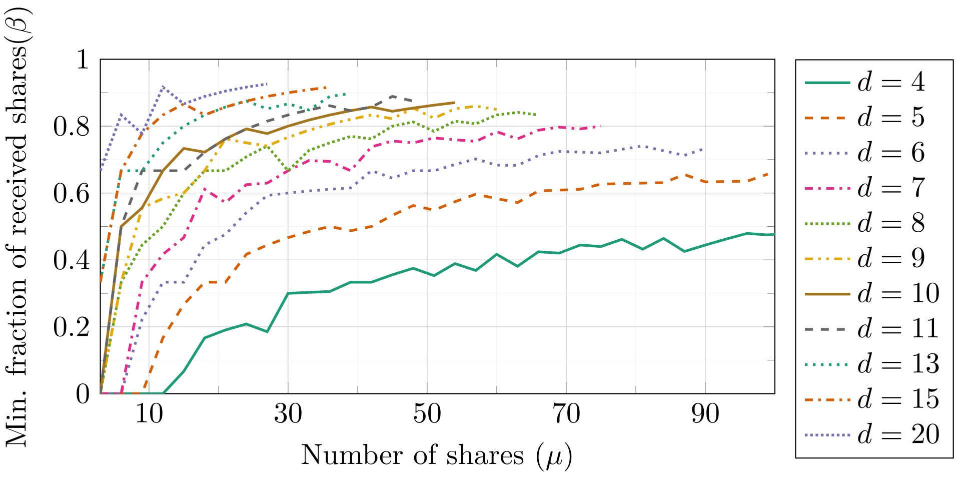

Figure 6: Results for simulations of $\mathsf{ECFlood}(d)$ for different values of d, a fixed number of parties $n = 8192$, and a variable number of shares μ. The graphs show the minimum fraction of shares received by any party in the simulations for different number of expected neighbors $\mu \cdot d$. The number of shares is incremented in steps of 3.

Note that if one instantiates ϱ such that $\mu - \varrho = \beta \cdot \mu \iff \varrho = \mu \cdot (1 - \beta)$ then all parties would be able to reconstruct the message in all simulations. For these parameters, the protocol would have a total communication complexity of roughly $\beta^{-1} \cdot d \cdot n \cdot l$. So we will refer to $\beta^{-1} \cdot d$ as the redundancy of the protocol.

In Figure 7, we plot the redundancy when letting $\varrho = \mu \cdot (1 - \beta)$ as a function of the expected size of the neighborhood $\mu \cdot d$ for a varying number of shares μ and degrees d. We do not plot the redundancy for $d = 3$ and $d = 4$ as it for all considered values of μ will be so high that it is of no interest. Note that even though the expected redundancy has a minimum around $d = 4$ (according to Figure 4), it seems that the actual redundancy of protocols instantiated securely for relatively small expected neighborhoods ≤ 100 is as small for degree $d = 7$. This is because if ξ is higher (provided by a higher degree), then it will concentrate more quickly around the mean (when μ increases) than when ξ is relatively small. Suppose you are willing to accept having 120 outgoing connections. In that case, $d = 7$ and $\mu = 20$ will deliver a flooding protocol that in non of the simulations fails and has a redundancy of just 15. If you are willing to have larger neighborhoods, then we can instantiate our protocol such that redundancy goes below 10 (for example, with $d = 5$ and $\mu = 70$).

For a specific number of shares, $\mu = 20$, we additionally record the percentages of shares where all parties received at least a certain fraction of shares for different parameters d. The results are plotted in Figure 8. The maximum fraction of shares that all parties in all simulations received in Figure 8 corresponds to β in Figure 6. From the plot, it can be seen that if one is willing to accept a low rate of failing executions where not all parties can reconstruct, then one can instantiate ϱ slightly lower than $\mu \cdot (1 - \beta)$.

