Message Dissemination in the Nakamoto Era

Søren Eller Thomsen

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark

Message Dissemination in the Nakamoto Era

A Dissertation
Presented to the Faculty of Natural Sciences
of Aarhus University
in Partial Fulfillment of the Requirements
for the PhD Degree

by
Søren Eller Thomsen
Monday 30th January, 2023

Abstract

In 2008 an alleged pseudonym "Satoshi Nakamoto" published a white paper entitled "Bitcoin: A Peer-to-Peer Electronic Cash System". The white paper is credited with kickstarting the blockchain industry, and the ideas presented there revolutionized the design and practice of byzantine fault-tolerant protocols for state machine replication. This dissertation comprises three publications and one manuscript addressing research questions relevant to the period following the release of the white paper, with a particular focus on message dissemination.

Our first contribution is to generalize and modify the Bitcoin protocol to obtain a protocol with improved throughput under favorable network conditions. We do so by deriving a framework for proving the generalized protocol's security and show how the original Bitcoin protocol instantiates this framework. We then use our framework to propose a new protocol with all Bitcoin's worst-case properties. Still, additionally, under favorable conditions, our new protocol has a throughput proportional to the *actual* latency of the underlying message dissemination protocol instead of the worst-case latency (as Bitcoin).

Next, we present a formal model that quantifies the delay from an adaptive adversary decides to corrupt a party until the adversary effectively gains control over them. Using this model, we provide the first message dissemination protocol that is secure against a "slightly" delayed adaptive adversary instead of a static adversary. We prove that our protocol ensures that each party must only communicate with a logarithmic number of parties to provide a logarithmic delivery latency.

Thereafter, we present a new message dissemination protocol that relies *only* on the assumption that a fraction of resources remains honest. This deviates from the standard assumption for such protocols, i.e., that a certain fraction of parties in a protocol for secure message dissemination remains honest. However, it aligns with the trust assumption commonly used for state-of-the-art state machine replication protocols. Our new protocol, therefore, allows to reduce the trust assumptions of a combined system significantly. The protocol has an efficiency similar to state-of-the-art protocols for the setting where a fraction of parties remains honest.

Finally, we present a new protocol for disseminating messages with asymptotically optimal per-party communication. In this protocol, to spread a message, each party only needs to forward a number of bits linearly proportional to the actual length of the message to ensure a logarithmic delivery latency. We also show how to generically transform some message dissemination protocols to rely only on a fraction of honest resources, thereby making our new protocol apply to the blockchain setting.

Resumé

I 2008 offentliggjorde det formodede pseudonym “Satoshi Nakamoto” en rapport med titlen “Bitcoin: A Peer-to-Peer Electronic Cash System”. Rapporten blev startskuddet til blockchain-industrien, og ideerne i den har revolutioneret både design og brug af fejltolerante protokoller for tilstandsreplikation. Denne afhandling består af tre publikationer og et manuskript, der hver adresserer problemstillinger relevante for perioden efter offentliggørelsen af rapporten med et særligt fokus på protokoller for spredning af beskeder.

Først modificerer vi Bitcoin-protokollen og forbedrer dermed effektiviteten af denne. Vi opnår dette ved at lave en abstraktion af Bitcoin-protokollen og bruger denne abstraktion til at udlede præcise krav til en sikker protokol. Hernæst instantierer vi vores abstraktion med Bitcoin-protokollen og beviser sikkerheden af denne. Endelig bruger vi vores abstraktion til at udlede en ny protokol, der i tillæg til de originale egenskaber under gunstige forhold har en effektivitet, der er proportional til de *faktiske* netværksforhold. Dette er modsat Bitcoin-protokollen, der har en effektivitet der *altid* er proportional til de værst tænkelige netværksforhold.

Herefter præsenterer vi en model, der tillader at kvantificere forsinkelsen, fra en modstander begynder at overtage en part, til at modstanderen rent faktisk opnår kontrol over denne. Vi bruger modellen til at præsentere en protokol for udbredelse af beskeder og beviser sikkerheden af denne over for en adaptiv modstander, der kun er “lettere” forsinket ved overtagelse af parter. Vores protokol sikrer, at alle parter kun behøver at kommunikere med et logaritmisk antal andre parter, for at garantere at tiden, det tager at sprede en besked, maksimalt er logaritmisk i antallet af parter.

Derpå præsenterer vi en ny protokol for udbredelse af beskeder, der *kun* beror på en antagelse om, at en konstant andel af ressourcer er kontrollerede af ærlige parter. Dette er i modsætning til den almindelige antagelse for sådanne protokoller, der er at en bestemt andel af parterne forbliver ærlige. Til gengæld er antagelsen vores protokol beror på i overensstemmelse med standard antagelser for blockchains. Ved brug af vores nye protokol er det derfor muligt at opnå en sikker kombineret protokol, der kun beror på *en* ærlighedsantagelse. Effektiviteten af vores protokol er på niveau med tilsvarende protokoller, der beror på antagelsen om, at en andel af deltagerne forbliver ærlige.

Endeligt præsenterer vi en ny protokol for udbredelse af beskeder med en asymptotisk optimal besked kompleksitet. For at sprede en besked skal hver enkelt part kun sende et antal bits der er lineært proportionalt til antallet af bits i selve beskeden. Ydermere viser vi, at man kan transformere en sådan protokol til at bero på en antagelse om, at en andel af ressourcer er under kontrol af ærlige parter. Derved bliver resultatet også relevant for blockchains.

Acknowledgments

Throughout my PhD studies, I have had the privilege to meet and collaborate with many exceptional people. Their expertise and friendliness have made my time as a PhD student a true joy.

I want to thank everyone in Aarhus Crypto Group for creating an excellent and delightful working environment.

I want to thank each single one of my co-authors: Aslan Askarov, Simon Oddershede Gregersen, Simon Holmgaard Kamp, Chen-Da Liu-Zhang, Bernardo Magri, Ueli Maurer, Christian Matt, Jesper Buus Nielsen, Guilherme Rito, Bas Spitters, and Daniel Tschudi.

I want to thank Vassilis Zikas for hosting my stay abroad and for many interesting discussions.

I want to thank Sabine Oechsner for our countless informal discussions and for always sharing her insights on the many challenges I have encountered during my PhD.

I want to express my sincere appreciation to Christian Matt for the many enjoyable hours we spent collaborating on projects and for his mentorship.

Above all, I am deeply grateful to my supervisor Jesper Buus Nielsen for his unwavering support, for sharing his many insights, and for letting me explore my interests in this thesis.

Søren
Aarhus, Denmark

Contents

1 Introduction

This chapter serves as an introduction to the rest of the dissertation. In Section 1.1, we will provide a high-level introduction to the context of this dissertation. In Section 1.2, we will provide an overview of the publications and manuscripts in this dissertation, and finally, in Section 1.3, we will introduce general notation used throughout the remainder of the dissertation.

1.1 Background

The aim of this section is to give a non-technical overview of the background and context in which this dissertation is written and thereby motivate and clarify the research problems addressed in the later chapters. First, we will introduce the "Nakamoto era" and provide an overview of some of the many research directions that have emerged during this period. Thereafter, we will briefly examine techniques for dissemination of messages related to the results presented in this dissertation.

1.1.1 The Nakamoto Era

The Byzantine Generals Problem first described by Lamport, Shostak, and Pease [LSP82], is a thought experiment that illustrates the challenges of achieving consensus in a distributed system. In the problem, a group of generals is encamped around a city, each with their armies which they command. The generals must decide whether to attack the city or retreat, but they can only communicate with each other via a messenger. However, it may be that some of the generals are traitorous and may send false messages in an attempt to deceive the others.

The problem is to devise a protocol that allows the generals to reach a *agreement* on whether or not to attack, despite the possibility of some generals being dishonest. The challenge is that honest generals cannot distinguish between which generals are honest and which are not, so they must come up with a way to reach a decision based on the messages they receive.

This problem sparked the development of byzantine fault tolerant (BFT) protocols, designed to function correctly even when a subset of the participants act maliciously. In other words, these systems can withstand the "byzantine" behavior of some of their components. The property of being able to deal with such arbitrary faults also transcended to the important area of state machine replication (SMR) (originally introduced by Lamport [Lam78]), where a group of machines is to maintain consistent states. The idea is that if one machine fails, the others can take over and continue providing the service without interruption.

A common approach to achieving such state machine replication is ensuring that all machines agree on a sequence of instructions that each party can execute locally to update their state. Suppose the parties that participate in the protocol *agree* on the sequence of instructions, and the instructions are deterministic. In that case, this approach ensures that all machines will have consistent states. We refer to the sequence of instructions agreed upon as being *totally ordered* and any instruction in the total order as being *final*.

Numerous research papers have studied how to achieve consensus on a total order of instructions. Among them is Nakamoto's white paper [Nak08], which introduced the Bitcoin protocol and revolutionized the field. At the core, Nakamoto's protocol is so simple that it can be described in just two sentences:

1. At all times each party attempts to solve "puzzles"[1] that contain unexecuted instructions and are irrevocably linked to the longest sequence of solved puzzles known to the party.

2. When a party finds a solution to a puzzle, they distribute it to all participants of the protocol.

A solution to a puzzle containing a set of instructions is commonly referred to as a *block*, the process of solving a puzzle as *mining*, a sequence of linked blocks as a *chain*, and the act of distributing a message to all other parties as *multicasting*. A solution to a puzzle can be considered proof that the party has invested a certain amount of computing power. Therefore this security mechanism is referred to as *Proof-of-Work* (PoW). Using this terminology, the protocol can be summarized in just one sentence as follows: Each party continuously mines to extend their best chain and multicasts any newly minted block.

Nakamoto's protocol ensures that if all participants can multicast blocks with a known latency, and less than 50% of the computing power in the protocol is malicious, any prefix of the longest chain known to an honest participant and that is at least $\Omega(\kappa)$ blocks deep (where κ is a security parameter) will be a fixed prefix of the total order of the protocol. Therefore, the instructions contained in any such prefix of the longest chain are final and can be safely executed.

Despite its simplicity, the protocol made several novel improvements to key areas of BFT-SMR:[2]

Scalability: Nakamoto's protocol is extremely scalable. To ensure that a specific instruction becomes final, only $\Theta(\kappa)$ blocks need to be appended to it (for a security parameter κ), and therefore only $\Theta(\kappa)$ multicasts needs to be processed. Furthermore, each block itself also contains instructions, and so the amortized multicast complexity to finalize a block becomes just $O(1)$!

[1] For the purpose of this discussion, it is not relevant how these puzzles concretely are instantiated, but Nakamoto's original instantiation was to let each party group a set of transactions and append different nonces (numbers only used once) to this set of transactions, and the hashed value of the previous puzzle. A solution to the puzzle would then be a set of transactions, a nonce, and a hash of the previous puzzle, which, when hashed, has a value below some predefined threshold.

[2] Many of the ideas in the protocol were not new but combining them in this way was unique. For a historical account of where the ideas originally arose from, we refer to [NC17]

Permissionless: Rather than relying on a fraction of the parties participating in the protocol behaving honestly, Nakamoto's protocol instead depends on a fraction of the computing power following the protocol. That is, there is no trust placed in the identities of parties which is very difficult to establish since a single user may control several Sybils. Thereby, the protocol does not need to impose any restrictions on *whom* is allowed to participate in the protocol, and as such, it can be open for everyone to participate! In fact, the protocol does not even require all participants to know each other as long as it is possible to multicast messages among all parties.

Robustness: Not only is Nakamoto's protocol able to withstand up to 50% percent of the computing power behaving maliciously, but there is also no requirement that the set of honest parties is static. In fact, as no private state needs to be preserved between consecutive mining attempts, the protocol can withstand an adversary that can adaptively choose which machines they control (as long as less than 50% of the computing power remains honest throughout the protocol). The only requirement on this adaptivity from the adversary is that the adversary cannot prevent a multicast that has been initialized from succeeding. This makes the protocol extremely robust towards Denial-of-Service attacks, which have become increasingly common in recent years.

With these properties, the protocol gave birth to the cryptocurrency industry, which has since evolved into a multi-billion dollar industry. The technology that underpins this industry is known as "blockchains", and we will refer to the period following the release of this white paper, as the *Nakamoto era*.

1.1.2 Research Directions in the Nakamoto Era

With its groundbreaking ideas, Nakamoto's original article sparked an abundance of research in the area of BFT systems. Below we provide a selective overview of some of the directions that have been pursued since Nakamoto's original paper, focusing on provable security using the cryptographic methodology. Note that the overview is not intended to be exhaustive but merely to highlight some important directions to help contextualize this dissertation.

Security of the Bitcoin protocol. In Nakamoto's original white paper, no precise properties of the protocol were defined, nor was the protocol's security proven. Since then a significant line of research has addressed this shortcoming [Bad+17; Dem+20; GKL20; GKL15; GKL17; GRR22; Niu+19; PSS17; Ren19]. Garay, Kiayias, and Leonardos [GKL15] introduced and proved the properties of *chain growth*, *chain quality*, and *common prefix*, which together ensures both safety and liveness of the induced SMR protocol. Since then, have analyses of the protocol been extended to cover more difficult network conditions and varying participation [GKL20; GKL17; PSS17], and other work has presented alternative analyses that aim to be simple [Dem+20; Niu+19; Ren19], the security of the protocol has been proven in a composable model [Bad+17], and also practical bound on the time it takes to make a block final has been established [GRR22].

Energy efficiency. The security of Nakamoto's original protocol crucially relies on the fact that a block is only valid if it is an actual solution to a puzzle and that honest parties create more such solutions to puzzles than dishonest parties. The security, therefore, increases the more honest computing power that is spent solving such puzzles. A significant drawback of this security mechanism is that constantly solving puzzles is very costly energy-wise.

To address this issue while retaining the permissionless property and the robustness of Nakamoto's protocol, various trust assumptions have been proposed [Ate+14; CM19; DPS19; Dav+18; Dzi+15; Kia+17]. The most popular alternative has become *proof of stake* (PoS) protocols, where each party participates in the protocol weighted by their current holding of stake on their account [CM19; DPS19; Dav+18; Kia+17]. Another alternative is *proof of space* where the underlying resource that parties participate in a protocol is the amount of storage available to them [Ate+14; Dzi+15]. These approaches reduce the energy consumption to only a fraction of PoW protocols.

Instruction throughput and time to finality. Independently of which underlying resource assumption is used, Nakamoto's original protocol design has two limitations:

1. The security of Nakamoto's protocol relies on the fact that a chain produced only by honest parties must be able to outgrow a chain made only by dishonest parties. Therefore, it is crucial for the protocol's security that honest parties can propagate previous blocks before a new block can be produced, as honest blocks otherwise might not contribute to growing a long chain. Consequently, an essential parameter of the protocol is the *block production rate*.[3] If the block production rate is set to high compared to the actual time it takes to multicast a block, then the *safety* of the protocol may be violated. Furthermore, if the number of instructions in a block is increased, it may take longer to multicast such a block. Therefore, it is inherently limited how many instructions can be totally ordered in Nakamoto's protocol. For example, the Bitcoin protocol has an average block production rate of 10 minutes, whereas Ethereum's block production rate is roughly 12 seconds pr. block. We will refer to the number of instructions that can be put in a total order as the instruction throughput of a protocol.

2. It is not only the instruction throughput that is affected by this limit on the block production rate. Because an instruction needs to be buried by $\Omega(\kappa)$ blocks before it becomes final, the time it takes from an instruction makes it into a block until it becomes final will be $\Omega(\kappa/R)$ if the protocol has a block production rate of R. That is, the *confirmation time* of the protocol is also inherently limited.

A plethora of research has gone into addressing these two issues and below we will mention a selection of it.

Some projects have tried to preserve Nakamoto's protocol's overall "style" while increasing the throughput [Eya+16; LSZ15; PS17a; SZ15].[4] Eyal, Gencer, Sirer, and

[3]In practice, the block production rate is controlled by how the difficulty of the puzzles that are to be solved is set.

[4]The motivation for some of these works is also to mitigate rational attacks (one that the protocol incentives participants to do) on Nakamoto's protocol known as *selfish mining* [Eya+16; PS17a]. As this is out of the scope of this dissertation, it will not be discussed further.

Renesse [Eya+16] proposes a new protocol, Bitcoin-NG, where they let the miner of the last block append additional instructions to this block until a new block arises. Thereby, the overall throughput of the protocol is increased because the period between the blocks' production is no longer unused. Sompolinsky and Zohar [SZ15] introduces the GHOST-rule (Greedy Heaviest-Observed Sub-Tree), which instead of simply letting all parties prefer the longest chain, also takes into account the blocks that does not make it into the main chain when selecting the preferred chain. Lewenberg, Sompolinsky, and Zohar [LSZ15] and Pass and Shi [PS17a] obtain a higher throughput of instructions by ordering instructions that do not appear on the main chain in a DAG. In Chapter 2, we will present a new NSB that under certain optimistic conditions will have increased throughput.

A completely different approach than Nakamoto's probabilistic approach was taken in original effective solutions to the BFT-SMR problem [CL99]. The idea of Castro and Liskov [CL99] was to let a leader propose instructions that a committee of parties then validates. Once a party has heard from sufficiently many such validators, they will accept the instruction as final. Since then, the scalability and throughput of this protocol type have been vastly improved [Abr+20; Kot+07; Mil+16; Yin+19]. Contrary to Nakamoto's protocol, this type of protocol does, however, require trust in the identities of the participating parties. Therefore the protocols are necessarily permisioned to stay secure and require the parties to stay honest for more extended periods of time.

A breakthrough for this type of protocol was recently made by Gilad, Hemo, Micali, Vlachos, and Zeldovich [Gil+17] with the introduction of the Algorand protocol. The protocol is a committee-based BFT-SMR, but instead of letting the same parties stay in the committee, each activity in the protocol is abstracted into a "role", and then these are occupied using PoS as a resource together with a VRF (verifiable random function) to select a party at random but weighted according to the fraction of stake they own. Because parties can take over from each other, the authors define this as a protocol being *player-replaceable.* This approach has three advantages:

1. Instead of having all parties be a part of the committee, it is sufficient to let the committees be of size $O(\kappa)$ because then the ratio between adversarial and honest players in the committee will be roughly equal to the balance between honest and adversarial stake. The smaller committee size drastically reduces the number of multicasts to finalize a block to be just $O(\kappa)$ making the protocol *scalable.*

2. Because the protocol utilizes PoS as a resource, no trust is needed to be placed on the identities of the respective parties, and thereby the protocol becomes *permisionless.*

Finally, it should be mentioned that there has also been research in combining the robustness of NSBs with the fast confirmation time of committee-based approaches using hybrid approaches [PS18a] and "finality layers" [BG17; Din+20; Kam+22; SK20]. Pass and Shi [PS18a] introduces a hybrid approach where a NSB is run, but additionally, a leader is selected that interacts with a committee. If this party is honest, they achieve a speedup in terms of both throughput and finality time. [BG17; Din+20; SK20] introduces an additional layer on top of NSB finality where a committee tries to detect the prefix of a chain that honest parties already agree upon and agree on an irrevocable certificate such that this prefix becomes final immediately.

All of the protocols referenced above that obtain a superset of Nakamoto's original protocols properties has one thing in common: To deliver on their promise of scalability, robustness, and being permissionless, they rely on a multicast network! In particular, if the underlying multicast network does not have these properties, neither will a protocol build on top. The focus of Chapters 3 to 5 will be to design multicast networks with such properties and prove them secure.

Next, we introduce multicasting more carefully and review the state-of-the-art solutions for this problem.

1.1.3 Multicast

For the purpose of this dissertation, *multicasting* is the act of disseminating a message known by a single party to the complete set of parties participating in a protocol.[5] To devise such a protocol, some network topology of point-to-point channels between parties needs to be assumed. In this dissertation, the focus will be on complete topologies where each pair of parties can communicate. The motivation for this is that the protocols we are interested in are intended to run on the public internet, and as such, it is possible for all notes to communicate.

If such a complete topology is assumed, a straightforward way of performing a multicast is to let the party that knows the message send it to all other parties in the protocol. This elementary protocol achieves the optimal message complexity (the number of messages sent to deliver a single message) of just $n-1$ (for n parties). However, its simplicity also imposes an immense workload on the initial party knowing the message that is obliged to send it to all other parties. In a decentralized setting, parties' bandwidth and computational power are often limited; therefore, this approach is not feasible. To deem whether a particular multicast protocol is appropriate for a specific use-case, it is necessary to consider the number of neighbors each party needs to communicate with, the per-party communication, and the delivery latency of such protocol.

Since the origin of computer science, there has been extensive research in devising such efficient multicast protocols. There are two main approaches to implementing multicast: deterministic and probabilistic.

A deterministic multicast approach uses a pre-defined and fixed graph, such as a spanning tree, to deliver messages. This approach is predictable and provides guaranteed delivery of multicast packets to all participants if all parties behave honestly. However, with its predictability, it is, unfortunately, not very robust against failures and, in particular, very vulnerable to a byzantine adaptive adversary.

On the other hand, a probabilistic multicast approach uses a randomized algorithm to let each party select to whom to forward the message. When receiving a message, a party will typically forward it to another random selection of peers. If the protocols increase the redundancy and are high entropy, they can be way more robust w.r.t. byzantine failures and still be effective w.r.t. both per-party communication and latency. Due

[5]This act is also referred to as broadcasting or flooding a message. However, in the literature, broadcast often refers to a primitive where parties must agree on which message they received from a sender. Therefore to avoid further confusion, we will use either multicasting or flooding in this dissertation.

to such protocols' resemblance with how a rumor spreads, this type of protocol is often referred to as gossip protocols. This type of protocol will be the main focus of this dissertation due to its robustness properties. Next, we provide a small selective review of research on gossiping protocols.

Gossiping. The first to consider gossip protocols for forwarding messages in a replicated database was Demers, Greene, Hauser, Irish, Larson, Shenker, Sturgis, Swinehart, and Terry [Dem+87].[6] In this work, the authors considered a model where servers periodically check for updates. In this model, they devise two protocols: The "anti-entropy" protocol, wherein each period, the server pushes an update to a random neighbor, and the "rumor-mongering" protocol, where parties keep pushing a "hot rumor" to new neighbors until a certain number of people have told them they already have heard about this rumor. Both protocols were implemented and deployed on Xerox's servers and showed to perform very well in practice.

Feige, Peleg, Raghavan, and Upfal [Fei+90] showed that if in each period a rumor is sent to a random party, then it takes only $O(\log(n))$ periods until all n parties have learned about this rumor. Thereby, the message complexity of the protocol will be $O(n \cdot \log(n))$. Karp, Schindelhauer, Shenker, and Vöcking [Kar+00] showed that by letting parties that have not learned the message additionally pull for news, then only $O(n \cdot \log\log(n))$ messages have to be sent. This was even further improved by Doerr and Fouz [DF11], who showed that by steering the randomness based upon whether or not the last party had already received the message, then only $O(n)$ messages have to be sent to spread a rumor in only $O(\log(n))$ periods. We note protocols that are reactive in the sense that parties choose to forward a message depending on which messages they receive (such as the protocols in [DF11; Kar+00], makes them inherently difficult to apply in a byzantine setting where parties may send wrong information.

In the seminal work of Kermarrec, Massoulié, and Ganesh [KMG03], a protocol where each party forwards a message to each other party with a certain independent probability is considered.[7] Due to the similarity with Erdős–Rényi graphs [ER60], where each edge appears with equal independent probability, we will refer to this as Erdős–Rényi-style gossiping. They show that for $n \to \infty$, each party needs to connect to $\log(n)$ parties in this way to ensure that a message reaches everybody with overwhelming probability. Additionally, it is shown that this protocol is resilient to removal of random nodes.