Finally, we count the maximum number of hops throughout the executions for any party to have received the minimum fraction of shares (at which point all parties could reconstruct the message if parameters were set accordingly). We plot this for a fixed number of shares

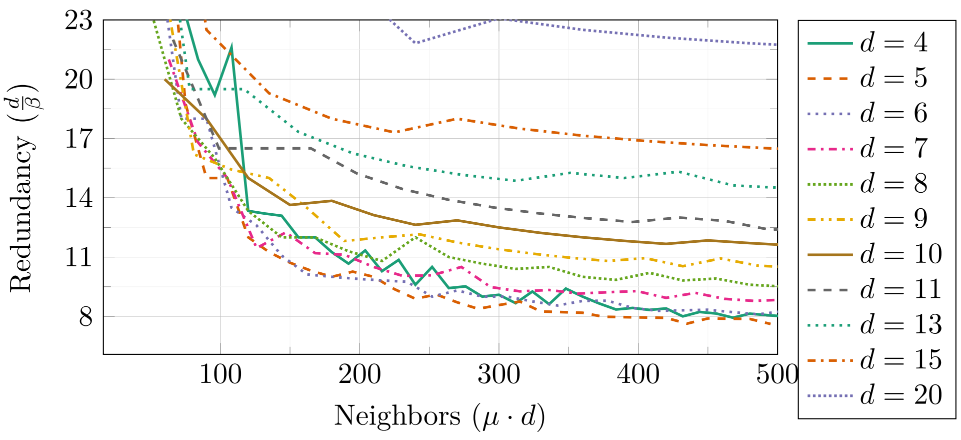

Figure 7: Results for simulations of ECFlood(d) for different values of d, a fixed number of parties $n = 8192$, and a variable number of shares μ. The graphs show the redundancy of the protocol as a function of the expected number neighbors. The number of shares is incremented in steps of 3.

$\mu = 30$ and a varying number of parties μ and internal parameter d in Figure 9. The reason that this is plotted for a fixed number of shares is that we observed that it does not change when increasing the shares. Therefore the results are representative of latencies for all numbers of shares discussed previously.

Surprisingly, the latency for ECFlood is significantly lower than that of FFlood (Figure 5). We believe this is because it is a rare event that the maximum latency of FFlood occurs. So it will become very unlikely that such rare events coincide for shares sent to the same party that receives the minimum number of shares. Note that by adjusting the parameter d, ECFlood can be tuned to achieve a similar latency to that in [LMM+22] while still maintaining a significantly lower redundancy.

Figure 7 shows that increasing d also increases the redundancy and thus the communication complexity. But Figure 9 shows that increasing d decreases the latency. Since there are μ shares to be sent to d parties, the total number of neighbors per parties is $\mu \cdot d$. Figure 7 further shows that increasing the number of shares μ decreases the redundancy. For a practical implementation Figures 7 and 9 can thus be used to find a tradeoff between latency, number of neighbors, and redundancy.

As noted in Section 4, the protocol ECCast, has a redundancy of γ^{-1} (which for the parameters of our simulations are only 2) and a latency of just 2. But to achieve this low redundancy and latency, ECCast requires each party to talk to 8191 neighbors. So the protocols present a tradeoff between low redundancy and diameter and a low number of neighbors.

8.4 Comparison to State-of-the-Art

To compare the practicality of ECFlood to existing approaches for byzantine fault-tolerant protocols, we compare it to FFlood where the number of parties each party forwards to is increased to obtain a decreasing failure probability.[7] To ensure that the comparison does not hide any large constants, and therefore accurately depicts the performance that can be expected in practice, we compare a scenario where both protocols are to sent out a 1 megabyte message (corresponding to a Bitcoin block), and take into account the overhead of the additional nonce, erasure correcting codes, and cryptographic accumulator used in the solution for ECFlood. As

[7]For a discussion of why other classic approaches fail in the byzantine setting, see Section 1.3.