Gossip techniques have been applied in many more areas of computer science. It has also been considered how to ensure that a message is either spread to very few or almost all parties [Bir+99; HHL06] known as "bimodal" gossip and how to take into account spatial distance when gossiping [KKD04], how aggregated values can be computed using gossip-based communication [KDG03], and how gossiping can be done in mobile ad-hoc networks where the topologies considered are often incomplete [Cri+09; Gut+15; Hu+12; SCS03].

However, to the author's best knowledge, none of these directions aim to ensure a multicast network that is secure against deliberate attacks, and therefore as such, not applicable if one wants robustness properties compatible with BFT-SMR protocols

[6]For a historical overview of early research on the topic, we refer to [HHL88].

[7]This protocol is also used in Chapters 3 to 5.

of the Nakamoto era. Next, we give an overview of gossiping techniques applied in the byzantine setting and finally provide an overview of the very little research on multicasting specifically for blockchains in the Nakamoto era.

Gossiping techniques in the BFT setting. Gossiping techniques have also been used in the BFT context to reduce the message complexity and robustness of these. One line of work [MMR99; MPS01; MS03] considers how to efficiently propagate an update that is initially known by more processes than those that may fail to all nodes in a system. The ideas do however only have limited transferability to the multicast setting, where it may be that only a single party knows the initial message.

Chandran, Chongchitmate, Garay, Goldwasser, Ostrovsky, and Zikas [Cha+15] used gossiping techniques to improve the *communication locality* (the number of parties each party needs to communicate with) of secure multi-party computation. Concretely, they let parties connect in a Erdős–Rényi pattern and show how this can be used to achieve *reliable message transfer*, which they use to run MPC protocol. By assuming that communication is hidden and parties can instantly erase memory such that an adversary does not learn the communication pattern of a party even if this party is corrupted, they prove their protocol secure against an adaptive adversary.

Guerraoui, Kuznetsov, Monti, Pavlovic, and Seredinschi [Gue+19] used gossiping in Erdős–Rényi style and subset sampling to obtain a protocol with reduced communication cost for *reliable* broadcast in the setting where less than one-third of parties are controlled by an adversary. This work considered only a static adversary that corrupted a fixed set of parties. Tsimos, Loss, and Papamanthou [TLP22] used Erdős–Rényi style gossiping to reduce the complexity to improve efficiency for both reliable broadcast and reliable parallel broadcast by cleverly aggregating information along the gossiping. Noticeably, they manage to make their parallel broadcast algorithm secure even for adaptive adversaries by hiding the communication pattern using only encryption, making it impossible to eclipse an honest party with noticeable probability. They can do so without increasing the communication complexity because the parallel broadcast problem inherently requires all-to-all communication.

Multicasting for blockchains. Compared to the immense amount of work focused on proving security and improving the efficiency of BFT-SMR in the Nakamoto era, secure and efficient multicast for blockchains has received very little attention.

A recent line of work [Lon+21; Mao+20; RT19; RT21; Sal+22; VT19; Vyz+20; Wan+22] seeks to optimize the efficiency and security of the multicast protocols particular for the blockchain setting. However, these works mainly optimize for efficiency and only apply heuristics to prevent specific attacks. In particular: none of them is *provable* secure, and they are, as such incomparable to the works presented in this dissertation, even though the end goal is similar.

Another line of work has focused on attacking the underlying multicast network of blockchains currently deployed [AZV17; Hei+15; MHG18; Tra+20]. This serves as additional motivation for investigating provably secure solutions for multicasting in the blockchain area that are secure using the same assumptions as the blockchain protocols themself (see Chapters 4 and 5).

Concurrently with and independently of this dissertation, Coretti, Kiayias, Moore, and Russell [Cor+22] proposed a message dissemination protocol for the Ouroboros Praos [Dav+18]. They show how selectively propagating updates can be used to avoid a known denial of service attack on the protocol, where the network is spammed with many blocks. Assuming only that a majority of the stake in the system behaves correctly, they show that with overwhelming probability, all but a small fraction of the stake receive the necessary updates. They additionally show that this weakened functionality is sufficient for the Ouroboros Praos protocol to achieve a weakened version of SMR.

1.2 Overview

This dissertation taps into the cryptographic tradition of carefully stating and proving properties within a well-defined model. We apply this methodology to address challenges in the Nakamoto era with a particular focus on message dissemination. The remainder of this thesis is divided into four chapters.

In Chapter 2, we make a tweak to Nakamoto's original protocol by generalizing what parties consider the best chain. Our generalization of the protocol allows it to achieve additional properties and, under favourable circumstances, improves the throughput and finality time compared to the Bitcoin protocol.

In Chapter 3, we introduce the first precise model that quantifies the adaptivity of an adversary and show how to construct an efficient multicast protocol that is secure against certain adaptive adversaries within this model. This allows the robustness of blockchains w.r.t. an adaptive adversary to be stated precisely without assuming a multicast functionality.

In Chapter 4, we devise the first multicast protocol that is provably secure under similar resource assumptions as SMR protocols in the Nakamoto era and thereby lay the ground for a genuinely permissionless combined construction.

In Chapter 5, we introduce a multicast protocol that is asymptotically optimal and additionally shows how the techniques of the previous chapter extend to work in a black-box manner. With our black-box transformation, our optimal multicast protocol can be used directly as a foundation for a SMR protocol in the Nakamoto era without introducing additional trust assumptions.

Chapters 2 to 4 are based on existing peer-reviewed publications, whereas Chapter 5 is based on a manuscript that at the time of writing is being reviewed. The author of this dissertation has been a part of all aspects of these research projects. In particular, the author has contributed to idea generation, proving technical results, evaluations, and writing the articles.

Below is an overview of the individual chapters and their relation to each other. Each chapter will additionally come with a separate introduction and related work and can be read as a separate entity. Throughout the rest of this dissertation, "we" refers to the authors of the individual publications and the manuscript, while "I" refers to the author of this dissertation.

1.2.1 Chapter 2: Weight-Based Nakamoto-Style Blockchains

The chapter is based almost verbatim on [Kam+20], which is the full version of the publication [Kam+21].

[Kam+20]: Simon Holmgaard Kamp, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, Søren Eller Thomsen, and Daniel Tschudi. *Weight-Based Nakamoto-Style Blockchains*. Cryptology ePrint Archive, Paper 2020/328. 2020. URL: `https://eprint.iacr.org/2020/328`.

[Kam+21]: Simon Holmgaard Kamp, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, Søren Eller Thomsen, and Daniel Tschudi. "Weight-Based Nakamoto-Style Blockchains". In: *LATINCRYPT*. vol. 12912. Lecture Notes in Computer Science. Springer, 2021, pp. 299–319.

Below I will explain the motivation for the project, provide a high-level summary of its contributions, and place it in its immediate context.

Motivation. As explained in Section 1.1, it is crucial for the security of both Nakamoto's original protocol and subsequent protocols in a similar style that they are instantiated with a correct block production rate w.r.t. the latency of the multicast network. Hence, this block production rate must be instantiated conservatively to accommodate the worst-case latency that the network may experience throughout the entire lifetime of the protocol. The latency of a multicast network may, however, greatly depend on the conditions of the underlying point-to-point channels, which, if realized by the public internet, again varies depending on the amount of network traffic at the time. Therefore there may be a big gap between the worst-case performance of multicast networks and the normal-case performance. Current, NSBs are not able to utilize the actual latency of the network (which may be way better than worst-case latency) because the block production rate has to be fixed once and for all when the protocol is deployed. The motivation for this work is to improve upon this state of affairs.

Contributions. In this work, we make a small change to the design of NSBs to try to utilize this wasted potential. We do so by modifying the "longest-chain-rule", which says that parties should try to extend the *longest* chain they know of, to instead assign a weight to each block based on the hash of the block and let parties extend the *heaviest* chain they know. At an intuitive level, we remove the requirement that a block needs to be a strict solution to a puzzle for it to be valid but instead introduce a judgment of "how good a solution to a puzzle" a block is.

In more detail, we introduce an abstract function referred to as *weight function*, and prove that Nakamoto's protocol with our modifications has both chain growth, chain quality, and common prefix when the used weight function has specific properties. Together with generic bounds on the production of weight for certain classes of weight functions, this constitutes a framework that allows easy exploration of different weight functions. We show the applicability of our framework by applying it to two examples of weight functions:

1. We cast Nakamoto's original longest chain rule as a 0/1-weight function within our framework. That is, we assign a weight of 0 to all blocks with a hash above

the threshold that determines if a block is valid in Nakamoto's protocol and a weight of 1 to blocks with a hash below. For this function, we use our framework to derive settlement bounds similar to state-of-art analyses.

2. We instantiate our framework with a "capped exponential weight function" and show that this, in addition to achieving worst-case guarantees similar to the Bitcoin protocol, also achieves a weak form of optimistic responsiveness (explanation follows below).

The capped exponential weight function is a function that increases exponentially in the hash value until it is capped at a certain threshold. The intuition for this type of function is that the threshold should be set such that the production rate of the blocks above the threshold enables a secure protocol during the worst-case latency periods of the network. Blocks with hashes lower than this threshold do not weigh enough to harm the worst-case settlement guarantees. Still, they allow the parties to extend the total order at a rate that depends on the actual network latency if no parties behave maliciously (referred to as being weakly optimistic responsive). While this may seem like a fragile property (as it is not clear when parties are malicious and when they are not), we envision that combining our protocol with a finality layer [BG17; Din+20; Kam+22; SK20] that detects what honest parties agree on and issues certificates on this, could be a compelling combination.

Closely related work. This research project fits into the line of research making tweaks to Nakamoto's original protocol [Eya+16; LSZ15; PS17a; SZ15]. Even though these previous protocols manage to increase the throughput of Nakamoto's protocol, they do not improve upon the finality time. Under favorable conditions, our suggested protocol does both.

The Thunderella protocol by Pass and Shi [PS18b] achieves a more robust version of optimistic responsiveness by running a committee on top of a NSB. With this approach, they achieve a speedup w.r.t. both throughput and finality time when the leader of their committee is honest, and less than 25% of the parties in the committee behave maliciously. By needing to appoint a leader, their protocol is more vulnerable to denial-of-service attacks, as this leader must stay "alive" for a more extended time to gain the speedup.

In the context of the remainder of the dissertation, the chapter mainly serves as a concrete illustration of why and how the security of NSBs inherently depends on the underlying multicast network.

Differences to original work. The differences between the chapter appearing in this dissertation and the full version of the publication [Kam+20] are:

- An appendix that provided a "weighted" version of the chain-quality theorem is blended into the chapter instead of appearing as an appendix.
- A brief outline of the chapter has been added.
- An appendix that contained additional bounds for the production of weight for weight functions different from the ones appearing in this work is left out.

- Minor phrasings have been changed to improve readability, and typos have been corrected.
- A section that concluded on the work has been removed as the contributions have already been summarized earlier.

1.2.2 Chapter 3 Formalizing Delayed Adaptive Corruptions and the Security of Flooding Networks

This chapter is based almost verbatim on [MNT22a], which is the full version of the publication [MNT22b].

[MNT22a]: Christian Matt, Jesper Buus Nielsen, and Søren Eller Thomsen. *Formalizing Delayed Adaptive Corruptions and the Security of Flooding Networks.* Cryptology ePrint Archive, Paper 2022/010. 2022. URL: https://eprint.iacr.org/2022/010.

[MNT22b]: Christian Matt, Jesper Buus Nielsen, and Søren Eller Thomsen. "Formalizing Delayed Adaptive Corruptions and the Security of Flooding Networks". In: *CRYPTO (2).* Vol. 13508. Lecture Notes in Computer Science. Springer, 2022, pp. 400–430.

Below I will explain the motivation for the project, provide a high-level summary of its contributions, and place it in its immediate context.

Motivation. One of the main selling points and key innovations of NSBs in comparison to previous work is their robustness (as mentioned in Section 1.1) against an adaptive adversary. A significant line of research has shown that if just each honest party in the protocol stays honest long enough to be able to multicast a single message within a known bounded time, and at all times less than 50% of the underlying resources are under the control of the adversary, then this type of protocol is both safe and live. If multicast is implemented by letting the sender simply forward a message to all parties and atomicity of this is assumed, then this is guaranteed even against an adaptive adversary. On the other hand, any protocol where an initial sender sends to fewer parties than those that can be adaptively corrupted can not guarantee delivery because the neighbors of the initial sender may be corrupted. In practice, efficient multicast networks are implemented using gossiping techniques, and by the above argument, this implies that no guarantees can be given against an adaptive adversary for these systems. Phrased differently: No security guarantees can be given against an adaptive adversary against currently running NSBs. However, intuitively it seems that gossip protocols should be resistant to a slightly weaker notion of adaptivity, which excludes the type of attack mentioned above. This research project aims to redeem provably security for gossip protocols in such a slightly weaker adversary's presence.

Contributions. We formalize the notion of δ-delayed adaptive adversaries within the UC framework [Can20] that is considered the gold standard for cryptographic security. These are adaptive adversaries for whom it takes δ time from they initiate the process of corrupting a party until they control the party. In this model, we prove the security of a very simple protocol where each party uses an independent coin for each party to decide whether or not to forward a particular message to them. This protocol is secure against an adversary that is delayed just for the time it takes to send a message over a point-to-point channel plus the time it takes to resend a message. In a setting with n parties, point-to-point channels with a maximum delivery latency of Δ, an adversary able to corrupt any constant known fraction of the parties in, and a security parameter κ, we prove it secure for two different sets of parameters: 1) if each party resends each message to $\Theta(\kappa + \log(n))$ parties, then this will ensure a maximum delivery latency of $O(\log(n) \cdot \Delta)$ with a probability overwhelming in κ, and 2) if each party resends each message to $\Theta\big(\sqrt{(\kappa + \log(n)) \cdot n}\big)$ parties then this ensures a maximum delivery latency of just $2 \cdot \Delta$ with a probability overwhelming in κ.

As a sanity check for our modeling, we also prove the intuitive result that security against a "fast" adversary implies security against a "slow" adversary and that our formalization of adaptive adversaries is compatible with the standard modeling of adaptive adversaries in UC. That is, we prove that a 0-delayed adversary corresponds to a standard adaptive adversary. Together, these two results allow using the UC composition theorem to achieve security when instantiating protocols that rely on a mix of constructions proved secure against a standard adaptive adversary, and others proved secure against a delayed adversary.

Closely related work. Pass and Shi [PS17b] introduced the notion of δ-agile adversaries that are conceptually identical to our δ-delayed adversaries, which they used to define adaptive security for their committee-based consensus mechanism (if an adversary can corrupt the committee, their protocol is insecure and therefore their protocol is only secure against an adversary that is delayed until the next committee is selected). Contrary to our work, they did not provide precise execution semantics for their notion of delayed adversaries.

Chandran, Chongchitmate, Garay, Goldwasser, Ostrovsky, and Zikas [Cha+15] considered a model where an adversary does not learn which channels are used for communicating between honest parties. Within this model, they constructed an MPC protocol where each party only talks to a polylogarithmic number of parties. They proved the security of their protocol against a fully adaptive adversary by additionally assuming that each party can securely erase their memory such that by a subsequent corruption, the adversary would not learn information about with whom they communicated.

Kermarrec, Massoulié, and Ganesh [KMG03] considered the same protocol as us and proved that it has a connectivity threshold with a logarithmic number of neighbors. They additionally proved that the protocol remains connected when a fraction of the nodes are removed (and the parameters are adjusted accordingly). However, contrary to our work, they did not consider any form of adaptive adversaries.

Differences to original work. Apart from minor differences (such as rephrasings and correction of typos) between this dissertation's chapter and [MNT22a], the only change are:

1. To make the results easy to compare to the results in Chapters 4 and 5, a simplification on the asymptotic number of neighbors, possible within the UC framework,

has not been applied to the presentation of the results. A remark on this has been added (Remark 3.6.1).

2. An appendix is left out. In this appendix, a proof from [Cha+15] that showed the Erdős–Rényi graphs had a logarithmic diameter when each node is connected to an expected poly-logarithmic number of neighbors was replicated to obtain concrete bounds.

3. A section that concluded on the work has been removed as the contributions have already been summarized earlier.

1.2.3 Chapter 4 Practical Provably Secure Flooding for Blockchains

This chapter is based almost verbatim on [Liu+22a], which is the full version of the publication [Liu+22b].

[Liu+22a]: Chen-Da Liu-Zhang, Christian Matt, Ueli Maurer, Guilherme Rito, and Søren Eller Thomsen. *Practical Provably Secure Flooding for Blockchains*. Cryptology ePrint Archive, Paper 2022/608. 2022. URL: `https://eprint.iacr.org/2022/608`.

[Liu+22b]: Chen-Da Liu-Zhang, Christian Matt, Ueli Maurer, Guilherme Rito, and Søren Eller Thomsen. "Practical Provably Secure Flooding for Blockchains". In: *ASIACRYPT*. vol. 13791. Lecture Notes in Computer Science. Springer, 2022, pp. 774–805.

Below I will explain the motivation for the project, provide a high-level summary of its contributions and place it in its immediate context.

Motivation. A key innovation of the Nakamoto era was to design protocols such that no fraction of parties needed to be trusted but instead relied on a fraction of some underlying resource behaving honestly. However, as summarized in Section 1.1, most protocols in this era rely on trust in some underlying resources *and* a functioning multicast network. Until this work, there have been no multicast networks that have been designed to work based on a resource assumption different from the fraction of parties, and as a consequence, currently deployed blockchains de facto rely on both an assumption on the amount of honest stake or computing power *and* an assumption on a fraction of parties behaving honestly. The latter is an overly optimistic assumption on the public internet, as Sybil attacks are cheap and straightforward to deploy. Furthermore, widely used blockchains have been shown to be vulnerable to such attacks in practice [AZV17; Hei+15; MHG18; Tra+20].

This work is motivated by remedying this state of affairs such that the security of blockchains can be guaranteed, assuming only a constant fraction of the resources behave honestly.

Contributions. In a model where there are n parties, and each party p is assigned a public fraction of weight α, we propose a new multicast protocol: For some parameter k, we let each party p forward a received message to $k \cdot \lceil \alpha_p \cdot n \rceil$ distinct parties chosen using weighted sampling without replacement and where each party p' again is weighted by $\lceil \alpha_{p'} \cdot n \rceil$.

Despite the protocols simplicity, we prove that it ensures delivery to all parties by letting $k = \frac{\log(n)+\kappa}{\gamma}$ by only assuming that a constant fraction γ of the total weight remains honest. If the delivery latency of the underlying point-to-point channels is bounded Δ, then the delivery latency of the protocol will be just $O(\Delta \cdot \log(n))$ and the total message complexity will be just $O(n \cdot (\log(n) + \kappa))$. Additionally, we prove that it is inherent that "heavy" parties necessarily have to send to many parties, and for this particular type of protocol, it is necessary to send to at least $\log(n)$ neighbors.

As a final contribution, we do probabilistic simulations of our protocol under various adversarial strategies and weight distributions. Our simulations confirm that our protocol's parameters can be instantiated within the practical range for all such configurations.

Closely related work. The protocol presented in this work achieves an asymptotic efficiency similar to [MNT22b]. The main difference between the work is that this work does not rely on a constant fraction of the parties being honest but instead on a constant fraction of weight being honest. This work is, however, only proven secure against a static adversary. We conjecture that the techniques from [MNT22b] can be applied to achieve such delayed adaptive security, but due to these techniques' technical overhead, this has not been done.

Concurrent with and independent of this work, Coretti, Kiayias, Moore, and Russell [Cor+22] presented a message dissemination protocol tailored for the Ouroboros Praos protocol [Dav+18], which relies on a constant fraction of the stake to behave honestly. There is an overlap of ideas between their protocol and ours as they also choose several connections weighted by the amount of stake each party has. However, instead of ensuring the delivery of messages to all parties, they allow a small fraction of the stake to be eclipsed and not receive updates. Because of this, they do custom proofs for the security of the Ouroboros Praos protocol when deployed together with their protocol.

Differences to the original work. Apart from minor differences (such as rephrasings and correction of typos) between this dissertation's chapter and the full version of the publication [Liu+22a] is that the appendix in [Liu+22a] is blended into the main body and now refers directly to a lemma in Chapter 3 instead of [MNT22b].

1.2.4 Chapter 5 Asymptotically Optimal Message Dissemination with Applications to Blockchains

This chapter is based almost verbatim on [LMT22]. This is a full version of an article that, at the time of writing, is submitted for peer review.

> *[LMT22]:* Chen-Da Liu-Zhang, Christian Matt, and Søren Eller Thomsen. *Asymptotically Optimal Message Dissemination with Applications to Blockchains.* Cryptology ePrint Archive, Paper 2022/1723. 2022. URL: https://eprint.iacr.org/2022/1723.

Below I will explain the motivation for the project, provide a high-level summary of its contributions, and place it in its immediate academic context.

Motivation. Because Blockchains protocols rely on multicast networks, any improvement in the delivery latency of a multicast network will directly affect the throughput of a blockchain built on top. For parties with limited bandwidth, the delivery latency in a multicast protocol is not only affected by the number of neighbors and the distance to the party furthest away but also by the actual number of bits that needs to be transmitted. Chapters 3 and 4 present secure multicast protocols that send a total of $\Theta(l \cdot n \cdot (\kappa + \log(n)))$ when a message of length l is disseminated among n parties with a security parameter κ. Compared to an optimal message dissemination protocol where each party only receives a message once, this means there is an overhead of $\Theta(l \cdot (\kappa + \log(n))$ for each party. We seek to reduce this overhead.

Contributions. We present an asymptotically optimal multicast protocol. For n parties with security parameter κ, and a message of length $l = \Omega((\log(n) + \kappa) \cdot (\log(\log(n) + \kappa)))$, each party only needs to communicate transmit just $\Theta(l)$ bits. Still, the number of parties each party needs to communicate with is only $\Theta(\kappa + \log(n))$ (similar to previous work). We achieve this (optimal) per-party communication and communication complexity by using erasure-correcting codes to lower the variance of previous multicast protocols. Using erasure-correcting codes opens the possibility for an adversary to try to prevent honest parties from reconstructing the original message. We prevent this by using a cryptographic accumulator for each message.

We do probabilistic simulations of our protocol that show that this is not only a theoretical improvement over previous protocols but our protocol can cut the total communication in half compared to the protocol presented in Chapter 4. Additionally, we show how the techniques from Chapter 4 can be leveraged to, in a black-box fashion, transform any multicast protocol secure assuming any constant fraction of parties honest to a protocol secure assuming a constant fraction of the public weights. Using this transformation, our protocol also applies to the blockchain setting.

Closely related work. Doerr and Fouz [DF11] also seeks to optimize the communication complexity of "rumor spreading" and achieves asymptotic optimality within their model, which is slightly different than ours as it proceeds in predefined rounds instead of at the speed of the underlying network as ours. Contrary to our protocol, the correctness of their protocol crucially relies on correct feedback from parties on whether or not they already received the message, which makes it inapplicable to a setting with byzantine faults.

Rohrer and Tschorsch [RT19] uses similar techniques to ours to design a multicast network for blockchains. In particular, they also rely on error-correcting codes to increase the robustness of the network. However, contrary to our protocol, their protocol relies on disseminating the message over a tree structure with less redundancy, which makes it more vulnerable to adversarial behavior. Additionally, are no proof of the security of their protocol provided.