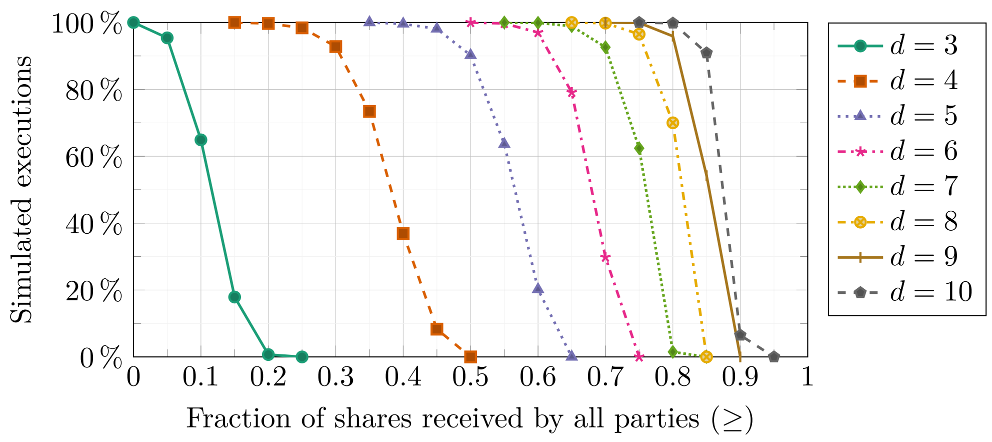

Figure 8: Results for simulations of $\mathsf{ECFlood}(d)$ for different values of d, a fixed number of parties $n = 8192$, and a fixed number of shares $\mu = 20$. The graphs show the percentages of the simulated executions (on the y-axis) where all parties received at least a certain percentage of all the shares sent out by the sender.

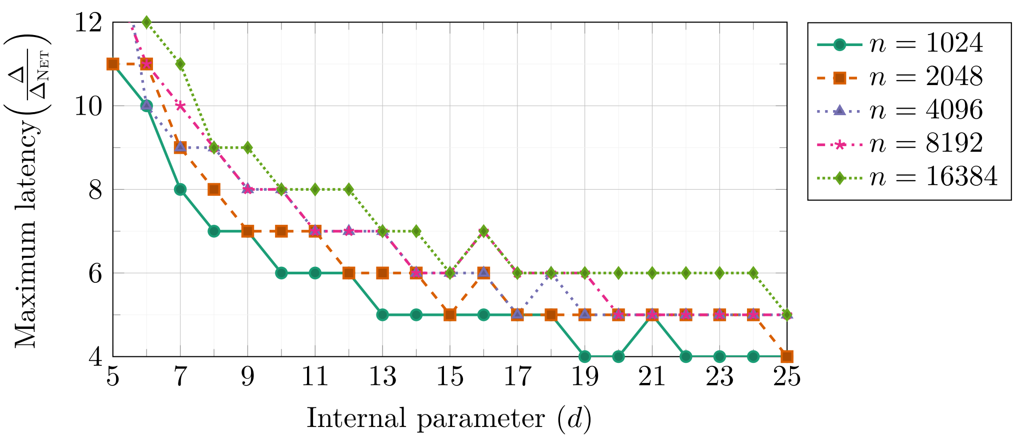

Figure 9: Results for simulations of $\mathsf{ECFlood}(d)$ for different values of d and n, but with a fixed number of shares $\mu = 30$. The graphs show the maximal latency before any party has received enough shares to be able to reconstruct for different values of d.

previously noted, the per-party communication overhead of $\mathsf{ECFlood}(d, \zeta, \alpha)$ is upper bounded by

$$\mu \cdot d \cdot (\zeta.\texttt{ShareSize}(l) + \log(\mu) + \kappa + \alpha.\texttt{ProofSize}(\mu) + \alpha.\texttt{AccSize}). \tag{18}$$

We let ζ be instantiated with Reed-Solomon codes s.t. it is a (μ, ϱ)-ECCS, and let the accumulator α is implemented with a Merkle-tree (using 256-bit hashes). Then the per-party communication complexity in bits is upper bounded by

$$\mu \cdot d \cdot \left(\left\lceil \frac{l}{\mu - \varrho} \right\rceil + \lceil \log(\mu) \rceil + 256 \cdot \lceil \log(\mu) \rceil + 256 \right). \tag{19}$$

This calculation is included in the comparison which is presented in Figure 1, and thereby it depicts the actual number of bits communicated in a real execution, and not only an asymptotic estimation. In the figure, two configurations of $\mathsf{ECFlood}(d)$ are included, where each has a neighborhood of 200 parties (which we deem to be practical for blockchains as the Bitcoin client currently allows up to 125 neighbors by default [GRKC15]). ECFlood(8) is a configuration optimized for redundancy with the number of shares $\mu = 25$ which has a slightly higher latency than FFlood, whereas ECFlood(20) is a configuration optimized for latency with the numbers of shares being $\mu = 10$. We note that the latter configuration has a latency that is as good as the latency of FFlood.