Differences to the original work. Apart from minor differences (such as rephrasings and correction of typos) between this dissertation's chapter and [LMT22] are:

1. The flooding skeleton for a flooding protocol is not recapped since it appears in Chapter 4.

2. A brief outline of the chapter has been added.

3. A section that concluded on the work has been removed as the contributions have already been summarized earlier.

1.2.5 Other Contributions

Throughout my PhD, I co-authored the following publications/manuscripts that are not a part of this dissertation.

[GTA19]: Simon Oddershede Gregersen, Søren Eller Thomsen, and Aslan Askarov. "A Dependently Typed Library for Static Information-Flow Control in Idris". In: *POST*. vol. 11426. Lecture Notes in Computer Science. Springer, 2019, pp. 51–75.

[TS21]: Søren Eller Thomsen and Bas Spitters. "Formalizing Nakamoto-Style Proof of Stake". In: *Computer Security Foundations*. IEEE, 2021, pp. 1–15.

[Kam+22]: Simon Holmgaard Kamp, Jesper Buus Nielsen, Søren Eller Thomsen, and Daniel Tschudi. *Enig: Player Replaceable Finality Layers with Optimal Validity*. Cryptology ePrint Archive, Paper 2022/201. 2022. URL: https://eprint.iacr.org/2022/201.

1.3 General Preliminaries

The individual chapters of the thesis will introduce specific notation used within the respective chapters. However, this section introduces some basic notation and bounds used in several parts of the thesis.

Notation. We use the infix notation "$:=$" for assigning a variable a (new) value, the infix notation "$\triangleq$" to emphasize that a concept is being defined formally for the first time, the infix notation "$==$" to denote an equality test returning a boolean value, and the infix notation "$::$" to denote list-extension. The set of natural numbers is denoted by $\mathbb{N} \triangleq \{0, 1, 2, \ldots\}$, the set of real numbers is represented by $\mathbb{R}$, and the set of non-negative real numbers is denoted $\mathbb{R}_{\geq 0}$. We denote the probability of an event E by $\Pr[E]$ and the expected value of a random variable X by $\mathbb{E}[X]$. We will write $A \xleftarrow{\$} \mathbb{D}$ to sample the value A from the distribution $\mathbb{D}$ and use the infix notation $\sim$ to denote that two random variables are distributed identically. We will let $\mathcal{B}(n, \rho)$ denote the binomial distribution with parameters n and ρ, and $\mathcal{U}(A)$ denote the uniform distribution on a set A. We denote by $\log x$ the natural logarithm of x. In our proofs, we will use the acronyms LHS and RHS to refer to the left-hand and right-hand sides of an (in)equality.

Negligible and overwhelming functions. When referring to a probability as being negligible in a specific parameter (often the security parameter), we use the standard mathematical definition of negligibility. That is, a function $\mathtt{f}(\kappa)$ is negligible in κ if for any $d \in \mathbb{N}$ there exists an κ_0 s.t. for all $\kappa \geq \kappa_0$ it holds that $\mathtt{f}(\kappa) \leq \kappa^{-d}$. If an expression contains free variables (such as the number of parties in a protocol), we require the above to hold independently of these. A function $\mathtt{f}(\kappa)$ is overwhelming in the κ if $1 - \mathtt{f}(\kappa)$ is negligible in κ.

Tail bounds. We record the Chernoff bound, which states that the sum of independent random variables concentrates around their mean. This bound will be used extensively throughout the remainder of the dissertation.

Lemma 1.3.1 (Chernoff bound)**.** *Let $X_1, \ldots, X_n$ be independent random variables with $X_i \in \{0,1\}$ for all i, and let $\mu \coloneqq \mathbb{E}\big[\sum_{i=1}^n X_i\big]$. We then have for all $\delta \in [0,1]$,*

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

Part I

Publications and Manuscript

2 Weight-Based Nakamoto-Style Blockchains

2.1 Introduction

In classical blockchains such as Nakamoto's Bitcoin [Nak08], the parties run a distributed "lottery" to decide who can append the next block to the existing chain. When there is a lottery winner, a block is produced and disseminated to the other parties, which will perform a series of checks to guarantee that the block is valid and that the party that created the block actually won the lottery. If all the checks are correct, the parties append the new block to their local view of the chain. Classical blockchains (also called Nakamoto-style, or NSB for short) usually assume the majority of the resources (e.g., computational power or stake) to be trusted, from which they can achieve a totally ordered broadcast.

Bitcoin is an NSB based on proof-of-work (PoW) where a block is only considered valid and allowed to be appended to the chain if its hash value is below some threshold value T. The probability of this is proportional to T. The value T is computed in real-time by the network such that a single valid block is created, on average, every 10 minutes. In a period where T is fixed[1] the "best-chain" rule for Bitcoin is determined by how many blocks are on the chain. Previous analyses of the Bitcoin protocol [GKL15; GKL17; Niu+19; PSS17; Ren19] show that under certain network assumptions, Bitcoin satisfies the properties of chain growth, chain quality and common prefix (introduced by [GKL15]) for some choice of parameters.

The *block time* of an NSB is the average time between blocks. Existing analyses use at their core the fact that the block time is longer than the average network delay. This allows for honest block winners to typically have seen all previous honest blocks when they add a new block. This allows the longest chain to grow by one block when there is an honest winner. On the other hand, if blocks are produced faster than they propagate, then all "bets are off". Therefore the block time of existing NSB needs to be set conservatively to some worst-case value. At a conceptual level, our study is motivated by the simple observation that on existing NSBs, whenever the block time is fixed to a constant, the protocols do not respond with higher throughput when the network is, in fact, much faster than the worst case assumed.

At a technical level, our study departs from the observation that not all types of blocks are equal. For example, in Bitcoin, there are two types of blocks, those above the threshold T, which do not count at all, and those below T, which count as one block. However, blocks with a hash below T/m for some integer m have an average block time about m times as long as blocks with a hash below T. Therefore, one could, for instance,

[1] For simplicity, in this work, we only consider the case of fixed participation. We leave the case of adaptive T as future work.

consider counting blocks with a hash below T/m with "weight" m or "weight" 2^m. That is, we can consider different *weight functions* assigning weights to blocks based on their hash values. This raises the following question:

> *Can we get better guarantees for NSBs if we assign different weights to the blocks?*

In that vein, we provide a general framework to analyze PoW protocols under different weight functions. The main goal of the framework is to provide useful tools where one can easily explore and analyze the impact of varying weight functions applied to a Bitcoin-like protocol. As a sanity check, we first instantiate the (standard) Bitcoin weight function in our framework (Section 2.4.2.1) and show similar bounds as previous work.

As evidence of the usefulness of our framework in exploring different weight functions, we show that a large class of weight functions achieves a weak form of "optimistic responsiveness" (c.f. [PS18b]). In a nutshell, we show that in periods without corruption, the time it takes for blocks to be in a common prefix only depends on the actual network delay instead of a known upper bound.

2.1.1 Overview of Our Results

Our contributions are twofold: (1) We provide a general framework for easy exploration and design of protocols with different weight functions, and (2) we show that there are weight functions that are strictly better than the traditional longest chain rule of Bitcoin. We detail our contributions next:

Generic framework. Our framework constitutes the backbone of a PoW blockchain where its valid block predicate and best-chain rule rely on a weight function that establishes a numerical value (i.e., weight) to each individual block in the chain. The best chain at any given time is the chain with more accumulated weight over all its blocks. We provide general lemmas for several bounds on the produced weight of a PoW protocol instantiated with any weight function. Furthermore, we derive for any weight function the concrete bounds needed for the main blockchain properties of growth, quality, and common prefix to be guaranteed and calculate how these bounds translate into guarantees for the protocol. The main goal of our generic framework is that any weight function can be "plugged" into the framework, and the parameters needed for the desired levels of guarantees can be obtained almost directly. This enables an easy exploration and design of protocols without needing to redo a series of complex and potentially error-prone proofs.

Weakly optimistically responsive protocol. We introduce in Section 2.4 the class of T-capped weight functions, which are monotonically increasing weight functions that are constant if the input is larger than a threshold T. We show that a PoW blockchain that employs a particular weight function from such a class achieves chain growth, chain quality, and common prefix parameters similar to the ones achieved by Bitcoin in previous works [GKL15; GKL17]. We also note that instantiating a PoW protocol with a particular T-capped weight function can make it *weakly* optimistically responsive, i.e., under no corruption, we show common prefix guarantees for the protocol that are based on the *actual* network delay, and not on the known upper bound.

Intuitively, a weight function needs to satisfy two properties: First, blocks produced at a suitable frequency with respect to the actual network delay should get enough weight to cancel out the weight of blocks produced too fast. Secondly, it should be difficult for the adversary to create extremely heavy blocks, as these can be used to cause massive rollbacks and violate common prefix. To satisfy both conditions, we let the weight functions grow exponentially until they reach a threshold determined by the known upper bound $\hat{\Delta}_{\mathsf{Net}}$ on the network delay; above the threshold, the weight remains constant. The cap ensures the adversary cannot cause rollbacks longer than this upper bound with a single block. Growing exponentially below the threshold gives us responsiveness in the all-honest setting: Assume the actual network delay Δ_{NET} is much lower than the known upper bound $\hat{\Delta}_{\mathsf{Net}}$. Blocks produced at the right frequency with respect to Δ_{NET} are weighted much heavier than more frequent blocks. Thus, the honest parties essentially build a chain just with these blocks, and the lighter ones are negligible in comparison. It is not necessary to wait for even heavier blocks up to the threshold to get the desired properties. Note that this only provides responsiveness if there are no corrupted parties: A single dishonest party can, with non-negligible probability, produce a block with maximal possible weight and thus cause a rollback of honest blocks produced in $\hat{\Delta}_{\mathsf{Net}}$ time.

While this may seem not particularly useful, the responsiveness can still significantly improve the throughput of the chain when the protocol is combined with a finality layer such as Casper the Friendly Finality Gadget [BG17], GRANDPA [SK20], or Afgjort [Din+20], where blocks are declared as final (and cannot be rolled back) as soon as they are in the common-prefix of honest users. In that case, the time it takes for blocks to be in the common prefix in periods without corruption only depends on the actual network delay, and finalization ensures that all users know which blocks to trust. We leave it as interesting future work to analyze the feasibility of responsiveness in the face of active corruption.

Outline of the chapter. In the remainder of this section, we will review some closely related work. In Section 2.2, we will describe our model, introduce our protocol and notation for it, and provide bounds on the amount of produced weight. Next, in Section 2.2, we provide general lemmas that show for weight functions satisfying certain bounds; our protocol has both chain growth, chain quality, and the common prefix property. Finally, in Section 2.4, we instantiate our framework with the class of capped weight functions and show how Bitcoin is an instance of this class.

2.1.2 Related Work

The first formal analysis of NSB blockchains was given in the seminal paper [GKL15] for a fixed threshold T, which was later extended to a variable threshold in [GKL17], and to a different setting with more variable message delivery times, adaptive corruption, and spawning of new players in [PSS17]. Ren [Ren19] gives a more straightforward analysis of the standard Bitcoin protocol under the assumption that mining on Bitcoin can be modeled as a Poisson process.

Responsiveness was defined by Pass and Shi [PS17b] as the property of a blockchain that achieves a liveness parameter expressed in terms of the actual network delay, independent of the conservative upper bound on the network delay used to instantiate the protocol. They show that a protocol tolerating up to a $\frac{1}{3}$ corruption can achieve responsiveness and that this bound is tight. They later show in [PS18b] that assuming only an honest majority (and a delay for the corruption of parties), it is possible to obtain the weaker property of *optimistic responsiveness*, i.e., responsiveness under some additional “goodness” condition, while still providing security in the worst case. In particular, they show responsiveness in the case of more than $\frac{3}{4}$ honest computing power and an additional assumption of an honest *accelerator*. In [Shr+20], a lower bound is given for the latency in the *optimistic* setting of [PS18b] alongside a protocol achieving this within a constant factor of the actual network delay.

Since [PS18b] and [Shr+20] both require a committee and an accelerator, their results only hold, assuming considerably delayed corruption, allowing the accelerator to make progress. On the other hand, our generic weighted protocol can tolerate immediate adaptive corruptions, as desired in the permissionless setting. However, our result is weaker with respect to the “goodness” condition since we only achieve responsiveness in the case of no corruption. Whether one can get responsiveness with non-zero fully adaptive corruption in the permissionless setting remains an open problem.

The concept of assigning different weights to blocks based on their hash value has already been considered in the context of proofs of proof of work [KLS16; KMZ17]. However, the purpose there, and consequently the analysis, was completely different: Heavy blocks are used to link to older blocks in addition to the direct parents to allow for faster verification of recent transactions without verifying the whole chain.

2.2 Our Generic Framework for Weight-Based Analysis

In this section, we formally describe our generic framework and introduce the concept of weight functions for PoW blockchains. In Sections 2.2.3 and 2.2.5, we provide generic definitions and tools that we will use to show the properties of chain growth, chain quality, and common prefix for PoW blockchains that leverage weight functions (in Section 2.3). Our analysis builds upon the ideas of previous work [GKL15; Ren19] and extends those to the more general setting of weighted blocks. We start by describing the blockchain model that we consider for our framework.

2.2.1 Blockchain Model

Network and time. We assume that time is divided into *rounds*, which correspond to the smallest unit of time of interest. We consider a network with bounded delay parameterized by an upper bound Δ_{NET} on the network delivery time. It allows parties to multicast messages. That is, any message sent by an honest party in round r is guaranteed to arrive at all honest parties until round $r + \Delta_{\text{NET}}$. As in, e.g., [PSS17], we assume a gossip network, which ensures that all messages (sent by a dishonest sender and) received by an honest party in round r are received by all honest parties until round $r + \Delta_{\text{NET}}$. The adversary can set the actual delay of messages (per message and party) (within Δ_{NET}). The delay Δ_{NET} is *not* known to the honest parties. However, we assume that honest parties know a rough upper bound $\hat{\Delta}_{\mathsf{Net}}$, potentially much larger than Δ_{NET}, on the network delay.

In Chapters 3 to 5, we discuss how such a network can be efficiently and securely realized from underlying point-to-point channels. The results in these chapters are proven assuming only an upper bound on the underlying point-to-point channels, and therefore only upper bounds are derived for the message propagation delay. However, the analysis holds for an upper bound on the point-to-point channels throughout the time it takes to propagate a single message (as the protocols do not depend on timing), and therefore these realizations of multicast networks will be responsive.

Random oracle. Following [PSS17], we assume every "party" can make at most one query to a random oracle in each round. The idea is that one round corresponds to the time it takes to evaluate the hash function on one CPU and is the smallest unit of time of interest. To model real-world parties with different amounts of computing power, one can assume that they control different amounts of these "one-query-per-round" parties. As in [Bad+17; GKL15; PSS17], we allow the corrupted parties to make their queries sequentially, while honest parties have to make the queries in parallel. We assume the range of the random oracle to be $\mathcal{H} := \{1, \ldots, 2^k\}$.

In the remainder of the chapter, we let $q \in \mathbb{N}$ denote the number of parties in the protocol. As each party has one query, this is also the maximum amount of queries that can be made to the oracle in each round.

Corruptions. We allow the adversary to adaptively corrupt up to a $\beta < \frac{1}{2}$ fraction of all parties before each round. Newly corrupted parties are then entirely under the adversary's control from that round on. We denote by $\alpha := 1 - \beta$ the minimal fraction of participating parties that are honest at any time. Note that by our definition of the random oracle, there can be at most $q\beta$ random-oracle queries by corrupted parties in each round, and there are at least $q\alpha$ queries by honest parties in each round (since honest parties in our protocol query the random oracle in each round, c.f. Section 2.2.2). We will thus, for most of the chapter, only consider these upper and lower bounds on the numbers of dishonest and honest queries and not explicitly map these to parties.

2.2.2 Blockchain Protocol

Our protocol is similar to Bitcoin [Nak08], and the following description assumes at least some basic prior knowledge of the Bitcoin protocol. However, we deviate from the original Bitcoin protocol in two crucial aspects; we change the *best chain rule* and the *valid block predicate*. While the valid block predicate is used to decide what blocks should be considered valid, the best chain rule decides where parties need to append new blocks. Our notation follows the one from [GKL15] closely.

Mining. As in Bitcoin, miners in our protocol continuously take what they currently consider the best chain and try to extend it with a new block. The proof of work aspect corresponds to miners finding an input to a hash function with certain properties. In the Bitcoin protocol, a valid block must satisfy (among other things) that its hash is smaller than some threshold T. The challenge of finding a nonce that makes the block hash small enough is what makes Bitcoin a proof-of-work blockchain. Therefore, the threshold T is adjusted such that the block-production rate is approximately constant. The constant is chosen as a trade-off between performance and security. The block validity predicate of Bitcoin thus consists of checking the block hash along with some (for our purposes unimportant) syntactic well-formedness conditions on the block and its contents. In our protocol, blocks are considered valid independent of their hash value. Instead, the hash of a block determines how much the block weighs when selecting the best chain. To avoid having many low-weight blocks swarm the network, we can use a *cutoff*. However, since it does not impact the protocol's security but is merely a parameter that can be optimized for throughput, we will ignore it in this work.

We define the round in which a block was mined to be the round in which the corresponding query to the random oracle was made.

Best chain. In Bitcoin (with fixed difficulty), the length of the chain is what decides how "good" a chain is [GKL15; Nak08]. Thus, in Bitcoin, chains with more blocks are considered better.[2] In our protocol, we use a best chain rule based on the accumulated weight of the blocks in a chain, i.e., the heavier a chain is, the better, as in bitcoin with variable difficulty [GKL17].

No insertions, copies, and predictions. To simplify our analysis and following [GKL15], we assume throughout the chapter that it never happens that a new block is added between two existing blocks (*insertion*), the same block occurs in two different positions (*copy*), or a block extends a block that is mined in a later round (*prediction*). As shown in [GKL15], insertions and copies can only occur if there is a collision in the random oracle linking blocks together, which has a negligible probability, and the probability of guessing a block is negligible as well.

2.2.3 Basic Definitions

In this section, we first present some basic definitions for the weight of a chain and the weight of a block. Then we present a categorization for certain good events which are essential for the analysis, and finally, we introduce the notation for upper and lower bounds on the weight produced.

2.2.3.1 Weight

We define the chain of a block B denoted $\mathsf{Chain}(B)$ to be the list of all blocks one gets by following the pointers in the chain from B up to the genesis block. We next define the concept of weight for blocks and chains.

Definition 2.2.1 (Weight functions, weight of blocks and chains)**.** We define a *weight function* as a function of type $\mathcal{H} \to \mathbb{R}_{\geq 0}$. Let w be a weight function. We then define the weight of a block B to be $\mathsf{Weight}_{\mathsf{w}}(B) \triangleq \mathsf{w}(\mathtt{Hash}(B))$, and the weight of a chain C to be $\mathsf{Weight}_{\mathsf{w}}(C) \triangleq \sum_{B \in C} \mathsf{Weight}_{\mathsf{w}}(B)$.

Next, we define the weight range, which is analogous to the depth of a block in Bitcoin.

[2]The actual best-chain rule is augmented with (for security, an insignificant) tie-breaking rule.

Definition 2.2.2 (Weight range)**.** Given a weight function w, we define the *start weight* of a block B to be

$$\mathsf{StartWeight}_{\mathsf{w}}(B) \triangleq \mathsf{Weight}_{\mathsf{w}}(\mathsf{Chain}(B)) - \mathsf{Weight}_{\mathsf{w}}(B)$$

and the *end weight* to be

$$\mathsf{EndWeight}_{\mathsf{w}}(B) \triangleq \mathsf{Weight}_{\mathsf{w}}(\mathsf{Chain}(B)).$$

We also define the weight range of a block B to be

$$\mathsf{WeightRange}_{\mathsf{w}}(B) \triangleq (\mathsf{StartWeight}_{\mathsf{w}}(B), \mathsf{EndWeight}_{\mathsf{w}}(B)].$$

Consequently,

$$|\mathsf{WeightRange}_{\mathsf{w}}(B)| = \mathsf{Weight}_{\mathsf{w}}(B).$$

2.2.3.2 Good Events

Previous analyses [GKL15; Niu+19; PSS17; Ren19] are based on the fact that in a certain amount of rounds, a block is produced that has enough time to propagate to all honest parties before a new block is mined. Ren [Ren19] takes a slightly different approach and defines this in terms of blocks rather than rounds. More concretely, he defines a "non-tailgater" as an honest block mined at time t such that no other honest block is mined between time $t - \Delta_{\text{NET}}$ and t. We believe this is closer to the intuition for the proof, namely that once in a while, an honest party mines a block that has enough time to propagate. In his analysis, mining is assumed to be a Poisson process; therefore, no mining events occur simultaneously with positive probability. In our model, however, several blocks can be mined in the same round. If several blocks are mined in a round after Δ_{NET} empty rounds, we can count one of them as a "good" block.

To leverage this in the analysis, we introduce an order in the mined blocks that we call "proof-order". With the order fixed, one can choose, e.g., the first of these blocks as the "good" block.[3] More formally, we introduce an arbitrary but fixed total order on all blocks produced in the protocol. We order blocks lexicographically first based on the production round (i.e., the round the block was created) and secondly on the party that made the query to the random oracle. Note that the production time of a block is well-defined, even for adversarial blocks, as they also need to query the random oracle in some rounds. We stress that this enumeration and induced order of blocks is completely unrelated to the total order of blocks that the protocol achieves and is only needed as an artifact of our proofs. We will refer to the above as the *proof-order* to avoid confusion.

We now use this order on blocks to precisely categorize certain "good" events (blocks mined with sufficient time between them). We further generalize previous notions to our setting with different weights, i.e., instead of requiring that no blocks are mined within a propagation period, we only need that no blocks above a certain threshold are mined within this period.

[3]The proof-order could be defined to take the block with maximal weight in each round instead of ordering them by the parties. This would give a slightly tighter analysis as there would be slightly more "good" weight. For simplicity, we have chosen not to take this approach.

Definition 2.2.3 (h-(left-)isolation)**.** Let $h \in \mathcal{H}$, and let B be a block mined in round $r \in \mathbb{N}$. We say B is *h-left-isolated* if B is honest, $\texttt{Hash}(B) > h$, and there is no block left of B in the proof-order with hash above h mined in rounds $[r - \Delta_{\text{NET}}, r]$. If B is honest, $\texttt{Hash}(B) > h$, and no other blocks with hash above h are mined in rounds $[r - \Delta_{\text{NET}}, r + \Delta_{\text{NET}}]$, we say B is *h-isolated.*

Note that we define h-(left-)isolation with respect to the unknown upper bound Δ_{NET} on the network delay, not on the known bound $\hat{\Delta}_{\mathsf{Net}}$.

Remark 2.2.1. Similar notions have been defined in previous work [GKL15; Niu+19; PSS17; Ren19]. We deviate from these definitions by defining (resp. left-) isolation to require that *no* blocks are mined on either side (resp. to the left) of a block, whereas earlier work had the requirement that no other *honest* block was mined within that period. We use the stricter definition because it simplifies some of the arguments (especially with respect to adaptive corruptions). Only considering honest blocks may potentially allow proving tighter bounds, though. Note that we define the round in which a block was mined to be the round in which the corresponding query to the random oracle was made, so this is also well-defined for corrupted parties, who may send their block in a later round.