If one wants to virtually eliminate errors our protocol enables a per-party communication of just ∼12 MB to send out a 1 MB message, and if the latency optimized version is used each party communication will be roughly ∼25 MB. FFlood which can be considered state-of-the-art, on the other hand, needs a per-party communication of $\geq$ 45 MB to eliminate delivery errors. We conclude that our protocol allows to drastically (as much as ∼75%) cut the communication complexity at the cost slightly larger neighborhoods.

8.5 Computational Overhead of using a WSCAS and a ECCS

The results of the probabilistic simulations above neglect the computational overhead of using an ECCS and a WSCAS. To ensure that the use of these primitives does not constitute a bottleneck in a practical scenario, we have implemented the protocol WeakFlood2Flood in C++ using open-source libraries for Reed-Solomon codes as ECCS and Merkle trees as WSCAS. The ECFlood would then send the encoded messages using a weak flooding protocol. Hence measuring the overhead of WeakFlood2Flood, allows us to precisely measure the computational overhead incurred by our protocol over existing protocols. We note that our implementation is just a prototype to demonstrate the overall feasibility and is not heavily optimized. It should therefore be possible to get even better results with an optimized production-ready implementation.

To send a 1 MB block using the WeakFlood2Flood protocol with 10 shares and a reconstruction threshold of 8 (which is needed to make all of the ECFlood(20) simulations succeed), the sender needs to encode the block into shares and construct an accumulator with the corresponding proofs. This takes around 12 ms in our benchmark. The time for the other parties receiving the data in the worst case where 2 out of 10 accumulators are corrupted to verify all the accumulator proofs and reconstruct the message amounts to around 16 ms. The benchmarks leading to these numbers were carried out on a desktop with an AMD Ryzen 9 5950X processor.

Based on these benchmarks, we can conclude that the total computational overhead for the sender plus a receiver is less than 30 ms. With a 1 Gigabit internet connection, the parties could communicate less than 4 MB during that 30 ms. As our simulations show, our new protocol reduces the communication complexity per party by around 20 MB for this setting. Since this reduction is substantially more than 4 MB, we conclude that our protocol has an improved efficiency overall.

9 Conclusion

We presented two protocols for message dissemination, ECCast and ECFlood, and showed that these are both asymptotically optimal w.r.t. per-party communication for sufficiently long messages. Our simulations of ECFlood indicate that the protocol is not only asymptotically optimal but actually offers significant improvements in terms of redundancy over existing provably secure protocols for parameters within the practical realm. The two protocols present a tradeoff between a low number of neighbors of each party and the per-party complexity and latency on the other side. Additionally, we presented the protocol Flood2WeightedFlood, which allows our protocols to be used in the weighted setting, which is typical for blockchains.

For blockchain protocols where parties' bandwidth is a limiting factor for the protocol's throughput, our results allow to increase the throughput of the protocol, e.g., by increasing the size of blocks or by simply benefiting from decreased latency of sending fewer bits on the underlying point-to-point channels.

References

[AZV17] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *IEEE Symposium on Security and Privacy*, pages 375–392. IEEE, 2017.

[BDD$^+$21] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In *EUROCRYPT (3)*, volume 12698 of *Lecture Notes in Computer Science*, pages 429–459. Springer, 2021.

[BdM93] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 1993.

[BLZLN22] Amey Bhangale, Chen-Da Liu-Zhang, Julian Loss, and Kartik Nayak. Efficient adaptively-secure byzantine agreement for long messages. *ASIACRYPT*, 2022.

[BP97] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 480–494, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[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 *CCS*, pages 595–608. ACM, 2022.

[CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[DF11] Benjamin Doerr and Mahmoud Fouz. Asymptotically optimal randomized rumor spreading. In *Automata, Languages and Programming: 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II 38*, pages 502–513. Springer, 2011.

[DGH$^+$87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, 1987.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98. Springer, 2018.

[Did09] Frédéric Didier. Efficient erasure decoding of reed-solomon codes. *CoRR*, abs/0901.1886, 2009.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography*, volume 11598 of *Lecture Notes in Computer Science*, pages 23–41. Springer, 2019.