Left-isolated blocks are called "non-tailgaters" and isolated blocks are called "loners" by Ren [Ren19]. Analogous notions to that of a round with a left-isolated block have in previous work been called an "effective-round" [Niu+19] and "isolated successful round" [GKL15]. The event of an isolated block has in previous work been called "convergence opportunity" [PSS17], "uniquely effective round" [Niu+19] and an "uniquely isolated successful round" [GKL15]. We chose the terms "left-isolated" and "isolated" as we believe them to be more intuitive.

2.2.4 Bounds on Produced Weight

We now introduce definitions for weight functions describing different bounds on the weight that can be produced with a specific weight function. We start with the upper bounds on how much weight a certain number of queries can produce. We will later use this fact to reason about how much weight any adversary can produce.

We say a weight function is $(\hat{W}_g, \hat{p}_g)$-upper-bounding for some parameter $g \leq q$ if the weight of all blocks mined in r rounds (for all $r \in \mathbb{N}$) with at most g queries (honest or dishonest) per round is at most $\hat{W}_g(r)$, except with probability $\hat{p}_g(r)$. Similarly, we introduce $(\hat{W}_g^{\leq h_0}, \hat{p}_g^{\leq h_0})$-below-threshold-upper-bounding to bound the weight produced by blocks with hash value at most h_0, and $(\hat{W}_g^{>h_0}, \hat{p}_g^{>h_0})$-above-threshold-upper-bounding to bound the weight produced by blocks with hash value more than h_0.

Definition 2.2.4. Let w be a weight function, let $g \in \mathbb{N}$, $h_0 \in \mathcal{H}$, let $\hat{W}_g$,$\hat{W}_g^{\leq h_0}$, $\hat{W}_g^{>h_0} \colon \mathbb{N} \to \mathbb{R}$, and let $\hat{p}_g, \hat{p}_g^{\leq h_0}, \hat{p}_g^{>h_0} \colon \mathbb{N} \to [0, 1]$ be monotonically decreasing. Further, let $W_{g,r}$ for $r \in \mathbb{N}$ be the random variable corresponding to the total weight of all blocks weighted with w mined in r consecutive rounds with at most g queries in each round, and similarly, $W_{g,r}^{\leq h_0}$ ($W_{g,r}^{>h_0}$) for $r \in \mathbb{N}$ be the random variable corresponding to the total weight of all blocks with hash value at most h_0 (more than h_0) weighted with w mined in r consecutive rounds with at most g queries in each round. We say w is

$\big(\hat{W}_g, \hat{p}_g\big)$*-upper-bounding* if for all $r \in \mathbb{N}$,

$$\Pr\big[W_{g,r} \geq \hat{W}_g(r)\big] \leq \hat{p}_g(r),$$

w is $\big(\hat{W}_g^{\leq h_0}, \hat{p}_g^{\leq h_0}\big)$*-below-threshold-upper-bounding* if for all $r \in \mathbb{N}$,

$$\Pr\Big[W_{g,r}^{\leq h_0} \geq \hat{W}_g^{\leq h_0}(r)\Big] \leq \hat{p}_g^{\leq h_0}(r),$$

and w is $\big(\hat{W}_g^{>h_0}, \hat{p}_g^{>h_0}\big)$*-above-threshold-upper-bounding* if for all $r \in \mathbb{N}$,

$$\Pr\Big[W_{g,r}^{>h_0} \geq \hat{W}_g^{>h_0}(r)\Big] \leq \hat{p}_g^{>h_0}(r).$$

Next, we introduce the definition for lower bounds on the amount of (left-) isolated weight, i.e., on how much weight is produced by honest parties with sufficient time in between. By our definition of (left-)isolated blocks, only honest blocks can be left-isolated. We, therefore, do not use a parameter g here but always consider q queries in each round in total, with at least $q\alpha$ queries from honest parties. We introduce the notion of a $\Big(\check{W}_{\mathsf{Iso}^h}, \check{p}_{\mathsf{Iso}^h}\Big)$-isolated-lower-bounding weight function. It means that the total weight of all h-isolated blocks mined in r consecutive rounds is at least $\check{W}_{\mathsf{Iso}^h}(r)$, except with probability $\check{p}_{\mathsf{Iso}^h}(r)$. Left-isolated-lower-bounding weight functions are defined analogously.

Definition 2.2.5. Let w be a weight function, and let $h_0 \in \mathcal{H}$, $\check{W}_{\mathsf{Iso}^{h_0}}, \check{W}_{\mathsf{LeftIso}^{h_0}} : \mathbb{N} \to \mathbb{R}$, and let $\check{p}_{\mathsf{Iso}^{h_0}}, \check{p}_{\mathsf{LeftIso}^{h_0}} : \mathbb{N} \to [0, 1]$ be monotonically decreasing. Further let $W_{r,\mathsf{Iso}^{h_0}}$ for $r \in \mathbb{N}$ be the random variable corresponding to the total weight of all h-isolated blocks weighted with w mined in r consecutive rounds, and let $W_{r,\mathsf{LeftIso}^{h_0}}$ for $r \in \mathbb{N}$ be the random variable corresponding to the total weight of all h-left-isolated blocks weighted with w mined in r consecutive rounds. We say w is $\Big(\check{W}_{\mathsf{Iso}^{h_0}}, \check{p}_{\mathsf{Iso}^{h_0}}\Big)$*-isolated-lower-bounding* if for all $r \in \mathbb{N}$,

$$\Pr\Big[W_{r,\mathsf{Iso}^{h_0}} \leq \check{W}_{\mathsf{Iso}^{h_0}}(r)\Big] \leq \check{p}_{\mathsf{Iso}^{h_0}}(r),$$

and w is $\Big(\check{W}_{\mathsf{LeftIso}^{h_0}}, \check{p}_{\mathsf{LeftIso}^{h_0}}\Big)$*-left-isolated-lower-bounding* if for all $r \in \mathbb{N}$,

$$\Pr\Big[W_{r,\mathsf{LeftIso}^{h_0}} \leq \check{W}_{\mathsf{LeftIso}^{h_0}}(r)\Big] \leq \check{p}_{\mathsf{LeftIso}^{h_0}}(r).$$

2.2.5 Proving Bounds from Properties of the Weight Functions

In this section, we show how to derive some of the thresholds defined in Section 2.2.4.

Notation. In the remainder of the chapter we define $p_{\leq h_0} := \frac{h_0}{2^k}$ to be the probability that a single random oracle query returns a value at most h_0, and $\mathsf{w}_{\max\leq h_0} := \max_{h\in\{1,\ldots,h_0\}} \mathsf{w}(h)$, $\mathsf{w}_{\max>h_0} := \max_{h\in\{h_0+1,\ldots,2^k\}} \mathsf{w}(h)$, and $\mathsf{w}_{\min>h_0} := \min_{h\in\{h_0+1,\ldots,2^k\}} \mathsf{w}(h)$ (for the weight function that is clear from the context).

First, we provide a simple upper-bound for the total weight above and below a threshold.

Lemma 2.2.6 (Weight above and below a threshold)**.** *Let* w *be a weight function, let* $g \in \mathbb{N}$*, and* $h_0 \in \mathcal{H}$*. Then, for all* $\delta \in (0,1)$*,* w *is*

(i) $\left(\hat{W}_g^{\leq h_0}, \hat{p}_g^{\leq h_0}\right)$*-below-threshold-upper-bounding with*

$$\hat{W}_g^{\leq h_0} = \mathsf{w}_{\max \leq h_0} \cdot (1+\delta) \cdot g \cdot r \cdot p_{\leq h_0}, \qquad \hat{p}_g^{\leq h_0} = e^{-\frac{\delta^2 \cdot g \cdot r \cdot p_{\leq h_0}}{3}},$$

(ii) and $\left(\hat{W}_g^{> h_0}, \hat{p}_g^{> h_0}\right)$*-above-threshold-upper-bounding with*

$$\hat{W}_g^{>h}(r) = \mathsf{w}_{\max > h_0} \cdot (1+\delta) \cdot g \cdot r \cdot (1 - p_{\leq h_0}), \quad \hat{p}_g^{>h}(r) = e^{-\frac{\delta^2 \cdot g \cdot r \cdot (1 - p_{\leq h_0})}{3}}.$$

Proof. The probability to get a block below a threshold in just one query is $p_{\leq h_0}$ and above a threshold is $1 - p_{\leq h_0}$. The amount of blocks below/above a threshold can be upper bounded with Chernoff (Lemma 1.3.1). Each block below contributes with weight at most $\mathsf{w}_{\max \leq h_0}$, and blocks above with weight at most $\mathsf{w}_{\max > h_0}$. □

We next prove bounds on the number of (left-)isolated blocks and afterward use this for a simple bound on the amount of (left-)isolated weight. The proof follows some ideas from Ren [Ren19]. At a very high level, we proceed by first applying the Chernoff bound to obtain a bound on the number of blocks with hash above h_0, and then using Chernoff again to bound how many of these blocks are (left-)isolated. The main difficulty lies in proving the independence of the involved variables as needed for the Chernoff bound.

Lemma 2.2.7 (Amount of (left-)isolated blocks)**.** *Let* r *be a number of consecutive rounds, let* $h_0 \in \mathcal{H}$*, let* $N_{r,\mathsf{LeftIso}^{h_0}}$ *denote the number of* h_0*-left-isolated blocks produced, and let* $N_{r,\mathsf{Iso}^{h_0}}$ *denote the number of* h_0*-isolated blocks produced during these* r *rounds. We then have for any* $\delta \in (0,1)$*,*

$$\Pr\left[N_{r,\mathsf{LeftIso}^{h_0}} \leq (1-\delta) \cdot \alpha q r \cdot (1 - p_{\leq h_0}) \cdot p_{\leq h_0}^{q \Delta_{\text{NET}}}\right] \leq 2e^{-\frac{\delta^2 \cdot \alpha q r \cdot \left(1 - p_{\leq h_0}\right) \cdot p_{\leq h_0}^{q\Delta_{\text{NET}}}}{16}}, \tag{2.1}$$

$$\Pr\left[N_{r,\mathsf{Iso}^{h_0}} \leq (1-\delta) \cdot \alpha q r \cdot (1 - p_{\leq h_0}) \cdot p_{\leq h_0}^{2 \cdot q \Delta_{\text{NET}}}\right] \leq 3e^{-\frac{\delta^2 \cdot \alpha q r \cdot \left(1 - p_{\leq h_0}\right) \cdot p_{\leq h_0}^{2q\Delta_{\text{NET}}}}{108}}. \tag{2.2}$$

Proof. To prove the lemma, we start by lower-bounding the amount of left-isolated blocks within any sequence of consecutive honest blocks. For any n, we look at the first (according to the proof-order) n honest blocks with a hash above h_0 produced since the start of the r considered rounds. The probability that block i is left-isolated is given by the probability that all of the blocks in Δ_{NET} time before and to the left (with respect to the proof-order) of the block in the same round do not result in a winning event with hardness above h_0. In the worst case, the considered block is the last one in its round, i.e., there are $q - 1$ to the left of block i in that round. Hence, there are at most $q \cdot (\Delta_{\text{NET}} - 1) + (q - 1)$ queries to be considered. Note that if the corrupted parties make fewer queries, this can only increase the probability of left-isolated blocks. The probability that block i is left-isolated is thus at least the probability that all these queries result in a hash value at most h_0. We define $Y_i = 1$ if the ith honest block is h_0-left-isolated. Then,

$$\Pr[Y_i = 1] \geq p_{\leq h_0}^{q \cdot (\Delta_{\text{NET}} - 1) + (q-1)} \geq p_{\leq h_0}^{q \Delta_{\text{NET}}}. \tag{2.3}$$

We further define $N_{\mathsf{LeftIso}^{h_0}}(n) := \sum_{i=1}^{n} Y_i$, i.e., the number of left isolated blocks of the n honest blocks above h_0. The above implies

$$\mathbb{E}\left[N_{\mathsf{LeftIso}^{h_0}}(n)\right] \geq n \cdot p_{\leq h_0}^{q\Delta_{\mathrm{NET}}}. \tag{2.4}$$

Note that $Y_i = 1$ if and only if the inter-arrival time between the $(i-1)$th and the ith honest block with hash above h_0 is at least $q \cdot (\Delta_{\mathrm{NET}} - 1) + (q-1)$.[4] Since the inter-arrival times of independent Bernoulli trials are independent, the Y_i are also independent. We can therefore use the Chernoff bound (Lemma 1.3.1) for $\delta_1 \in (0,1)$ to obtain

$$\Pr\left[N_{\mathsf{LeftIso}^{h_0}}(n) \leq (1-\delta_1) \cdot n \cdot p_{\leq h_0}^{q\Delta_{\mathrm{NET}}}\right] \leq e^{-\frac{\delta_1^2 \cdot n \cdot p_{\leq h_0}^{q\Delta_{\mathrm{NET}}}}{2}}. \tag{2.5}$$

We now bound the number of honest blocks with hash above h_0 produced during the R considered rounds. Let $X_i = 1$ if the i'th honest query results in a hash above h_0. We note that $\Pr[X_i = 1] = 1 - p_{\leq h_0}$. Let $N_{\alpha qr,>h_0} := \sum_{i=1}^{\alpha qr} X_i$ and note that $\mathbb{E}[N_{\alpha qr,>h_0}] \geq \alpha qr \cdot (1 - p_{\leq h_0})$ as αqr is a lower bound on the amount of honest queries. The Chernoff bound (Lemma 1.3.1) for $\delta_2 \in (0,1)$ then implies

$$\Pr\left[N_{\alpha qr,>h_0} \leq (1-\delta_2) \cdot \alpha qr \cdot (1-p_{\leq h_0})\right] \leq e^{-\frac{\delta_2^2 \alpha qr \cdot \left(1-p_{\leq h_0}\right)}{2}}. \tag{2.6}$$

Note that $N_{r,\mathsf{LeftIso}^{h_0}} = N_{\mathsf{LeftIso}^{h_0}}(N_{\alpha qr,>h_0})$. We set $\delta_1 := \delta_2 := \frac{\delta}{2}$. We then have $\delta_1, \delta_2 \in (0,1)$ and $(1-\delta_1)(1-\delta_2) \geq (1-\delta)$. Together with Equations (2.5) and (2.6), and using that $N_{\alpha qr,>h_0} \in \mathbb{N}$, we can conclude that

$$\begin{aligned}
&\Pr\left[N_{r,\mathsf{LeftIso}^{h_0}} \leq (1-\delta) \cdot \alpha qr \cdot (1-p_{\leq h_0}) \cdot p_{\leq h_0}^{q\Delta_{\mathrm{NET}}}\right] \\
&\leq e^{-\frac{\delta^2 \cdot \alpha qr \cdot \left(1-p_{\leq h_0}\right)}{8}} + e^{-\frac{\delta^2 \cdot (1-\delta_2) \cdot \alpha qr \cdot \left(1-p_{\leq h_0}\right) \cdot p_{\leq h_0}^{q\Delta_{\mathrm{NET}}}}{8}} \\
&\leq 2e^{-\frac{\delta^2 \cdot \alpha qr \cdot \left(1-p_{\leq h_0}\right) \cdot p_{\leq h_0}^{q\Delta_{\mathrm{NET}}}}{16}},
\end{aligned} \tag{2.7}$$

where we used $1 - \delta_2 = 1 - \frac{\delta}{2} \geq \frac{1}{2}$ in the last step. This concludes the proof of Equation (2.1).

To prove Equation (2.2), we again first bound how many isolated blocks we get within a sequence of n blocks. As above, we use the proof order to enumerate the first n honest blocks since the start of the R considered rounds with hash above h_0. We define $Z_i = 1$ if the ith block is h_0-isolated, and $Z_i = 0$ otherwise. We note that $Z_i = Y_i \cdot Y_{i+1}$ as $i+1$ is the winning event that happened the shortest time after i, and there are more than Δ_{NET} rounds between these if and only if the latter is left-isolated. Since Y_i and Y_{i+1} are independent, we have

$$\Pr[Z_i = 1] = \Pr[Y_i = 1 \wedge Y_{i+1} = 1] = \Pr[Y_i = 1] \cdot \Pr[Y_{i+1} = 1] \geq p_{\leq h_0}^{2 \cdot q\Delta_{\mathrm{NET}}}. \tag{2.8}$$

Note that Z_i and Z_{i+1} are not independent since they both depend on Y_{i+1}, but Z_i and Z_{i+2} are independent. We therefore write $N_{\mathsf{Iso}^{h_0}}(n) = \sum_{i \in \{1,\ldots,n\} \wedge \mathsf{Odd}(i)} Z_i +$

[4]We are slightly abusing notation since for $i = 1$, the $(i-1)$th block is not part of the n considered blocks, but last the honest block with hash above h_0 before Y_1. Note that if such $(i-1)$th block does not exist in the chain, $Y_i = 1$ with probability 1, and therefore Y_i and the other Y_j are independent.

$\sum_{i\in\{1,\dots,n\}\wedge\mathsf{Even}(i)} Z_i$. Let $N_{\mathsf{Odd}}(n)$ be the number of odd $i \in \{1,\dots,n\}$, and let $N_{\mathsf{Even}}(n)$ be the number of even $i \in \{1,\dots,n\}$. Since $\mathbb{E}\left[\sum_{i\in\{1,\dots,n\}\wedge\mathsf{Odd}(i)} Z_i\right] \geq N_{\mathsf{Odd}} \cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}$, we can apply the Chernoff bound (Lemma 1.3.1) for $\delta_3 \in (0,1)$ to obtain

$$\Pr\left[\sum_{i\in\{1,\dots,n\}\wedge\mathsf{Odd}(i)} Z_i \leq (1-\delta_3)N_{\mathsf{Odd}}(n)\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}\right] \leq e^{-\frac{\delta_3^2\cdot N_{\mathsf{Odd}}(n)\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}}{2}}.$$

We can also apply the Chernoff bound for $\delta_4 \in (0,1)$ to the even case and together with the above obtain

$$\Pr\left[N_{\mathsf{Iso}^{h_0}}(n) \leq ((1-\delta_3)N_{\mathsf{Odd}}(n) + (1-\delta_4)N_{\mathsf{Even}}(n))\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}\right] \leq e^{-\frac{\delta_3^2\cdot N_{\mathsf{Odd}}(n)\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}}{2}} + e^{-\frac{\delta_4^2\cdot N_{\mathsf{Even}}(n)\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}}{2}}. \quad (2.9)$$

Let $\delta_4 = \delta_3$ and note that if n is even then $N_{\mathsf{Odd}}(n) = N_{\mathsf{Even}}(n) = \frac{n}{2}$ and we obtain

$$\Pr\left[N_{\mathsf{Iso}^{h_0}}(n) \leq (1-\delta_3)\cdot n\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}\right] \leq 2e^{-\frac{\delta_3^2\cdot n\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}}{4}}. \quad (2.10)$$

Note that $N_{r,\mathsf{Iso}^{h_0}} = N_{\mathsf{Iso}^{h_0}}(N_{\alpha qr,>h_0})$, and by Equation (2.6), we have $N_{\alpha qr,>h_0} > (1-\delta_2)\cdot\alpha qr\cdot(1-p_{\leq h_0})$ except with small probability. There exists $\delta_2 \in \left(\frac{\delta}{3},\frac{2\delta}{3}\right)$ such that $(1-\delta_2)\cdot\alpha qr\cdot(1-p_{\leq h_0})$ is even if

$$\begin{aligned}
&\left(1-\frac{\delta}{3}\right)\cdot\alpha qr\cdot(1-p_{\leq h_0}) - \left(1-\frac{2\delta}{3}\right)\cdot\alpha qr\cdot(1-p_{\leq h_0}) > 2\\
\iff\quad &\frac{\delta}{3}\cdot\alpha qr\cdot(1-p_{\leq h_0}) > 2\\
\iff\quad &\alpha qr > \frac{6}{\delta\cdot(1-p_{\leq h_0})}. \quad (2.11)
\end{aligned}$$

First assume that Equation (2.11) is satisfied. We then pick $\delta_2 \in \left(\frac{\delta}{3},\frac{2\delta}{3}\right)$ accordingly and $\delta_3 := \delta - \delta_2$. We have

$$(1-\delta_2)\cdot(1-\delta_3) = 1-\delta+\delta_2\delta-\delta_2^2 \geq 1-\delta. \quad (2.12)$$

Note that we further have $\delta_3 \in \left(\frac{\delta}{3},\frac{2\delta}{3}\right)$. Together with Equations (2.6) and (2.10) and using that $1-\delta_2 \geq \frac{1}{3}$ and $\delta_2^2,\delta_3^2 \geq \frac{\delta^2}{9}$, we can conclude that

$$\begin{aligned}
&\Pr\left[N_{r,\mathsf{Iso}^{h_0}} \leq (1-\delta)\cdot\alpha qr\cdot(1-p_{\leq h_0})\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}\right]\\
&\leq e^{-\frac{\delta_2^2\alpha qr\cdot\left(1-p_{\leq h_0}\right)}{2}} + 2e^{-\frac{\delta_3^2\cdot(1-\delta_2)\cdot\alpha qr\cdot\left(1-p_{\leq h_0}\right)\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}}{4}} \quad (2.13)\\
&\leq 3e^{-\frac{\delta^2\cdot\alpha qr\cdot\left(1-p_{\leq h_0}\right)\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}}{108}}.
\end{aligned}$$

We finally consider the case where Equation (2.11) is not satisfied. Then $\alpha qr \leq \frac{6}{\delta\cdot(1-p_{\leq h_0})}$, which implies that

$$3e^{-\frac{\delta^2\cdot\alpha qr\cdot\left(1-p_{\leq h_0}\right)\cdot p_{\leq h_0}^{2\cdot q\Delta_{\text{NET}}}}{108}} \geq \frac{3}{e} \geq 1. \quad (2.14)$$

In this case, Equation (2.2) is therefore trivially satisfied. □

Lemma 2.2.8. *Let* w *be a weight function and* $h_0 \in \mathcal{H}$*. Then, for all* $\delta \in (0, 1)$*,*

(i) w *is* $\left(\check{W}_{\mathsf{LeftIso}^{h_0}}, \check{p}_{\mathsf{LeftIso}^{h_0}}\right)$*-left-isolated-lower-bounding with*

$$\check{W}_{\mathsf{LeftIso}^{h}}(r) = \mathsf{w}_{\min > h_0} \cdot (1-\delta) \cdot \alpha q r \cdot (1 - p_{\leq h_0}) \cdot (p_{\leq h_0})^{q\Delta_{\text{NET}}},$$
$$\check{p}_{\mathsf{LeftIso}^{h}}(r) = 2e^{-\frac{\delta^2 \cdot \alpha q r \cdot (1-p_{\leq h_0}) \cdot (p_{\leq h_0})^{q\Delta_{\text{NET}}}}{16}},$$

(ii) and w *is* $\left(\check{W}_{\mathsf{Iso}^{h_0}}, \check{p}_{\mathsf{Iso}^{h_0}}\right)$*-isolated-lower-bounding with*

$$\check{W}_{\mathsf{Iso}^{h}}(r) = \mathsf{w}_{\min > h_0} \cdot (1-\delta) \cdot \alpha q r \cdot (1 - p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\text{NET}}},$$
$$\check{p}_{\mathsf{Iso}^{h}}(r) = 3 \cdot e^{-\frac{\delta^2 \cdot \alpha q r \cdot (1-p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\text{NET}}}}{108}}.$$

Proof. Each (left-)isolated block contributes at least $\mathsf{w}_{\min > h_0}$ weight. Hence, the bounds on the amount of (left-)isolated blocks from Lemma 2.2.7 directly imply the lower bounds on (left-)isolated weight. □

2.3 Proving Chain Properties

In this section, we prove the standard properties of chain growth, chain quality, and common prefix for our generic framework by only assuming bounds on the produced weight, as introduced in Section 2.2. We consider a fixed weight function w for the entire section, so we leave it out of the notations.

We warm up with some fundamental lemmas that will be used as building blocks when proving the more complex theorems of the chain properties. The following lemma is a generalization of Lemma 5 (i) in [Ren19]. It intuitively says that if we only consider blocks above a specific hash, and enough time has passed since an honest block was mined, then a new honest block will have a different position in the chain than the previous block.

Lemma 2.3.1. *Let* $h \in \mathcal{H}$ *and let* $B \neq B'$ *be* h*-left-isolated blocks. Then,* B *and* B' *have disjoint weight ranges.*

Proof. Without loss of generality, we assume that B is mined first. The party P' who mines B' receives B within Δ_{NET} rounds, which is by definition of h-left-isolation before B' is mined. After receiving B, P' only extends chains with weight at least $\mathsf{EndWeight}(B)$. Hence, $\mathsf{EndWeight}(B) \leq \mathsf{StartWeight}(B')$, and thus, $\mathsf{WeightRange}(B) \cap \mathsf{WeightRange}(B') = \varnothing$. □

The following lemma is a generalization of Lemma 5 (ii) in [Ren19]. The lemma says that if we only consider honest blocks above a specific hash, then if such a block has had enough time to propagate before the next block is produced and no other block was mined in a period before, then this block will not share a position in the chain with any other block.

Lemma 2.3.2. *Let* $h \in \mathcal{H}$ *and let* B *be a* h*-isolated block. Further let* $B' \neq B$ *be an honest block with* $\mathtt{Hash}(B') > h$*. Then,* B *and* B' *have disjoint weight ranges.*

Proof. Let $B_0 \in \{B, B'\}$ be the block that is mined first. By definition of h-isolation, the other block is mined more than Δ_{NET} rounds later. As in the proof of Lemma 2.3.1, we can thus conclude that the party mining the second block knows B_0 beforehand and thus extends a chain with weight at least $\mathsf{EndWeight}(B_0)$. Hence, $\mathsf{WeightRange}(B) \cap \mathsf{WeightRange}(B') = \varnothing$. $\square$

2.3.1 Chain Growth

The chain growth property intuitively says that a chain will increase its weight by at least a fixed bound at every round. We give a formal definition of our weight-based chain growth property next.

Definition 2.3.3 (Chain Growth)**.** Let w be a weight function. The chain growth property with parameters $\rho \in \mathbb{N}$ and $\tau \in \mathbb{R}$, states that for any honest party P that has a chain C_1, it holds that after any ρ consecutive rounds P adopts a chain C_2 such that $\mathsf{Weight}(C_2) \geq \mathsf{Weight}(C_1) + (\rho \cdot \tau)$ for $\tau > 0$.

Next, we show that the accumulated weight of the chain grows at least by the accumulated weight of the left-isolated blocks at each round and therefore satisfies the property of Definition 2.3.3. We show a slightly more general version of chain growth, which is helpful for later proving chain quality.

Theorem 2.3.4 (Chain Growth)**.** *Let C_1 be the best chain of P_1 in round r_1 and let C_2 be the best chain of P_2 in round r_2, where $r_1 \leq r_2 - 2\Delta_{\text{NET}} + 1$. For any $h_0 \in \mathcal{H}$ such that the weight function is $\left(\check{W}_{\mathsf{LeftIso}^{h_0}}, \check{p}_{\mathsf{LeftIso}^{h_0}}\right)$-left-isolated-lower-bounding, we have*

$$\Pr\left[\mathsf{Weight}(C_2) < \mathsf{Weight}(C_1) + \check{W}_{\mathsf{LeftIso}^{h_0}}(r_2 - r_1 - 2\Delta_{\text{NET}} + 1)\right] \\ \leq \check{p}_{\mathsf{LeftIso}^{h_0}}(r_2 - r_1 - 2\Delta_{\text{NET}} + 1).$$

Proof. Let $\mathcal{B}_{\text{li}}^{h_0}$ be the set of all h_0-left-isolated blocks mined in $[r_1 + \Delta_{\text{NET}}, r_2 - \Delta_{\text{NET}}]$. Any block seen by P_1 in round r_1, will be seen by any honest party until round $r_1 + \Delta_{\text{NET}}$. This is specifically true for all blocks in C_1 and thus, $\mathsf{StartWeight}(B) \geq \mathsf{EndWeight}(C_1)$ for all $B \in \mathcal{B}_{\text{li}}^{h_0}$. Moreover, all blocks in $\mathcal{B}_{\text{li}}^{h_0}$ have disjoint weight ranges by Lemma 2.3.1. As all these blocks had enough time to propagate to P_2 in round r_2, P_2 will have at least one chain C_2' with $\mathsf{Weight}(C_2') \geq \mathsf{Weight}(C_1) + \sum_{B \in \mathcal{B}_{\text{li}}^{h_0}} \mathsf{Weight}(B)$. Note that $\mathsf{Weight}(C_2) \geq \mathsf{Weight}(C_2')$ as C_2 is P_2's best chain in round r_2 and $\sum_{B \in \mathcal{B}_{\text{li}}^{h_0}} \mathsf{Weight}(B) \geq \check{W}_{\mathsf{LeftIso}^{h_0}}(r_2 - r_1 - 2\Delta_{\text{NET}} + 1)$ except with probability $\check{p}_{\mathsf{LeftIso}^{h_0}}(r_2 - r_1 - 2\Delta_{\text{NET}} + 1)$. $\square$

When this theorem is instantiated with $P_1 = P_2$, we obtain chain growth for $\rho > 2\Delta_{\text{NET}}$ and $\tau = \frac{\check{W}_{\mathsf{LeftIso}^{h_0}}(\rho - 2\Delta_{\text{NET}})}{\rho}$ except with probability $\check{p}_{\mathsf{LeftIso}^{h_0}}(\rho - 2\Delta_{\text{NET}})$.

2.3.2 Chain Quality

The chain quality property intuitively says that within any consecutive chunk of blocks of an honest party's chain, at least a ratio of the blocks was produced by honest parties. We give a formal definition next.

Definition 2.3.5 (Chain Quality)**.** The chain quality property with parameters $\Lambda \in \mathbb{R}$ and $\mu \in \mathbb{R}$ states that for any honest party P that has a chain C as their best chain, it holds that for any sequence of consecutive blocks with a weight range of size at least Λ in C, it holds that the ratio of honest weight is at least μ.

We believe it is more intuitive to reason about the chain quality property in terms of *elapsed time* instead of weight. Hence, we present our results for a "timed" version of the chain quality property,[5] which intuitively ensures that a fraction of honest weight is contained in a sequence of blocks mined within some period.

Theorem 2.3.6 (Chain quality)**.** *Let P be an honest party with best chain $C = B_1 B_2 \dots B_n$ and let $R = B_i \dots B_j$ be any consecutive list of blocks in C with $1 \leq i < j \leq n$ where block B_i was mined in round r_i, B_j in round r_j, and $r_j - r_i \geq 2\Delta_{\text{NET}}$.*

Further let $h_0 \in \mathcal{H}$ and $X \in \mathbb{R}$ such that the weight function is $\left(\check{W}_{\mathsf{LeftIso}^{h_0}}, \check{p}_{\mathsf{LeftIso}^{h_0}}\right)$-left-isolated-lower-bounding and $\left(\hat{W}_{q\beta}, \hat{p}_{q\beta}\right)$-upper-bounding such that for any $\rho \geq r_j - r_i$, we have $\check{W}_{\mathsf{LeftIso}^{h_0}}(\rho - 2\Delta_{\text{NET}} + 1) \geq \hat{W}_{q\beta}(\rho) + X$. Finally let p_{bad} be the probability that the fraction of honest weight in R is less than $\frac{X}{\mathsf{Weight}(R)}$. Then,

$$p_{\mathsf{bad}} \leq \check{p}_{\mathsf{LeftIso}^{h_0}}(r_j - r_i - 2\Delta_{\text{NET}} + 1) + \hat{p}_{q\beta}(r_j - r_i).$$

Proof. Let $\hat{\imath}$ be the largest value such that $\hat{\imath} \leq i$ and $B_{\hat{\imath}}$ was mined by an honest party[6]. This is well defined as the genesis block B_1 is honest by definition. Let $\hat{\jmath}$ be the smallest value such that $\hat{\jmath} \geq j$ and there exists a round such that an honest player had that $B_1 \dots B_{\hat{\jmath}}$ was his best chain. Now let $r_{\hat{\imath}}$ be the round that $B_{\hat{\imath}}$ was created and let $r_{\hat{\jmath}}$ be the first round that an honest player had $B_1 \dots B_{\hat{\jmath}}$ as his best chain. This is well defined as B_n is actually the head of the best chain of an honest party.

Note that in round $r_{\hat{\imath}}$ was $B_1 \dots B_{\hat{\imath}}$ actually the best chain of the honest party who baked this block. By Theorem 2.3.4 do we thus know that

$$\Pr\left[\mathsf{Weight}_{\mathsf{w}}(B_{\hat{\imath}} \dots B_{\hat{\jmath}}) < \check{W}_{\mathsf{LeftIso}^{h_0}}(r_{\hat{\jmath}} - r_{\hat{\imath}} - 2\Delta_{\text{NET}} + 1)\right] \leq \check{p}_{\mathsf{LeftIso}^{h_0}}(r_{\hat{\jmath}} - r_i - 2\Delta_{\text{NET}} + 1), \tag{2.15}$$

as $B_1 \dots B_{\hat{\jmath}}$ could otherwise not be the best chain of any honest party in round $r_{\hat{\jmath}}$. On the other hand is the probability that the adversary him self have been able to generate more than $\hat{W}_{q\beta}(r_{\hat{\jmath}} - r_{\hat{\imath}})$ weight less than $\hat{p}_{q\beta}(r_{\hat{\jmath}} - r_{\hat{\imath}})$. As $\hat{\imath} < i$ and $\hat{\jmath} < j$ implies that $B_{\hat{\imath}+1} \dots B_i$ and $B_j \dots B_{\hat{\imath}}$ are all dishonest blocks, does this imply that at least X honest weight will be in R unless with probability $\check{p}_{\mathsf{LeftIso}^{h_0}}(r_{\hat{\jmath}} - r_{\hat{\imath}} - 2\Delta_{\text{NET}} + 1) + \hat{p}_{q\beta}(r_{\hat{\jmath}} - r_{\hat{\imath}})$. The statement now follows from the fact that the probability functions are monotonically decreasing and that $r_{\hat{\jmath}} - r_{\hat{\imath}} \geq r_j - r_i$. □

Next, we state the weighted chain quality theorem. We use Theorem 2.3.6 together with the fact that the amount of weight produced during a period is bounded; moreover, we use the collective mining rate to do this mapping, which is by no means a tight bound.

[5]We omit the formal definition here as it can be easily derived from Definition 2.3.5.

[6]Note that instead of defining $\hat{\imath}$ such that $B_{\hat{\imath}}$ is an honest block it could also have been defined as the largest index less than i such that there existed an honest party that had $B_{\hat{\imath}}$ as the head of his best chain. Even though that this does gives an $\hat{\imath}$ "closer" to i, this does not increase our bounds.

Corollary 2.3.7 (Weighted chain quality). *Let P be an honest party, let R be any consecutive list of blocks from the best chain of this party, and let $\rho \in \mathbb{N}$, $\rho \geq 2\Delta_{\text{NET}}$ be the largest value such that $\hat{W}_q(\rho) \leq \mathsf{Weight}(R)$. Further, let $h_0 \in \mathcal{H}$ and $X \in \mathbb{R}$ such that the weight function is $\left(\check{W}_{\mathsf{LeftIso}^{h_0}}, \check{p}_{\mathsf{LeftIso}^{h_0}}\right)$-left-isolated-lower-bounding and $(\hat{W}_{q\beta}, \hat{p}_{q\beta})$-upper-bounding such that for any $\rho' \geq \rho$, we have $\check{W}_{\mathsf{LeftIso}^{h_0}}(\rho' - 2\Delta_{\text{NET}} + 1) \geq \hat{W}_{q\beta}(\rho') + X$. Let p_{bad} be the probability that the fraction of honest weight in R is less than $\frac{X}{\mathsf{Weight}(R)}$. Then,*

$$p_{\mathsf{bad}} \leq \check{p}_{\mathsf{LeftIso}^{h_0}}(\rho - 2\Delta_{\text{NET}} + 1) + \hat{p}_{q\beta}(\rho) + \hat{p}_q(\rho).$$

Proof. By our assumption on the weight function, it took at least ρ rounds to produce R, except with probability $\hat{p}_q(\rho)$. We can thus apply Theorem 2.3.6 to conclude the proof of the theorem. □

2.3.3 Common Prefix

The common prefix property is arguably the most important property of blockchains. It informally says that the chains of honest parties are always a common prefix of each other after removing some blocks on the chain. Next, we define our two variants of the common prefix property. The first variant is with respect to the absolute number of rounds, which states that for any pair of honest parties that adopted chains at different rounds, the oldest chain is a prefix of the most recent chain. The second variant is analogous but concerning the accumulated weight.

Definition 2.3.8 (Pruning). Let C be a chain, $w \in \mathbb{R}$ be a weight, and let $r \in \mathbb{N}$ be a round. We define $C^{\mathrm{W}\lceil w}$ to be the longest prefix of C such that $\mathsf{Weight}(C^{\mathrm{W}\lceil w}) \leq \mathsf{Weight}(C) - w$, i.e., blocks with total weight at least w are removed from the end of C. We further define $C^{\mathrm{R}>\lceil r}$ to be the chain containing all blocks from C that were mined until round r, i.e., all blocks mined after round r are removed from C.

Definition 2.3.9 (Common Prefix). For parameter $\rho \in \mathbb{N}$, let C_1 be the best chain of honest party P_1 in round r_1, and let C_2 be the best chain of honest party P_2 in round r_2 for $r_1 \leq r_2$. The common-prefix property says that $C_1^{\mathrm{R}>\lceil r_1-\rho} \preceq C_2$.

Definition 2.3.10 (Weighted Common Prefix). For parameter $\omega \in \mathbb{R}$, let C_1 be the best chain of honest party P_1 in round r_1, and let C_2 be the best chain of honest party P_2 in round r_2 for $r_1 \leq r_2$. Then, $C_1^{\mathrm{W}\lceil \omega} \preceq C_2$.

Similarly to [GKL15], we prove our common prefix property in two steps. First, in Lemma 2.3.11, we show a weaker version of the property that says that the best chain of any pair of honest players at the *same* round must be a prefix of each other. Then, in Theorem 2.3.12, we prove Definition 2.3.9 by extending the proof to capture the case where the honest parties might be at different rounds.

Lemma 2.3.11 (Common-prefix lemma). *Let r be some round and let P_1 be some honest party with best chain C_1 in round r. Let p_{bad} be the probability that there is some chain C_2 such that all blocks on C_2 have been mined until round r, $\mathsf{Weight}(C_2) \geq \mathsf{Weight}(C_1)$, and the deepest honest common block $\hat{B}_0$ in C_1 and C_2 is mined in some round $r_0 \leq r - 2\Delta_{\text{NET}} + 1$. We then have the following two properties.*

(i) *For all $h_0 \in \mathcal{H}$ such that the weight function is $\left(\check{W}_{\mathsf{LeftIso}^{h_0}}, \check{p}_{\mathsf{LeftIso}^{h_0}}\right)$-left-isolated-lower-bounding and $(\hat{W}_q, \hat{p}_q)$-upper-bounding with*

$$2 \cdot \check{W}_{\mathsf{LeftIso}^{h_0}}(r - r_0 - 2\Delta_{\text{NET}} + 1) \geq \hat{W}_q(r - r_0),$$

we have

$$p_{\mathsf{bad}} \leq \check{p}_{\mathsf{LeftIso}^{h_0}}(r - r_0 - 2\Delta_{\text{NET}} + 1) + \hat{p}_q(r - r_0).$$

(ii) *For all $h_0 \in \mathcal{H}$ such that the weight function is $(\hat{W}_q^{\leq h_0}, \hat{p}_q^{\leq h_0})$-below-threshold-upper-bounding, $(\hat{W}_{q\beta}^{>h_0}, \hat{p}_{q\beta}^{>h_0})$-upper-bounding, and $\left(\check{W}_{\mathsf{Iso}^{h_0}}, \check{p}_{\mathsf{Iso}^{h_0}}\right)$-isolated-lower-bounding with*

$$\check{W}_{\mathsf{Iso}^{h_0}}(r - r_0 - 2\Delta_{\text{NET}} + 1) \geq \hat{W}_q^{\leq h_0}(r - r_0) + \hat{W}_{q\beta}^{>h_0}(r - r_0),$$

we have

$$p_{\mathsf{bad}} \leq \check{p}_{\mathsf{Iso}^{h_0}}(r - r_0 - 2\Delta_{\text{NET}} + 1) + \hat{p}_q^{\leq h_0}(r - r_0) + \hat{p}_{q\beta}^{>h_0}(r - r_0).$$

Proof. Assume a chain C_2 as described exists and let B_0 be the deepest common block in C_1 and C_2[7]. Let $\mathcal{B}_{\text{li}}^{h_0}$ and $\mathcal{B}_{\text{iso}}^{h_0}$ be the set of all h_0-left-isolated blocks and the set of all h_0-isolated blocks mined in some round in $[r_0 + \Delta_{\text{NET}}, r - \Delta_{\text{NET}}]$, respectively. Further let $\mathcal{B}_{\text{nli}}^{h_0}$, $\mathcal{B}_{\text{hon}}^{\leq h_0}$, and $\mathcal{B}_{\text{dis}}$ be the sets of all non-$(\Delta_{\text{NET}}, h_0)$-left-isolated blocks, all honest blocks with hash value at most h_0, and all dishonest blocks mined in some round in $(r_0, r]$, respectively. We define $W_{\text{li}}^{h_0} := \bigcup_{B \in \mathcal{B}_{\text{li}}^{h_0}} \mathsf{WeightRange}(B)$, $W_{\text{iso}}^{h_0} := \bigcup_{B \in \mathcal{B}_{\text{iso}}^{h_0}} \mathsf{WeightRange}(B)$, $W_{\text{nli}}^{h_0} := \bigcup_{B \in \mathcal{B}_{\text{nli}}^{h_0}} \mathsf{WeightRange}(B)$, $W_{\text{hon}}^{\leq h_0} := \bigcup_{B \in \mathcal{B}_{\text{hon}}^{\leq h_0}} \mathsf{WeightRange}(B)$, and $W_{\text{dis}} := \bigcup_{B \in \mathcal{B}_{\text{dis}}} \mathsf{WeightRange}(B)$ to be the sets of all weight depths in the weight ranges of the corresponding blocks.

We claim that

$$W_{\text{li}}^{h_0} \subseteq W_{\text{nli}}^{h_0}, \tag{2.16}$$
$$W_{\text{iso}}^{h_0} \subseteq W_{\text{hon}}^{\leq h_0} \cup W_{\text{dis}}. \tag{2.17}$$

To prove these claims, we first show that

$$W_{\text{li}}^{h_0}, W_{\text{iso}}^{h_0} \subseteq \left(\mathsf{EndWeight}(\hat{B}_0), \mathsf{EndWeight}(C_1)\right]. \tag{2.18}$$

All honest parties mining blocks in round $r_0 + \Delta_{\text{NET}}$ or later know about $\hat{B}_0$ and will therefore only extend chains with weight at least $\mathsf{EndWeight}(\hat{B}_0)$. Likewise, if some honest block with weight depth more than $\mathsf{EndWeight}(C_1)$ were mined until round $r - \Delta_{\text{NET}}$, no honest party would consider C_1 the best chain in round r.

We next show that descendants of $\hat{B}_0$ on C_1 or C_2 are mined in some round in $(r_0, r]$. Since $\hat{B}_0$ is honest, it is unknown to any party before r_0. All descendants of $\hat{B}_0$ are thus

[7]Note that if B_0 is honest, we have $\hat{B}_0 = B_0$. The reason for considering $\hat{B}_0$ in addition to B_0 is that only honest parties are guaranteed to broadcast blocks they mine immediately. Hence, for an honest $\hat{B}_0$, we know that other honest parties will know that block at most Δ_{NET} rounds after it was mined.

mined after round r_0.[8] Furthermore, honest parties only adopt chains containing blocks they know, which means all blocks on C_1 are mined until round r. The same holds for C_2 by assumption. We finally prove Equations (2.16) and (2.17). To this end, let $w \in W_{\mathrm{li}}^{h_0}$ or $w \in W_{\mathrm{iso}}^{h_0}$. We consider the following cases:

$w \in \big(\mathsf{EndWeight}(\hat{B}_0), \mathsf{EndWeight}(B_0)\big]$: There is a block on the chain from $\hat{B}_0$ to B_0 (excluding $\hat{B}_0$) whose weight range includes w. Since all these blocks are dishonest, they are, in particular non-h_0-left-isolated. Furthermore, they are descendants of $\hat{B}_0$ and are on C_1 and are thus mined in some round in $(r_0, r]$. Hence, $w \in W_{\mathrm{dis}} \subseteq W_{\mathrm{nli}}^{h_0}$.

$w \in \big(\mathsf{EndWeight}(B_0), \mathsf{EndWeight}(C_1)\big]$: There are blocks both on C_1 and on C_2 (and potentially more) that cover w. If $w \in W_{\mathrm{li}}^{h_0}$, Lemma 2.3.1 implies that there is a non-h_0-left-isolated block B' covering w on at least one of these chains. If $w \in W_{\mathrm{iso}}^{h_0}$, Lemma 2.3.2 implies that there is a block B' on one of these chains that is not both honest and has a hash value above h_0. Since B' in both cases is a descendant of $\hat{B}_0$ and on C_1 or C_2, it was mined in some round in $(r_0, r]$. We can therefore conclude that $w \in W_{\mathrm{nli}}^{h_0}$ in the first case, and $w \in W_{\mathrm{hon}}^{\leq h_0} \cup W_{\mathrm{dis}}$ in the latter case.

All cases together imply Equations (2.16) and (2.17).

We now prove Item (i) of the lemma. Since left-isolated blocks have disjoint weight ranges by Lemma 2.3.1, Equation (2.16) implies

$$w_{\mathrm{li}} := \sum_{B \in \mathcal{B}_{\mathrm{li}}^{h_0}} \mathsf{Weight}(B) \leq \sum_{B \in \mathcal{B}_{\mathrm{nli}}^{h_0}} \mathsf{Weight}(B) =: w_{\mathrm{nli}}. \tag{2.19}$$

Let w be the total weight of all blocks mined in some round in $(r_0, r]$. Recall that $\mathcal{B}_{\mathrm{li}}^{h_0}$ are all h_0-left-isolated blocks mined in some round in $[r_0 + \Delta_{\mathrm{NET}}, r - \Delta_{\mathrm{NET}}]$, and $\mathcal{B}_{\mathrm{nli}}^{h_0}$ are all non-$(\Delta_{\mathrm{NET}}, h_0)$-left-isolated blocks mined in some round in $(r_0, r]$. Since $[r_0 + \Delta_{\mathrm{NET}}, r - \Delta_{\mathrm{NET}}] \subseteq (r_0, r]$, we have $w_{\mathrm{nli}} \leq w - w_{\mathrm{li}}$. Hence,

$$2w_{\mathrm{li}} \leq w. \tag{2.20}$$

By assumption on the weight function, $w_{\mathrm{li}} > \check{W}_{\mathsf{LeftIso}^{h_0}}(r - r_0 - 2\Delta_{\mathrm{NET}} + 1)$ and $w < \hat{W}_q(r - r_0)$, except with probability $\check{p}_{\mathsf{LeftIso}^{\Delta_{\mathrm{NET}}}} h_0(r - r_0 - 2\Delta_{\mathrm{NET}} + 1) + \hat{p}_q(r - r_0)$. We can thus conclude by our assumptions on these quantities that the inequality $2w_{\mathrm{li}} \leq w$ can only hold with at most this probability, which concludes the proof of Item (i).