[FH06] Matthias Fitzi and Martin Hirt. Optimally efficient multi-valued byzantine agreement. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 163–168, 2006.

[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.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[GLL$^+$22] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Speeding dumbo: Pushing asynchronous bft closer to practice. *Cryptology ePrint Archive*, 2022.

[GP16] Chaya Ganesh and Arpita Patra. Broadcast extensions with optimal communication and round complexity. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 371–380, 2016.

[GRKC15] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 692–705, New York, NY, USA, 2015. Association for Computing Machinery.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium*, pages 129–144. USENIX Association, 2015.

[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 Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vocking. Randomized rumor spreading. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 565–574. IEEE, 2000.

[KYA22] Ioannis Kaklamanis, Lei Yang, and Mohammad Alizadeh. Poster: Coded broadcast for scalable leader-based BFT consensus. In *CCS*, pages 3375–3377. ACM, 2022.

[LLTW20] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th symposium on principles of distributed computing*, pages 129–138, 2020.

[LMM$^+$22] Chen-Da Liu-Zhang, Christian Matt, Ueli Maurer, Guilherme Rito, and Søren Eller Thomsen. Practical provably secure flooding for blockchains, 2022.

[Mer89] Ralph C. Merkle. A certified digital signature. In *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

[MHG18] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. 2018. https://eprint.iacr.org/2018/236.

[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, *Advances in Cryptology – CRYPTO 2022*, pages 400–430, Cham, 2022. Springer Nature Switzerland.

[MPR07] Ueli Maurer, Krzysztof Pietrzak, and Renato Renner. Indistinguishability amplification. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, pages 130–149, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2005.

[NNT21] Kamilla Nazirkhanova, Joachim Neu, and David Tse. Information dispersal with provable retrievability for rollups. *arXiv preprint arXiv:2111.12323*, 2021.

[NRS$^+$20] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. In *DISC*, 2020.

[ÖMBS21] Ilker Özçelik, Sai Medury, Justin T. Broaddus, and Anthony Skjellum. An overview of cryptographic accumulators. In *ICISSP*, pages 661–669. SCITEPRESS, 2021.

[PS17] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *PODC*, pages 315–324. ACM, 2017.

[PS18] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2018.

[RS60] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of The Society for Industrial and Applied Mathematics*, 8:300–304, 1960.

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

[TC84] Russell Turpin and Brian A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Inf. Process. Lett.*, 18(2):73–76, 1984.

[TCM$^+$20] Muoi Tran, Inho Choi, Gi Jun Moon, Anh V. Vu, and Min Suk Kang. A stealthier partitioning attack against bitcoin peer-to-peer network. In *IEEE Symposium on Security and Privacy*, pages 894–909. IEEE, 2020.

[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.

[W+14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[YMR+19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019.

[YPA+22] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. {DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, 2022.

A Channels in UC

Turing machines in the UC framework communicate with each other by writing directly on their respective tapes. To model unstable and leaky real world communication channels, it is therefore necessary to model each of these channel as an ideal functionality. Below, we recap a simple functionality for time bounded channels.

Functionality $\mathcal{F}^{\Delta}_{\mathsf{Channel}}(s, r)$

The functionality is parameterized by two parties s (the sender) and r (the receiver) and a delivery guarantee Δ. It maintains a mailbox for the receiver `Mailbox` and the set of corrupted parties `Corrupted`.

Initialize: Initially, `Mailbox` $:= \varnothing$.

Send: After receiving (Send, m) from s it leaks ($\mathit{LeakSend}, s, m$) to the adversary.

Get Messages: After receiving ($\mathit{GetMessages}$) from r it outputs `Mailbox` to party r.

Set Message: After receiving ($\mathit{SetMessage}, m$) from the adversary, the functionality sets `Mailbox` $:=$ `Mailbox` $\cup \{m\}$.

At any time the functionality automatically enforces the following property:

1. If m is a message that is input for the first time by an honest party $s \notin$ `Corrupted` at some time t then by time $t + \Delta$ it is ensured that $m \in$ `Mailbox`.

The property is ensured by the functionality automatically making the minimal possible additional calls with *SetMessage*.

We emphasize that even though these channels are time-bounded, the upper bound on the delivery time does not need to be known by parties. The modeling does, therefore, not tie us to a specific synchrony assumption.