We finally prove Item (ii). By Lemma 2.3.2, isolated blocks have disjoint weight ranges. Hence, Equation (2.17) implies

$$w_{\mathrm{iso}} \leq w_{\mathrm{hon}}^{\leq h_0} + w_{\mathrm{dis}}, \tag{2.21}$$

for $w_{\mathrm{iso}} := \sum_{B \in \mathcal{B}_{\mathrm{iso}}^{h_0}} \mathsf{Weight}(B)$, $w_{\mathrm{hon}}^{\leq h_0} := \sum_{B \in \mathcal{B}_{\mathrm{hon}}^{\leq h_0}} \mathsf{Weight}(B)$, and $w_{\mathrm{dis}} := \sum_{B \in \mathcal{B}_{\mathrm{dis}}} \mathsf{Weight}(B)$.

[8] Assuming there are no collisions in the random oracle, as otherwise, a dishonest party could extend $\hat{B}_0$ before it is mined by an honest party.

The dishonest blocks can be split up into the dishonest blocks with a hash below h_0 which we denote $\mathcal{B}_{\text{dis}}^{\leq h_0}$ and the blocks above which we denote $\mathcal{B}_{\text{dis}}^{>h_0}$. We let $w_{\text{dis}}^{\leq h_0} := \sum_{B \in \mathcal{B}_{\text{dis}}^{\leq h_0}} \mathsf{Weight}(B)$ and $w_{\text{dis}}^{>h_0} := \sum_{B \in \mathcal{B}_{\text{dis}}^{>h_0}} \mathsf{Weight}(B)$, which gives us that $w_{\text{dis}} = w_{\text{dis}}^{\leq h_0} + w_{\text{dis}}^{>h_0}$. We note that $w_{\text{hon}}^{\leq h_0} + w_{\text{dis}}^{\leq h_0}$ is upper-bounded by $\hat{W}_q^{\leq h_0}$ except with probability $\hat{p}_q^{\leq h_0}$. Together with the assumptions on $\check{W}_{\mathsf{Iso}^{h_0}}$, and $\hat{W}_{q\beta}^{>h_0}$, Item (ii) follows. □

Theorem 2.3.12 (Common prefix)**.** *Let $\rho \geq 2\Delta_{\text{NET}} - 1$, let P_1, P_2 be (not necessarily different) honest parties, let $r_1 \leq r_2$ be rounds, and let C_1 be the best chain of P_1 in round r_1. Further let p_{bad} be the probability that P_2 has a best chain C_2 in round r_2 with $C_1{}^{\mathrm{R}>\lceil r_1-\rho} \npreceq C_2$. We have*

(i) For all $h_0 \in \mathcal{H}$ such that the weight function is $\left(\check{W}_{\mathsf{LeftIso}^{h_0}}, \check{p}_{\mathsf{LeftIso}^{h_0}}\right)$-left-isolated-lower-bounding and $(\hat{W}_q, \hat{p}_q)$-upper-bounding, and for all $\rho' \geq \rho$

$$2 \cdot \check{W}_{\mathsf{LeftIso}^{h_0}}(\rho' - 2\Delta_{\text{NET}} + 1) \geq \hat{W}_q(\rho'),$$

we have

$$p_{\mathsf{bad}} \leq 2\check{p}_{\mathsf{LeftIso}^{h_0}}(\rho - 2\Delta_{\text{NET}} + 1) + 2\hat{p}_q(\rho).$$

(ii) For all $h_0 \in \mathcal{H}$ such that the weight function is $(\hat{W}_q^{\leq h_0}, \hat{p}_q^{\leq h_0})$-below-threshold-upper-bounding, $(\hat{W}_{q\beta}^{>h_0}, \hat{p}_{q\beta}^{>h_0})$-upper-bounding, and $\left(\check{W}_{\mathsf{Iso}^{h_0}}, \check{p}_{\mathsf{Iso}^{h_0}}\right)$-isolated-lower-bounding, and for all $\rho' \geq \rho$

$$\check{W}_{\mathsf{Iso}^{h_0}}(\rho' - 2\Delta_{\text{NET}} + 1) \geq \hat{W}_q^{\leq h_0}(\rho') + \hat{W}_{q\beta}^{>h_0}(\rho'),$$

we have

$$p_{\mathsf{bad}} \leq 2\check{p}_{\mathsf{Iso}^{h_0}}(\rho - 2\Delta_{\text{NET}} + 1) + 2\hat{p}_q^{\leq h_0}(\rho) + 2\hat{p}_{q\beta}^{>h_0}(\rho).$$

Proof. Assume the best chain C_2 of P_2 in round r_2 is such that $C_1{}^{\mathrm{R}>\lceil r_1-\rho} \npreceq C_2$, and let $r \leq r_2$ be the first round with $r \geq r_1$ in which some honest party P_2' (not necessarily P_1 or P_2) adopted a chain C_2' with $C_1{}^{\mathrm{R}>\lceil r_1-\rho} \npreceq C_2'$. We distinguish two cases:

Case 1: $r = r_1$. In this case, all blocks on C_2' have been mined until round r_1. Let r_0 be the round in which the deepest honest common block in C_1 and C_2' has been mined. Since $C_1{}^{\mathrm{R}>\lceil r_1-\rho} \npreceq C_2'$, we have $r_0 \leq r_1 - \rho \leq r_1 - 2\Delta_{\text{NET}} + 1$. Now let $C_1^\star \in \{C_1, C_2'\}$ be the chain with the smaller or equal $\mathsf{EndWeight}$, and let $C_2^\star$ be the other one. Note that $C_1^\star$ is the best chain of some honest party in round r, and all blocks on $C_2^\star$ have been mined until round r. We can thus apply Lemma 2.3.11 to obtain that the probability of this case for Item (i) is at most

$$\check{p}_{\mathsf{LeftIso}^{h_0}}(r - r_0 - 2\Delta_{\text{NET}} + 1) + \hat{p}_q(r - r_0) \leq \check{p}_{\mathsf{LeftIso}^{h_0}}(\rho - 2\Delta_{\text{NET}} + 1) + \hat{p}_q(\rho), \quad (2.22)$$

and for Item (ii)

$$\begin{aligned} &\check{p}_{\mathsf{Iso}^{h_0}}(r - r_0 - 2\Delta_{\text{NET}} + 1) + \hat{p}_q^{\leq h_0}(r - r_0) + \hat{p}_{q\beta}^{>h_0}(r - r_0) \\ &\qquad \leq \check{p}_{\mathsf{Iso}^{h_0}}(\rho - 2\Delta_{\text{NET}} + 1) + \hat{p}_q^{\leq h_0}(\rho) + \hat{p}_{q\beta}^{>h_0}(\rho), \end{aligned} \quad (2.23)$$

where we used $r - r_0 \geq \rho$ and the monotonicity of the probabilities.

Case 2: $r > r_1$. Let C_1' be the best chain of P_2' in round $r-1 \geq r_1$. We then have ${C_1}^{\mathrm{R}>\lceil r_1-\rho} \preceq C_1'$. This implies that C_2' cannot result from extending C_1', and therefore, C_2' must have been sent to P_2' from another party. Hence, all blocks in C_2' have been mined until round $r-1$. Since P_2' adopts C_2', we further have $\mathsf{Weight}(C_2') > \mathsf{Weight}(C_1')$. We claim that ${C_1'}^{\mathrm{R}>\lceil r_1-\rho} \not\preceq C_2'$. If this was not the case, C_1' and C_2' would agree on all blocks mined until round $r_1-\rho$. Since ${C_1}^{\mathrm{R}>\lceil r_1-\rho} \preceq C_1'$, C_1' also agrees with C_1 on all such blocks. That would imply ${C_1}^{\mathrm{R}>\lceil r_1-\rho} \preceq C_2'$, contradicting the definition of C_2'. We therefore have ${C_1'}^{\mathrm{R}>\lceil r_1-\rho} \not\preceq C_2'$. Let r_0 be the round in which the deepest honest common block in C_1' and C_2' was mined. We have $r_0 \leq r_1 - \rho \leq (r-1) - 2\Delta_{\mathrm{NET}} + 1$. We can therefore apply Lemma 2.3.11 with chains C_1' and C_2' in round $r-1$. For Item (i), we obtain that the given situation can only occur with probability at most

$$\check{p}_{\mathsf{LeftIso}^{h_0}}(r-1-r_0-2\Delta_{\mathrm{NET}}+1) + \hat{p}_q(r-1-r_0). \tag{2.24}$$

Using $(r-1)-r_0 \geq r_1 - r_0 \geq r_1 - (r_1-\rho) = \rho$ and the monotonicity of the probabilities, this probability can be upper bounded by $\check{p}_{\mathsf{LeftIso}^{h_0}}(\rho - 2\Delta_{\mathrm{NET}}+1) + \hat{p}_q(\rho)$.

For Item (ii), we obtain that the given situation can only occur with probability at most

$$\begin{aligned}\check{p}_{\mathsf{Iso}^{h_0}}(r-1-r_0-2\Delta_{\mathrm{NET}}+1) + \hat{p}_q^{\leq h_0}(r-1-r_0) + \hat{p}_{q\beta}^{>h_0}(r-1-r_0)&\\ \leq \check{p}_{\mathsf{Iso}^{h_0}}(\rho-2\Delta_{\mathrm{NET}}+1) + \hat{p}_q^{\leq h_0}(\rho) + \hat{p}_{q\beta}^{>h_0}(\rho).&\end{aligned} \tag{2.25}$$

We can conclude that the probability that case 1 or case 2 occurs is at most the sum of the two probabilities derived in these cases. □

Since the produced weight per round is bounded, this directly implies a common-prefix property for pruning blocks with a certain amount of weight. We formalize this fact in the following corollary that proves the weighted common prefix property (Definition 2.3.10).

Corollary 2.3.13 (Weighted common prefix)**.** *Let $\omega \in \mathbb{R}$, and let $\rho \in \mathbb{N}$ be the largest value such that $\hat{W}_q(\rho) \leq \omega$ and $\rho \geq 2\Delta_{\mathrm{NET}} - 1$. Further let $h_0 \in \mathcal{H}$ such that the weight function is $\left(\check{W}_{\mathsf{LeftIso}^{h_0}}, \check{p}_{\mathsf{LeftIso}^{h_0}}\right)$-left-isolated-lower-bounding and $(\hat{W}_q, \hat{p}_q)$-upper-bounding, and for all $\rho' \geq \rho$, we have $2 \cdot \check{W}_{\mathsf{LeftIso}^{h_0}}(\rho' - 2\Delta_{\mathrm{NET}}+1) \geq \hat{W}_q(\rho')$. Let P_1, P_2 be (not necessarily different) honest parties, let $r_1 \leq r_2$ be rounds, and let C_1 be the best chain of P_1 in round r_1. Then, the probability that P_2 has a best chain C_2 in round r_2 with ${C_1}^{\mathrm{W}\lceil\omega} \not\preceq C_2$ is at most*

$$2\check{p}_{\mathsf{LeftIso}^{h_0}}(\rho - 2\Delta_{\mathrm{NET}}+1) + 2\hat{p}_q(\rho).$$

Proof. By our assumption on the weight function, there is at most $\hat{W}_q(\rho) < \omega$ weight produced in ρ rounds, except with probability $\hat{p}_q(\rho)$. In this case, all blocks on ${C_1}^{\mathrm{W}\lceil\omega}$ are mined before round $r_1-\rho$, i.e., ${C_1}^{\mathrm{W}\lceil\omega} \preceq {C_1}^{\mathrm{R}>\lceil r_1-\rho}$. Therefore, we have ${C_1}^{\mathrm{R}>\lceil r_1-\rho} \not\preceq C_2$. We can thus apply Theorem 2.3.12 to conclude the proof of the theorem. □

2.4 Applying the Framework to Capped Weight Functions

Our framework allows the exploration of infinitely many different weight functions. Intuitively, good weight functions should ensure that a majority of the weight is produced by honest parties that have a nearly complete view of all other honest blocks, i.e., the winning events that produce most of the weight should, on average, occur so rarely that they have enough time to propagate before the next time such a rare event occurs. On the other hand, the weight difference between such winning events should not be too large as this increases the variance and thus gives worse bounds on the probabilities.

These considerations led us to focus on a particular class of functions we call *capped weight functions* that we use our framework to analyze in this section. We first prove general conditions that ensure common prefix for this class of functions using only very loose bounds. Next, we derive a condition that such functions should satisfy to provide a weak form of optimistic responsiveness. We then discuss how to pick such functions and how combining such a function with a finality layer provides very fast confirmation. Finally, we show how previous analyses of Bitcoin are subsumed by our framework and present a weight function that is strictly better than the Bitcoin function with respect to the properties presented in this work.

2.4.1 Definitions and General Results

To derive concrete equations for the bounds the weight functions should satisfy, we instantiate Theorem 2.3.12 with the *loose* bounds from Section 2.2.5. The specific conditions we achieve for *any* weight function are captured by the lemma below.

Lemma 2.4.1. *Let* w *be a weight-function. Further let* $h_0 \in \mathcal{H}$. *We assume that* $\mathsf{w}_{\min>h_0} > 0$. *Let* $\delta \in (0,1)$ *and* $\rho > 2\Delta_{\mathrm{NET}} - 1$ *such that*

$$\begin{aligned}&\alpha \cdot (1-\delta) \cdot (1-p_{\le h_0}) \cdot (p_{\le h_0})^{2q\Delta_{\mathrm{NET}}} \\ &\ge \frac{\rho}{\rho - 2\Delta_{\mathrm{NET}} + 1}\left(\frac{\mathsf{w}_{\max\le h_0}}{\mathsf{w}_{\min>h_0}} \cdot p_{\le h_0} + \frac{\mathsf{w}_{\max>h_0}}{\mathsf{w}_{\min>h_0}} \cdot \beta \cdot (1-p_{\le h_0})\right).\end{aligned}$$

Let P_1, P_2 *be (not necessarily different) honest parties, let* $r_1 \le r_2$ *be rounds, and let* C_1 *be the best chain of* P_1 *in round* r_1. *Finally let* p_{bad} *be the probability that* P_2 *has a best chain* C_2 *in round* r_2 *with* $C_1^{\mathsf{R}>\lceil r_1-\rho} \not\preceq C_2$. *We then have*

(i) for any β

$$p_{\mathsf{bad}} \le 10e^{-\frac{\delta^2 \cdot q\beta \cdot (\rho - 2\Delta_{\mathrm{NET}}+1)\cdot(1-p_{\le h_0})\cdot(p_{\le h_0})^{2q\Delta_{\mathrm{NET}}}}{432}},$$

(ii) and for $\beta = 0$

$$p_{\mathsf{bad}} \le 8e^{-\frac{\delta^2 \cdot q \cdot (\rho - 2\Delta_{\mathrm{NET}}+1)\cdot(1-p_{\le h_0})\cdot(p_{\le h_0})^{2q\Delta_{\mathrm{NET}}}}{432}}.$$

Proof. We want to use Theorem 2.3.12 (ii), and to this end, we show that the weight function satisfies

$$\check{W}_{\mathsf{Iso}^{h_0}}(\rho - 2\Delta_{\mathrm{NET}} + 1) \ge \hat{W}_q^{\le h_0}(\rho) + \hat{W}_{q\beta}^{>h_0}(\rho). \tag{2.26}$$

Let $\delta' := \frac{\delta}{2}$. Lemma 2.2.8 (ii) implies that w is $\left(\check{W}_{\mathsf{Iso}^{h_0}}, \check{p}_{\mathsf{Iso}^{h_0}}\right)$-isolated-lower-bounding with

$$\begin{aligned} &\check{W}_{\mathsf{Iso}^{h_0}}(\rho - 2\Delta_{\text{NET}} + 1) \\ &= \mathsf{w}_{\min>h_0} \cdot (1-\delta') \cdot q\alpha \cdot (\rho - 2\Delta_{\text{NET}} + 1) \cdot (1 - p_{\leq h_0}) \cdot (p_{\leq h})^{2q\Delta_{\text{NET}}}, \\ &\check{p}_{\mathsf{Iso}^{h_0}}(\rho - 2\Delta_{\text{NET}} + 1) = 3 \cdot e^{-\frac{(\delta')^2 \cdot q\alpha \cdot (\rho - 2\Delta_{\text{NET}}+1) \cdot (1-p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\text{NET}}}}{108}}. \end{aligned} \tag{2.27}$$

Lemma 2.2.6 (i) yields using $\alpha + \beta = 1$ and $\alpha > \beta$, that w is $\left(\hat{W}_q^{\leq h_0}, \hat{p}_q^{\leq h_0}\right)$-below-threshold-upper-bounding with

$$\begin{aligned} \hat{W}_q^{\leq h_0} &= \mathsf{w}_{\max\leq h_0} \cdot (1+\delta') \cdot q \cdot \rho \cdot p_{\leq h_0} \\ \hat{p}_q^{\leq h_0}(\rho) &= e^{-\frac{(\delta')^2 \cdot q \cdot \rho \cdot p_{\leq h_0}}{3}} \leq e^{-\frac{(\delta')^2 \cdot \beta q \cdot \rho \cdot p_{\leq h_0}}{3}} \end{aligned} \tag{2.28}$$

Finally, Lemma 2.2.6 (ii) implies that w is $\left(\hat{W}_{q\beta}^{>h_0}, \hat{p}_{q\beta}^{>h_0}\right)$-above-threshold-upper-bounding with

$$\begin{aligned} \hat{W}_{q\beta}^{>h_0}(\rho) &= \mathsf{w}_{\max>h_0} \cdot (1+\delta') \cdot q\beta \cdot \rho \cdot (1 - p_{\leq h_0}), \\ \hat{p}_{q\beta}^{>h_0}(\rho) &= e^{-\frac{(\delta')^2 \cdot q\beta \cdot \rho \cdot (1-p_{\leq h_0})}{3}}. \end{aligned} \tag{2.29}$$

We can conclude that Equation (2.26) is satisfied if

$$\begin{aligned} &\mathsf{w}_{\min>h_0} \cdot (1-\delta') \cdot q\alpha \cdot (\rho - 2\Delta_{\text{NET}} + 1) \cdot (1 - p_{\leq h_0}) \cdot (p_{\leq h})^{2q\Delta_{\text{NET}}} \\ &\quad \geq \mathsf{w}_{\max\leq h_0} \cdot (1+\delta') \cdot q \cdot \rho \cdot p_{\leq h_0} + \mathsf{w}_{\max>h_0} \cdot (1+\delta') \cdot q\beta \cdot \rho \cdot (1 - p_{\leq h_0}). \end{aligned} \tag{2.30}$$

This is equivalent to

$$\begin{aligned} &\frac{1-\delta'}{1+\delta'} \cdot \alpha \cdot (1 - p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\text{NET}}} \\ &\geq \frac{\rho}{\rho - 2\Delta_{\text{NET}} + 1} \left(\frac{\mathsf{w}_{\max\leq h_0}}{\mathsf{w}_{\min>h_0}} \cdot p_{\leq h_0} + \frac{\mathsf{w}_{\max>h_0}}{\mathsf{w}_{\min>h_0}} \cdot \beta \cdot (1 - p_{\leq h_0}) \right). \end{aligned} \tag{2.31}$$

Note that $\frac{1-\delta'}{1+\delta'} \geq 1 - \delta$ because $\delta' = \frac{\delta}{2}$. Hence this condition is satisfied by the assumption in the lemma statement. Further note that $\frac{\rho}{\rho - 2\Delta_{\text{NET}}+1}$ is monotonically decreasing in ρ, and thus the condition is also satisfied for all $\rho' \geq \rho$. We can therefore apply Theorem 2.3.12 (ii) to obtain

$$\begin{aligned} p_{\mathsf{bad}} &\leq 6e^{-\frac{(\delta')^2 \cdot q\alpha \cdot (\rho - 2\Delta_{\text{NET}}+1) \cdot (1-p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\text{NET}}}}{108}} + 2e^{-\frac{(\delta')^2 \cdot q\beta \cdot \rho \cdot p_{\leq h_0}}{3}} \\ &\quad + 2e^{-\frac{(\delta')^2 \cdot q\beta \cdot \rho \cdot (1-p_{\leq h_0})}{3}} \\ &= 6e^{-\frac{\delta^2 \cdot q\alpha \cdot (\rho - 2\Delta_{\text{NET}}+1) \cdot (1-p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\text{NET}}}}{432}} + 2e^{-\frac{\delta^2 \cdot q\beta \cdot \rho \cdot p_{\leq h_0}}{12}} + 2e^{-\frac{\delta^2 \cdot q\beta \cdot \rho \cdot (1-p_{\leq h_0})}{12}} \\ &\leq 10e^{-\frac{\delta^2 \cdot q\beta \cdot (\rho - 2\Delta_{\text{NET}}+1) \cdot (1-p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\text{NET}}}}{432}}. \end{aligned} \tag{2.32}$$

This concludes the proof of Item (i).

For Item (ii), note that if $\beta = 0$ and $\alpha = 1$, then $\hat{W}_{q\beta}^{>h_0}(\rho) = 0$, and thus $\hat{p}_{q\beta}^{>h_0}(\rho)$ does not contribute to the probabilities. Hence, we obtain in this case

$$\begin{aligned} p_{\mathsf{bad}} &\leq 6e^{-\frac{\delta^2 \cdot q \cdot (\rho - 2\Delta_{\mathrm{NET}} + 1) \cdot (1 - p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\mathrm{NET}}}}{432}} + 2e^{-\frac{\delta^2 \cdot q \cdot \rho \cdot p_{\leq h_0}}{12}} \\ &\leq 8e^{-\frac{\delta^2 \cdot q \cdot (\rho - 2\Delta_{\mathrm{NET}} + 1) \cdot (1 - p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\mathrm{NET}}}}{432}} . \end{aligned} \tag{2.33}$$

□

We now introduce the notion of a *capped-weight-function* to encapsulate the intuition for the properties a useful weight function should have.

Definition 2.4.2 (Capped weight functions)**.** Let w be a weight function, and $T \in \mathcal{H}$. We say that w is T*-capped* if for all $h, h' \in \mathcal{H}$, with $h, h' > T$, we have $\mathsf{w}(h) = \mathsf{w}(h')$.

Using this definition, we consider two special cases of the general common-prefix property: What should be satisfied to ensure a common prefix under the worst-case conditions, and how fast do we achieve a common prefix in the best case where the adversary only controls the network delay?

We next show one way to pick T such that the common-prefix property holds for the particular case where w is T-capped weight function. To this end, we use Lemma 2.4.1 with $h_0 = T$. The specific conditions we achieve are captured by the lemma below.

Lemma 2.4.3. *Let P_1, P_2 be (not necessarily different) honest parties, let $r_1 \leq r_2$ be rounds, let $\epsilon_{\mathsf{c}} := \alpha - \beta > 0$, let $\delta \in (0, 1)$, and let C_1 be the best chain of P_1 in round r_1. Finally let p_{bad} be the probability that P_2 has a best chain C_2 in round r_2 with ${C_1}^{\mathrm{R}>\lceil r_1 - \rho} \not\preceq C_2$. If $\rho > 2\hat{\Delta}_{\mathsf{Net}} - 1$ and w is a T-capped-weight-function that satisfies*

$$T \geq \left(\frac{\beta \cdot \rho}{\left(\beta + \frac{\epsilon_{\mathsf{c}}}{2}\right)(1 - \delta)(\rho - 2\hat{\Delta}_{\mathsf{Net}} + 1)} \right)^{\frac{1}{2q\hat{\Delta}_{\mathsf{Net}}}} \cdot 2^k, \tag{2.34}$$

and

$$\frac{1}{2\hat{\Delta}_{\mathsf{Net}}} \cdot (1 - \delta) \cdot \frac{\epsilon_{\mathsf{c}}}{2} \cdot (1 - p_{\leq T}) \cdot (p_{\leq T})^{2q\hat{\Delta}_{\mathsf{Net}} - 1} \geq \frac{\mathsf{w}_{\max \leq T}}{\mathsf{w}_{\min > T}}, \tag{2.35}$$

then

$$p_{\mathsf{bad}} \leq 10e^{-\frac{\delta^2 \cdot q\beta^2 \cdot \rho \cdot (1 - p_{\leq T})}{432\left(\beta + \frac{\epsilon_{\mathsf{c}}}{2}\right)(1 - \delta)}} . \tag{2.36}$$

Furthermore, if $\rho > 2\Delta_{\mathrm{NET}} - 1$, $\alpha = 1$, $\beta = 0$, and for all $h_0 \leq T$

$$\frac{1}{2\hat{\Delta}_{\mathsf{Net}}} \cdot \frac{(1 - \delta)}{e \cdot 2q\hat{\Delta}_{\mathsf{Net}}} \geq \frac{\mathsf{w}_{\max \leq h_0}}{\mathsf{w}_{\min > h_0}}, \tag{2.37}$$

then

$$p_{\mathsf{bad}} \leq 8e^{-\frac{\delta^2 \cdot q \cdot (\rho - 2\Delta_{\mathrm{NET}} + 1)}{432 \cdot e \cdot (2q\Delta_{\mathrm{NET}} + 1)}} . \tag{2.38}$$

Proof. We note that the condition in Lemma 2.4.1 is implied by the following two conditions:

$$\begin{aligned} &(1-\delta)\cdot\frac{\epsilon_{\mathsf{c}}}{2}\cdot(1-p_{\le T})\cdot(p_{\le T})^{2q\Delta_{\mathrm{NET}}} \ge \frac{\rho}{\rho-2\Delta_{\mathrm{NET}}+1}\cdot\frac{\mathsf{w}_{\max\le T}}{\mathsf{w}_{\min>T}}\cdot p_{\le T} \\ \iff &\frac{\rho-2\Delta_{\mathrm{NET}}+1}{\rho}\cdot(1-\delta)\cdot\frac{\epsilon_{\mathsf{c}}}{2}\cdot(1-p_{\le T})\cdot(p_{\le T})^{2q\Delta_{\mathrm{NET}}-1}\ge\frac{\mathsf{w}_{\max\le T}}{\mathsf{w}_{\min>T}} \end{aligned} \tag{2.39}$$

and

$$\begin{aligned} &(1-\delta)\cdot(\beta+\frac{\epsilon_{\mathsf{c}}}{2})\cdot(1-p_{\le T})\cdot(p_{\le T})^{2q\Delta_{\mathrm{NET}}} \ge \frac{\rho}{\rho-2\Delta_{\mathrm{NET}}+1}\cdot\frac{\mathsf{w}_{\max>T}}{\mathsf{w}_{\min>T}}\cdot\beta\cdot(1-p_{\le T}) \\ \iff &(1-\delta)\cdot(\beta+\frac{\epsilon_{\mathsf{c}}}{2})\cdot(p_{\le T})^{2q\Delta_{\mathrm{NET}}}\ge\frac{\rho}{\rho-2\Delta_{\mathrm{NET}}+1}\cdot\frac{\mathsf{w}_{\max>T}}{\mathsf{w}_{\min>T}}\cdot\beta. \end{aligned} \tag{2.40}$$

It is enough to show Equation (2.39) for the upper bound on the network delay $\hat{\Delta}_{\mathsf{Net}}$, and $\rho = 2\hat{\Delta}_{\mathsf{Net}}$ as $\frac{\rho-2\hat{\Delta}_{\mathsf{Net}}+1}{\rho}$ is monotonously increasing in ρ.

For any T-capped weight function w, we have

$$\min_{h\in\{T+1,\ldots,2^k\}}\mathsf{w}(h) = \max_{h\in\{T+1,\ldots,2^k\}}\mathsf{w}(h). \tag{2.41}$$

Combining this with Equation (2.40) and inserting the upper bound on the network delay derives the following two conditions:

$$\frac{1}{2\hat{\Delta}_{\mathsf{Net}}}\cdot(1-\delta)\cdot\frac{\epsilon_{\mathsf{c}}}{2}\cdot(1-p_{\le T})\cdot(p_{\le T})^{2q\hat{\Delta}_{\mathsf{Net}}-1}\ge\frac{\mathsf{w}_{\max\le T}}{\mathsf{w}_{\min>T}}, \tag{2.42}$$

and

$$\frac{\rho-2\hat{\Delta}_{\mathsf{Net}}+1}{\rho}\cdot(1-\delta)\cdot(p_{\le T})^{2q\hat{\Delta}_{\mathsf{Net}}}\ge\frac{\beta}{\beta+\frac{\epsilon_{\mathsf{c}}}{2}}. \tag{2.43}$$

Fulfilling these two equations will give us common prefix except with the probability stated in Lemma 2.4.1. One way to satisfy Equation (2.43) is to derive a condition for picking T. Recall that $p_{\le T} = \frac{T}{2^k}$. Hence, the condition is satisfied for

$$T \ge \left(\frac{\beta\cdot\rho}{\left(\beta+\frac{\epsilon_{\mathsf{c}}}{2}\right)(1-\delta)(\rho-2\hat{\Delta}_{\mathsf{Net}}+1)}\right)^{\frac{1}{2q\hat{\Delta}_{\mathsf{Net}}}}\cdot 2^k. \tag{2.44}$$

If a T-capped-weight-function satisfies these two conditions it provides common prefix except with the probability given by Lemma 2.4.1 (i). Using Equation (2.43), this can be simplified to

$$p_{\mathsf{bad}} \le 10e^{-\frac{\delta^2\cdot q\beta^2\cdot\rho\cdot(1-p_{\le T})}{432(\beta+\frac{\epsilon_{\mathsf{c}}}{2})(1-\delta)}}, \tag{2.45}$$

for any sufficiently large ρ.

We now consider the case when all parties are honest and analyze what conditions need to be satisfied for getting common prefix. Let w be a weight-function. The condition in Lemma 2.4.1, when instantiated with $\alpha = 1$ and $\beta = 0$, becomes

$$(1-\delta)\cdot(1-p_{\le h_0})\cdot(p_{\le h_0})^{2q\Delta_{\mathrm{NET}}-1}\ge\frac{\rho}{\rho-2\Delta_{\mathrm{NET}}+1}\cdot\frac{\mathsf{w}_{\max\le h_0}}{\mathsf{w}_{\min>h_0}}$$

$$\iff$$

$$\frac{\rho - 2\Delta_{\text{NET}} + 1}{\rho} \cdot (1-\delta) \cdot (1 - p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\text{NET}}-1} \geq \frac{\mathsf{w}_{\max \leq h_0}}{\mathsf{w}_{\min > h_0}}. \tag{2.46}$$

Let $\xi := 2q\Delta_{\text{NET}}$. We pick h_0 such that $(1 - p_{\leq h_0}) \cdot (p_{\leq h_0})^{\xi}$ is maximized (as this occurs in the probability p_{bad} of Lemma 2.4.1 (ii)), which is the case for

$$p_{\leq h_0} = \frac{\xi}{\xi+1} = \frac{2q\Delta_{\text{NET}}}{2q\Delta_{\text{NET}}+1} \quad \iff \quad h_0 = 2^k \left(\frac{2q\Delta_{\text{NET}}}{2q\Delta_{\text{NET}}+1} \right).^{9} \tag{2.47}$$

For this particular choice of h_0 we note that

$$(1 - p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\text{NET}}} = \frac{\left(\frac{\xi}{\xi+1}\right)^{\xi}}{\xi+1} = \frac{\frac{1}{\left(1+\frac{1}{\xi}\right)^{\xi}}}{\xi+1} \geq \frac{1}{(\xi+1)\cdot e}. \tag{2.48}$$

Hence, Equation (2.46) is satisfied if

$$\frac{\rho - 2\Delta_{\text{NET}} + 1}{\rho} \cdot \frac{(1-\delta)}{(\xi+1)e \cdot p_{\leq h_0}} = \frac{\rho - 2\Delta_{\text{NET}} + 1}{\rho} \cdot \frac{(1-\delta)}{e \cdot \xi} \geq \frac{\mathsf{w}_{\max \leq h_0}}{\mathsf{w}_{\min > h_0}}. \tag{2.49}$$

Note that $\frac{\rho - 2\Delta_{\text{NET}}+1}{\rho}$ and $\frac{1}{e \cdot \xi}$ are monotonously decreasing in $\Delta_{\text{NET}} \leq \hat{\Delta}_{\mathsf{Net}}$, and $\frac{\rho - 2\Delta_{\text{NET}}+1}{\rho}$ is monotonously increasing in $\rho \geq 2\Delta_{\text{NET}}$. This implies that it is sufficient to satisfy this equation for $\Delta_{\text{NET}} = \hat{\Delta}_{\mathsf{Net}}$ and $\rho = 2\hat{\Delta}_{\mathsf{Net}}$:

$$\frac{1}{2\hat{\Delta}_{\mathsf{Net}}} \cdot \frac{(1-\delta)}{e \cdot 2q\hat{\Delta}_{\mathsf{Net}}} \geq \frac{\mathsf{w}_{\max \leq h_0}}{\mathsf{w}_{\min > h_0}}. \tag{2.50}$$

This condition can be satisfied by again making the weight function grow fast enough such that $\frac{\mathsf{w}_{\max \leq h_0}}{\mathsf{w}_{\min > h_0}}$ is sufficiently small for all $h_0 \leq T$. If Equation (2.50) is satisfied then we obtain by Lemma 2.4.1 (ii) and using Equation (2.48) that the probability of a common-prefix violation is at most

$$p_{\mathsf{bad}} \leq 8e^{-\frac{\delta^2 \cdot q \cdot (\rho - 2\Delta_{\text{NET}}+1) \cdot (1 - p_{\leq h_0}) \cdot (p_{\leq h_0})^{2q\Delta_{\text{NET}}}}{432}} \leq 8e^{-\frac{\delta^2 \cdot q \cdot (\rho - 2\Delta_{\text{NET}}+1)}{432 \cdot e \cdot (2q\Delta_{\text{NET}}+1)}}. \tag{2.51}$$

□

2.4.1.1 Choosing a Weight Function

We suggest the following approach to instantiate a T-capped weight function. First, pick T such that it satisfies Equation (2.34) for a sufficiently large ρ. Next, select the function such that it additionally ensures the condition from Equation (2.35). For monotone functions, this can be done by increasing the growth of the function such that $\frac{\mathsf{w}_{\max \leq T}}{\mathsf{w}_{\min > T}}$ is sufficiently small. When a T-capped weight function is instantiated like this, it provides a common prefix except with the probability given by Equation (2.36). To

[9] As h_0 needs to be in $\mathcal{H}$, it can most likely not be set to exactly this value. Instead one can choose it as $h_0 = \left\lceil 2^k \frac{\xi}{\xi+1} \right\rceil$, which ensures that $\frac{\xi}{\xi+1} \leq p_{\leq h_0} \leq \frac{\xi}{\xi+1} + \frac{1}{2^k}$. This does not influence the conclusion and we ignore this for ease of presentation.

further satisfy Equation (2.37), one can additionally increase the growth of the function until it is true for all $h_0 \leq T$.[10]

Waiting time for common prefix. To ensure that parties are on a common prefix except with negligible probability, one has to wait until p_{bad} is negligible. If κ is the security parameter, this means that one has to wait for ρ rounds such that $\rho \cdot q(1-p_{\leq T}) = \Omega(\kappa)$. Note that $q(1-p_{\leq T})$ is the expected number of blocks with hash above the threshold T produced in each round. This means one needs to wait for $\Omega(\kappa)$ blocks above the threshold. This matches the bounds derived for the plain Bitcoin backbone, e.g., in [GKL15].

In the case without corruption, one has to wait ρ rounds such that $\rho \cdot \frac{1}{\Delta_{\mathrm{NET}}} = \Omega(\kappa)$. Note that this only depends on Δ_{NET}, not on $\hat{\Delta}_{\mathsf{Net}}$. Hence, the protocol is responsive in this case!

Chain growth and chain quality. Note that this approach automatically ensures some chain growth and chain quality as the preconditions for Theorem 2.3.4 and Theorem 2.3.6 are weaker than the precondition for Theorem 2.3.12. One can also obtain tighter bounds by optimizing for this, but we leave that for future work.

Finality layers. A practical issue of the responsiveness provided by Lemma 2.4.3 is that it is hard to know whether there are actively corrupted nodes. This means that even in the good case without corruption, where all parties quickly agree on blocks, parties typically do not know for sure that there is no corruption and thus cannot confirm transactions rapidly. As a solution to this issue, we propose to use a finality layer, such as Casper the Friendly Finality Gadget [BG17], GRANDPA [SK20], or Afgjort [Din+20]. These act as an additional layer on top of an NSB, where a committee votes on blocks to become final, and finalized blocks are never rolled back by adjusting the chain-selection rule to prefer chains with more finalized blocks. In such finality layers, a block can be declared final as soon as enough committee members vote for that block. In the optimistic case, this happens as fast as the actual network conditions allow in our responsive blockchain, as all honest parties will, in fact, have the same common prefix and thus vote for the same. And given the decision from the finalization committee, one can immediately trust these finalized blocks, yielding a high overall efficiency.

2.4.2 Examples of Capped Weight Functions

In this section, we provide two concrete instantiations of weight functions using our framework. For means of comparison, we first instantiate the standard Bitcoin weight function and afterward a capped-exponential weight function, which we compare to the Bitcoin protocol. See Figure 2.1 for plots of the considered weight functions.

[10] In our analysis, we need to set $\frac{\mathsf{w}_{\max \leq h_0}}{\mathsf{w}_{\min > h_0}}$ sufficiently small to satisfy both Equations (2.35) and (2.37). Note that no condition places a lower bound on this fraction. This means the weight function can be chosen to grow arbitrarily fast.

The trade-off hidden in our analysis is that faster-growing functions lead to less responsiveness if there is some corruption. That is because it becomes easier to produce very heavy blocks that can roll back many lighter blocks. The growth of the function should thus not be set higher than necessary. We leave exploring this trade-off for future work.

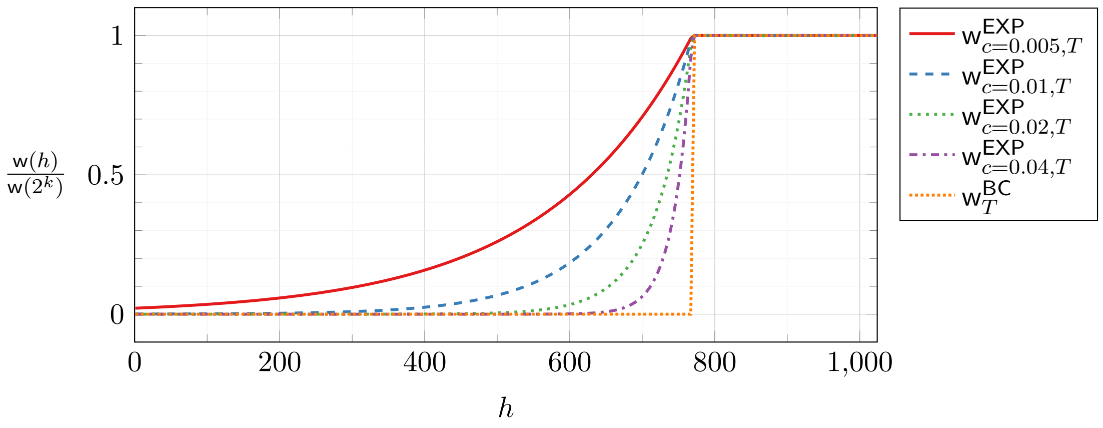


Figure 2.1: Plots of $\mathsf{w}^{\mathsf{BC}}_T$ and $\mathsf{w}^{\mathsf{EXP}}_{c,T}$ normalized with the maximal weight for $k = 10$, $T = \frac{3}{4} \cdot 2^k$, and different values of c. The values are chosen very small for illustrative purposes. Note that the larger c is, the closer the form of $\mathsf{w}^{\mathsf{EXP}}_{c,T}$ is to $\mathsf{w}^{\mathsf{BC}}_T$. This plot depicts the intuition that c can be picked so large that there is no security degradation by choosing $\mathsf{w}^{\mathsf{EXP}}_{c,T}$ over $\mathsf{w}^{\mathsf{BC}}_T$, even though an adversary can potentially control the honest weight produced below T through network delays.

2.4.2.1 Bitcoin Weight

The Bitcoin protocol originally considered the best chain to be the one that is the longest. Each block added to a chain can therefore be considered incrementing the chain's weight with 1. If a block is invalid, it does not change the weight of a chain, and it can thus be thought of as having weight 0. With this interpretation, the Bitcoin weight function with threshold T can be defined as[11]

$$\mathsf{w}^{\mathsf{BC}}_T(h) \triangleq \begin{cases} 0, & \text{if } h \leq T, \\ 1, & \text{else.} \end{cases} \tag{2.52}$$

This is clearly an instance of a T-capped-weight-function. Thus, the approach from Section 2.4.1 can be applied for picking T, i.e., set T such that Equation (2.34) is an equality.

For $\mathsf{w} = \mathsf{w}^{\mathsf{BC}}_T$, we have $\mathsf{w}_{\min>T} = 1$ and $\mathsf{w}_{\max\leq T} = 0$. Hence, Equation (2.35) is trivially satisfied, and Equation (2.36) thus provides the probability bound for the common prefix violations. This matches known bounds as explained in Section 2.4.1.

There only exists a single h_0 such that Equation (2.37) is satisfied, namely $h_0 = T$. This matches well with the intuition: Bitcoin is not reactive as T needs to be set based on the *worst case network delay* to ensure security.

2.4.2.2 Capped Exponential Weight

We now provide an example weight function that can be instantiated to obtain an optimistically responsive protocol. For some parameter $c \in \mathbb{R}$ and a threshold $T \in \mathcal{H}$,

[11]To adapt to our framework, we negate the condition on the valid block predicate. Note that this is without loss of generality.

we define

$$\mathsf{w}^{\mathsf{EXP}}_{c,T}(h) \triangleq \begin{cases} e^{hc}, & \text{if } h \leq T, \\ e^{(T+1)c}, & \text{else.} \end{cases} \tag{2.53}$$

Let $h \in \mathcal{H}$, $h \leq T$. We then have for $\mathsf{w} = \mathsf{w}^{\mathsf{EXP}}_{c,T}$,

$$\frac{\mathsf{w}_{\max \leq h}}{\mathsf{w}_{\min > h}} = \frac{\mathsf{w}^{\mathsf{EXP}}_{c,T}(h)}{\mathsf{w}^{\mathsf{EXP}}_{c,T}(h+1)} = \frac{e^{hc}}{e^{(h+1)c}} = e^{-c}. \tag{2.54}$$

Again we pick T such that Equation (2.34) is an equality. We now pick c such that both Equations (2.35) and (2.37) are satisfied for all h_0. In other words, we pick c such that both

$$e^{-c} = \frac{\mathsf{w}_{\max \leq T}}{\mathsf{w}_{\min > T}} \leq \frac{1}{2\hat{\Delta}_{\mathsf{Net}}} \cdot (1-\delta) \cdot \frac{\epsilon_{\mathsf{c}}}{2} \cdot (1 - p_{\leq T}) \cdot (p_{\leq T})^{2q\hat{\Delta}_{\mathsf{Net}}-1}, \tag{2.55}$$

and

$$e^{-c} = \frac{\mathsf{w}_{\max \leq h_0}}{\mathsf{w}_{\min > h_0}} \leq \frac{1}{2\hat{\Delta}_{\mathsf{Net}}} \cdot \frac{(1-\delta)}{e \cdot 2q\hat{\Delta}_{\mathsf{Net}}}, \tag{2.56}$$

are satisfied. Such a c exists as both right-hand sides are constant and e^{-c} drops exponentially in c. Instantiating w in this way provides a protocol that, under worst-case conditions, performs as the Bitcoin protocol but, in good conditions, is perfectly responsive to the actual network delay.

3 Formalizing Delayed Adaptive Corruptions and the Security of Flooding Networks

3.1 Introduction

3.1.1 Motivation

In *Nakamoto-style blockchains* (NSBs) such as Bitcoin [Nak08], several parties continuously try to solve cryptographic puzzles. The first party solving the puzzle “wins” the right to create a new block extending the previously longest chain. This block is then distributed to all other parties, who continue solving puzzles to create the next block. Extensive research has shown for different variations of NSBs that security can be guaranteed if honest parties solve a majority of the puzzles and if blocks can be propagated fast enough to ensure with a high probability that the next winner has learned about the previous block before creating a new block [GKL15; GKL17; PSS17; Ren19].

Since future block creators are unpredictable, these protocols have a high resilience against adaptive corruptions. Intuitively, the only chance to exploit the adaptivity of corruptions is to corrupt a party after learning that it has solved a puzzle and subsequently prevent this party from distributing the created block. An adversary with the power to stop messages from being delivered (or changing the message) by corrupting the sender after sending but before the message is delivered is often referred to as *strongly adaptive* [Abr+19]. On the other hand, if messages from honest senders are guaranteed to be delivered regardless of whether the sender gets corrupted before delivery, the adversary is only *weakly adaptive*, or equivalently, *atomic message send* (AMS) [Gar+11] is assumed.

Indeed, several papers [GKL15; GKL17; PSS17] have proven the security of Bitcoin’s consensus against adaptive corruptions, and Ouroboros Praos [Dav+18] has been developed as a proof-of-stake blockchain with resilience against fully adaptive corruptions as one of the main selling points. To achieve this, these papers have to assume atomic message dissemination. In reality, however, NSBs typically use complex peer-to-peer networks to disseminate blocks, in which each party propagates messages to only a small set of other parties (referred to as their neighbors), who will then propagate it to their neighbors and so forth. Even if the point-to-point channels between neighbors allow atomic sends, the overall network will not provide this guarantee because an adaptive adversary can simply corrupt all neighbors of the sender and thereby stop the block from being propagated. Hence, when considering the full protocol, which combines a NSB with a peer-to-peer flooding network, security against fully adaptive corruptions can no longer be guaranteed.

Formalizing delayed adaptive corruptions. To provide meaningful guarantees to blockchain protocols, including their peer-to-peer network, we observe that intuitively, one needs to restrict the *corruption speed* of an adversary such that parties in the peer-to-peer network have enough time to pass on the block they receive before being corrupted. Based on this observation, we introduce a precise model for δ*-delayed adversaries* in the Universal Composability framework [Can20]. Using this model, one can quantify the minimum amount of time δ it takes from when an adversary targets and starts attacking a specific party until this party is actually under adversarial control and prove the security of protocols against such corruptions. This allows us to precisely describe what kind of adversaries different P2P networks and protocols built on top can withstand.

Note that the corruption speed of an adaptive adversary also has a natural translation to reality. For an attacker to succeed in attacking some physical machine, it necessarily takes some time from targeting the device to actually hack into the network (by either physical or digital means) and take over the computer. Denial-of-service attacks are arguably faster to mount, but it still takes nonzero time to target a specific machine.

While unstructured peer-to-peer networks for message dissemination are the main focus of this chapter, delayed adversaries have much broader applications and were, in fact, already used in other works with varying degrees of formality. For example, the original Ouroboros [Kia+17], in contrast to its successor Ouroboros Praos [Dav+18], which only requires AMS, needs that corruptions are sufficiently delayed. The same is true for Snow White [DPS19], another early proof-of-stake blockchain. Another example is Hybrid Consensus [PS17b], which periodically elects committees using a blockchain and remains secure if corruptions are delayed until the next committee is selected. The same applies to blockchain sharding proposals [Kok+18; Luu+16; ZMR18] in which the members of shards are periodically chosen.

Concrete analysis of flooding networks. As mentioned above, the security of NSBs crucially relies on the assumption that blocks are, with high probability, propagated to other parties before the next winner creates another new block. If an upper bound on the propagation time is known, the difficulty of the puzzles can be set accordingly to provide this guarantee. On the other hand, setting the difficulty based on a too optimistic assumption about the delay jeopardizes the system's security, and setting it based on a too loose upper bound degrades efficiency. Knowing a tight bound on the propagation delay is thus crucial for the security and efficiency of an NSB.

Even more critical for the security of NSBs are so-called eclipse attacks that prevent some parties from receiving blocks [Hei+15; MHG18]. Furthermore, for large-scale distributed systems, the number of neighbors has a significant impact on the required communication. In particular, it is infeasible to send the message directly to everybody. In this work, we provide constructions for flooding networks with provable security against eclipse attacks in a well-defined adversarial model and show different trade-offs between the propagation time and neighborhood sizes.

Terminology. In the literature, different terminology has been used for the process of disseminating a message to all parties. Common terminology includes "broadcast", "flood" and "multicast". We will use the terminology "flood" for this process. Contrary to *byzantine broadcast*, there is no agreement requirement for a flooding network if the sender of a message is dishonest.

3.1.2 Contributions and Results

Our contributions are twofold:

1. We give precise semantics to δ-*delayed corruptions* (introduced in [PS17b] as δ-agile corruptions) within the UC framework [Can20]. We define the semantics via corruption shells which allows us to prove how these corruptions relate to standard adaptive corruptions.

2. We define a functionality for disseminating information, Flood, that can be used to implement a secure NSB, and that we implement using a flooding protocol against a *slightly delayed* adversary. Importantly, we quantify precisely how much is meant by "slightly" in terms of guarantees provided by the underlying point-to-point channels. We provide two instantiations of our protocol with different efficiency trade-offs.

Below we lay out the specifics of the individual contributions and state our results in more detail.

Precise model for δ-delayed adversaries. We define a δ-delayed adversary as an adversary who uses at least δ time to perform a corruption. We define this notion precisely within the UC framework using the notion of time from [Bau+21]. We do so by elaborating on the notion of corruption shells from [Can20].

Using the idea of corruption shells, we give semantics to both "normal" byzantine adaptive corruption and δ-delayed corruptions. We capture the semantics of byzantine adaptive corruptions in a corruption shell, $\mathcal{B}_{\mathsf{Real}}$, for protocols and in a corruption shell, $\mathcal{B}_{\mathsf{Ideal}}$, for ideal functionalities. Similarly, we capture the semantics of δ-delayed adversaries in a corruption-shell, $\mathcal{D}^{\delta}_{\mathsf{Real}}$, for protocols and in a corruption-shell, $\mathcal{D}^{\delta}_{\mathsf{Ideal}}$, for ideal functionalities.

$\mathcal{D}^{\delta}_{\mathsf{Real}}$ and $\mathcal{D}^{\delta}_{\mathsf{Ideal}}$ accepts two inputs: *Precorrupt* and *Corrupt* (both indexed by a specific party). Both shells ensure that at least δ time has passed after receiving *Precorrupt* before reacting upon *Corrupt*. Any *Corrupt* input that is sent prematurely is ignored.

Having defined the semantics for both standard adaptive corruptions and δ-delayed corruptions using corruption shells, we state basic results relating the two models. We show that a protocol is secure against a standard adaptive adversary *iff* it is secure against a 0-delayed adversary (Theorem 3.3.2). Furthermore, we show that if a protocol is secure against a "fast" adversary, it implies that it is also secure against a "slow" adversary (Theorem 3.3.3). Together these results allow constructions proven secure in the standard model of adaptive adversaries to be reused when constructing new protocols secure against a δ-delayed adversary and to compose protocols that are secure against adversaries with different delays.

Flooding networks. We define a functionality for flooding messages, $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$. It ensures that all parties learn messages that an honest party has sent or has received within

Δ time and is thereby similar to the flooding functionality assumed in many consensus protocols. We realize our flooding functionality with a naive protocol, $\Pi_{\mathsf{NaiveFlood}}$, where everybody sends to everybody, and a more advanced protocol, $\Pi_{\mathsf{ERFlood}}(\rho)$, where all parties choose to send to other parties with probability ρ.

To realize the flooding functionality, we introduce a functionality for a point-to-point channel $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}$. This functionality is also parameterized by a bound for the delivery time Δ and additionally has a parameter σ describing the time an honest party needs to stay honest after starting to send the message for the delivery guarantee to apply. If $\sigma = 0$, this corresponds to assuming AMS. On the other hand, if $\sigma \geq \delta$ and we consider δ-delayed adversaries, then this corresponds to not assuming AMS. However, having the time quantified allows us to relate this time to the delay we can tolerate when building more advanced constructions. In particular, we show that Π_{ERFlood} using $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}$ implements $\mathcal{F}^{\Delta'}_{\mathsf{Flood}}$ against a $(\sigma + \Delta)$-delayed adversary.

In this setting, we provide two different ways to instantiate the probability parameter ρ of Π_{ERFlood}, each presenting a different efficiency trade-off. Concretely, let h denote the minimum number of parties that will stay honest throughout the execution of the protocol, let n denote the total number of parties, and let κ be the security parameter. We provide the following two instantiations:

Instantiation 1: Guaranteed delivery within $\Delta' := 2 \cdot \Delta$ for $\rho := \sqrt{\frac{\kappa+\log(n)}{h}}$.

Instantiation 2: Guaranteed delivery within $\Delta' := \Delta \cdot \left(5\log\left(\frac{n}{2\kappa}\right) + 2\right)$ for $\rho := \frac{\kappa+\log(n)}{h}$.

Both instantiations ensure that the statistical distance between the ideal and the real executions of Π_{ERFlood} and $\mathcal{F}^{\Delta'}_{\mathsf{Flood}}$ is negligible in the security parameter. We provide concrete bounds for the statistical distance in Corollary 3.6.3. Furthermore, standard probability bounds ensure that each instantiation has a neighborhood of $O(n \cdot \rho)$ with high probability.

Outline of the chapter. In the remainder of this section, we review some selected related work and provide a high-level overview of the techniques used to prove our technical results. In Section 3.2, we recap some basic definitions for random graphs, provide a brief overview of the elements from UC used in this work, and introduce some basic notation. In Section 3.3, we introduce our new model for δ-delayed adversaries, in Section 3.4 we prove concrete bounds for the diameter of Erdős–Rényi graphs, and in Section 3.5 present our ideal functionalities. Finally, in Section 3.6, we present our two implementations of the flooding functionality.

Acknowledgements. We thank Ran Canetti for explaining a subtle detail of the UC framework, Sabine Oechsner for discussions in the initial phase of the project, and the anonymous reviewers of Eurocrypt and Crypto for their feedback.

3.1.3 Techniques

An Erdős–Rényi graph [ER60] is a graph where each edge appears with an equal and independent probability. Our flooding protocol Π_{ERFlood} is strongly inspired by this type of graph. Our main technical contributions are thus concerned with transporting bounds for Erdős–Rényi graphs to the cryptographic setting, especially in the presence of adaptive adversaries.

Concrete bounds for Erdős–Rényi graphs. The asymptotic behavior of Erdős–Rényi graphs has been thoroughly studied in the literature (for a comprehensive overview, see [Bol01]).

However, bounds about a graph's behavior when the amount of nodes goes toward infinity are of little use for protocols that are supposed to be run by a finite number of parties. For a protocol imitating the behavior of such graphs, we need concrete bounds when a security parameter is increased. As a technical contribution, we prove such concrete upper bounds for the diameter of Erdős–Rényi graphs. The upper bounds can be found in Section 3.4.

Applying Erdős–Rényi graph results in the presence of adaptive adversaries. For a flooding protocol as Π_{ERFlood}, it is straightforward to apply bounds about the diameter of an Erdős–Rényi to also bound the probability that a message is not delivered in the protocol in the presence of a *static* adversary. However, for an adaptive adversary capable of preventing certain nodes from connecting to their neighbors, it is by no means this easy. Our main technical contribution is to transfer the bounds on the diameter of an Erdős–Rényi graph to our flooding protocol in the presence of an adaptive adversary. We achieve this by relating the protocol execution to 7 random experiments.

First, we relate the protocol execution to a well-defined game between an adversary and an oracle, which returns a graph. The game's rules are that an adversary can query the oracle to reveal the edges of a node and query the oracle to remove a node from the graph. However, once either an incoming or outgoing edge to a node has been revealed, the adversary can no longer remove this node. This game mimics the powers of a slightly delayed adaptive adversary in the real protocol.

We relate this game to a similar game with undirected edges and do a couple of simple game hops where we show that an adversary does not gain any additional advantage w.r.t. increasing the diameter by stopping this game at an early point nor injecting any extra edges.

As the adversary can only remove nodes for which no information has been revealed, one might be led to believe that the Erdős–Rényi graph results apply to this game. However, the adversary can still dynamically control the size of the graph that is returned. At first, this may seem innocent, but it is not. Deciding whether or not more nodes are to be included in the graph can amplify the probability that the returned graph has a high diameter.

Therefore, we relate this game to a new one similar to the other, except that the oracle, now at random, fixes the graph size beforehand. The oracle fixes the size of the graph by making a uniform guess in the range of possible sizes. In case of a correct guess (a guess identical to what the adversary anyway would end up with), the adversary is only left with the choice of which parties to include in the random graph. Finally, we show that this game is equally distributed to a game that explicitly embeds an Erdős–Rényi graph of the fixed size. This allows us to apply the results bounding diameter of Erdős–Rényi graphs to the probability that a message is not delivered timely.

3.1.4 Related Work

Hybrid consensus. Hybrid Consensus [PS17b] is a recent consensus protocol that uses a blockchain to periodically select committees as subsets of the parties participating in the blockchain protocol, who can subsequently produce blocks more efficiently. Once a committee has been chosen, a fully adaptive adversary can corrupt the majority of its members to break the protocol's security. Hence, the protocol is only secure against corruptions that are delayed until the next committee gets selected.

To prove the security of hybrid consensus, that paper introduces *τ-agile corruptions*, which correspond to the capabilities of our τ-delayed adversaries. While that paper also uses the UC framework, the definitions for the corruption model mostly remain at a high level. For example, their definitions assume there is some notion of time, which does not exist in the original UC formalism. There are also no clear definitions of how the delayed corruptions are precisely embedded in the UC execution model.

In contrast, our work provides a precise embedding of the corruption model in the standard UC framework. This allows us to compose protocols formulated in standard UC with protocols proven secure against δ-delayed adversaries. It is thus fair to say that the hybrid consensus paper has introduced the delayed corruption model at an intuitive, semi-formal level, while our work fills in several missing technical details to provide a precise formalization within the UC framework.

Time in UC. There have been several suggestions for modeling time in UC.[Kat+13] models time using a clock functionality that is local to each protocol. This functionality synchronizes the parties by only allowing the adversary to advance the time when all parties have reported that they have been activated. As this is a *local* functionality, other ideal functionalities have no access to it and therefore need to provide their own notion of time, which can clutter the final guarantees from the functionality.

[KZZ16] takes a similar approach to Katz et al. but changes the clock to be a *global functionality* in GUC [Can+07]. This enables several different protocols to rely on the same notion of time when composed and also solves the problem of time not being available to ideal functionalities. Both functionalities and parties can query the global clock for the current time and thus inherently makes any protocol modeled with this a synchronous protocol.

A different approach is taken in [Bau+21]. They take the standpoint that parties should be oblivious to the passing of time. To allow this, they introduce a global functionality, dubbed a *ticker* (written $\bar{\mathcal{G}}_{\text{Ticker}}$), which exposes an interface to learn about the passing of time to functionalities *only*. In particular honest parties are oblivious to the passing of time. This allows time to be modeled without having synchrony as an inherent assumption. The specific timing assumptions can then be captured by adding an extra ideal functionality that exposes relevant information to the parties.

Contrary to [Bau+21; Kat+13; KZZ16], [Can+17] focuses on modeling and making *real* time available to parties in GUC and use this to model the expiration of certificates in a public-key infrastructure. In their modeling, the environment can advance a global clock without restrictions. In this work, our protocols do not rely on real-time but rather on an abstract notion of time used to state assumptions and guarantees about the delivery time of channels and protocols. For the guarantees to be upheld, we rely on restrictions about how time is advanced (namely that all parties must be activated once each abstract time step) by the environment.

We chose to rely on the modeling of time from [Bau+21]. This allows us to model general timing assumptions on the adversary's capabilities without tying our modeling to a particular assumption on synchrony for actual protocols.

Epidemic and gossip protocols. Epidemic algorithms or gossip protocols were first considered for data dissemination by Demers et al. [Dem+87], and have been studied extensively since then, see, e.g., [Bir+99; Cri+09; HHL06; Hu+12; Kar+00; KMG03]. In this line of work, many different protocols have been considered. Some are very closely related to our flooding protocol, where parties forward to a random set of parties, and some are more advanced, letting parties keep sending to new random peers until a certain number of recipients reply that they already knew the message. However, this line of work considers only random failures [KMG03] or incomplete network topologies [Cri+09; Hu+12] and not adaptive corruptions of a malicious adversary. Hence, while some of the protocols apply to our setting, their analysis is not. Among other results, [KMG03] showed how random node failures affect the success probability of a flooding process similar to ours. For this setting, they derive connectivity bounds similar to the bounds for logarithmic diameter we present in this work, but only for $n \to \infty$.

Kadcast. Kadcast [RT19] is a structured peer-to-peer network for blockchains. The paper claims that unstructured networks are inherently inefficient because many superfluous messages are sent to parties who have already received the message from other peers. They instead propose a structured network based on Kademlia [MM02], in which every node has $O(\log n)$ neighbors, and the diameter of the graph is also $O(\log n)$. Additionally, their protocol includes a parameter for controlling the redundancy and, thus, the resistance to attacks. Due to its structured nature, the suggested network is, however, not secure against adaptive corruption of any kind.

The hidden graph model. Chandran et al. [Cha+15] consider *communication locality* of multi-party computation (MPC) protocols, which corresponds to the maximal number of parties each honest party needs to interact with. They construct an MPC protocol with a poly-logarithmic communication locality that is secure against adaptive corruptions and that runs in a poly-logarithmic number of rounds. Their protocol uses a random communication graph, similar to our flooding protocols. However, to be secure against adaptive corruptions, they need to assume that the communication graph between honest parties remains hidden, i.e., they allow honest parties to communicate securely without an adversary learning who is communicating with whom. Furthermore, they only lose bounds on the locality and diameter of the obtained graph by showing that both are poly-logarithmic.

Message dissemination relying on resource assumptions. Recently, the problem of disseminating messages assuming a constant fraction of honest resources (computational power, stake, etc.) instead of assuming a constant fraction of honest parties (as assumed in this work) has received attention. Extending on results from this work, [Liu+22b] provides an efficient flooding protocol relying on a constant fraction of the resources behaving honestly. Their protocols achieve an asymptotic efficiency similar to the protocol presented in this work. [Cor+22] presents a block dissemination protocol for the Ouroboros Praos protocol [Dav+18] that also relies on the majority of honest stake assumption. By using long-lived connections between parties, they prevent a specific denial-of-service attack possible in the protocol. However, this comes at the cost of allowing a small fraction of honest parties to be eclipsed.

3.2 Preliminaries

3.2.1 Notation

When describing functionalities, we let $\mathcal{P}$ be a set of unique party identifiers (PIDs) and will leave out session identifiers for clarity of presentation.

As a convention, we use the variable $t \in \mathbb{N}$ to denote the maximal number of parties an adversary can corrupt, use the variable $n \coloneqq |\mathcal{P}|$ to represent the total number of parties in a protocol (except when we state and prove general results about graphs) and $h \coloneqq n - t$ to denote the minimal number of honest parties. Whenever we refer to *honest* parties, we will refer to parties that have not received any precorrupt or corrupt tokens.

3.2.2 Graphs

In this section, we briefly recap basic graph concepts and derive several concrete bounds for the diameter of Erdős–Rényi graphs.

3.2.2.1 Basic Definitions

We start by defining undirected graphs, a node's neighborhood, reachables, and degree.

Definition 3.2.1 (Undirected graph)**.** An undirected graph consists of a set of vertices $\mathcal{V}$ and a set of edges $E \subseteq \mathcal{V} \times \mathcal{V}$. For two vertices $v, u \in V$, we interpret the edge $\{v, u\}$ as an undirected edge from v to u. For a G which consist of the nodes $\mathcal{V}$ and the edges E we write $G = (\mathcal{V}, E)$.

Definition 3.2.2 (Neighbors, reachables and degrees)**.** For a graph $G = (\mathcal{V}, E)$ we define the neighborhood of a node $v \in \mathcal{V}$ to be

$$\Gamma(v) \triangleq \{u \mid \{v, u\} \in E\} \cup \{v\}.$$

We overload the notation to also work with any subset of vertices $S \subseteq \mathcal{V}$ and write

$$\Gamma(S) \triangleq \bigcup_{v \in S} \Gamma(v).$$

Applying the function Γ i times yields Γ^i, the set of vertices that can be reached from v in at most i steps,

$$\Gamma^0(v) \triangleq \{v\} \quad \text{and} \quad \Gamma^{i+1}(v) \triangleq \Gamma(\Gamma^i(v)).$$

We further define the set of nodes that can be reached from a node $v \in \mathcal{V}$ in exactly i steps recursively as

$$\theta(v, 0) \triangleq \{v\}$$

$$\theta(v, i+1) \triangleq \Gamma^{i+1}(v) \setminus \Gamma^i(v).$$

We finally define the degree $v \in \mathcal{V}$ as

$$\texttt{deg}(v) \triangleq |\Gamma(v)| - 1.$$

Remark 3.2.1. Note that $\Gamma^t(v) = \bigcup_{i=0}^t \theta(v, i)$.

We also define some of the notions for directed graphs.

Definition 3.2.3 (Directed graph)**.** A directed graph (digraph) consists of a set of vertices $\mathcal{V}$ and a set of edges $E \subseteq \mathcal{V} \times \mathcal{V}$. For two vertices $v, u \in \mathcal{V}$, we interpret the edge (v, u) as a directed edge from v to u. For a digraph G with nodes $\mathcal{V}$ and edges E we write $G = (\mathcal{V}, E)$.

We will use the same symbol Γ to denote the neighbors of a node or a subset of nodes for both directed and undirected graphs. When we use the function, it will be clear from the context if the graph is directed or undirected.

Definition 3.2.4 (Neighbors for directed graphs)**.** For a digraph $G = (\mathcal{V}, E)$ we define the outgoing neighborhood of a node $v \in \mathcal{V}$ to be

$$\Gamma(v) \triangleq \{u \mid (v, u) \in E\} \cup \{v\}.$$

We overload the notation to also work with any subset of vertices $S \subseteq \mathcal{V}$ similarly to how it is done for undirected graphs.

Furthermore, we define the distance between two nodes for both directed and directed graphs.

Definition 3.2.5 (Distance between nodes)**.** Let $G = (\mathcal{V}, E)$ be a directed or undirected graph and let $v_1, v_2 \in \mathcal{V}$. We define the distance from v_1 to v_2, $\texttt{dist}(v_1, v_2)$ as the minimum λ such that $v_2 \in \Gamma^\lambda(v_1)$. If no such λ exists, we define $\texttt{dist}(v_1, v_2)$ to be ∞.

Finally, we define two properties of graphs which we will use extensively to prove our results.

Definition 3.2.6 (Distance from node)**.** Let G be a directed or undirected graph, v be a node in the graph, and $\lambda \in \mathbb{N}$ be a distance. The property $\phi_{\text{Dist}}(v, G, d)$ decides if all nodes in G are within distance d of v. Formally,

$$\phi_{\text{Dist}}(v, G, \lambda) \triangleq \forall v' \in G, \texttt{dist}(v, v') \leq \lambda.$$

Definition 3.2.7 (Diameter)**.** Let $G = (\mathcal{V}, E)$ be a graph. We define the diameter of a graph to be the smallest $\lambda \in \mathbb{N}$ s.t. $\forall v_1, v_2 \in \mathcal{V}$ we have that $\texttt{dist}(v_1, v_2) \leq \lambda$. We define the property that a graph has a diameter at most λ as

$$\phi_{\text{Diam}}(G, \lambda) \triangleq \forall v \in \mathcal{V}, \phi_{\text{Dist}}(v, G, \lambda).$$

3.2.2.2 Erdős–Rényi Graphs

A particular type of random graphs where the edges are chosen from a uniform distribution is called Erdős–Rényi graphs.

Definition 3.2.8 (Erdős–Rényi graphs)**.** An Erdős–Rényi graph is an undirected graph $G = (\mathcal{V}, E)$ where all possible edges are present with probability ρ. That is for any $v, u \in \mathcal{V}$, we have $\Pr[\{v, u\} \in E] = \rho$. When G is such a graph and $|\mathcal{V}| = n$, we write $G \overset{\$}{\leftarrow} \mathbb{G}(n, \rho)$.

Remark 3.2.2 (Expected degree). If $G = (\mathcal{V}, E) \overset{\$}{\leftarrow} \mathbb{G}(n, p)$ and $p = \frac{d}{n}$, then the expected degree of each node is d, i.e., for all $v \in \mathcal{V}$,

$$\mathbb{E}[\mathtt{deg}(v)] = d. \tag{3.1}$$

Following from Chernoff Lemma 1.3.1, the degree of all nodes in Erdős–Rényi graphs concentrates around the expected value.

Lemma 3.2.9 (Maximum degree of Erdős–Rényi Graph)**.** *Let $n, d \in \mathbb{N}$, $\rho := \frac{d}{n}$, $G = (\mathcal{V}, E) \overset{\$}{\leftarrow} \mathbb{G}(n, p)$. For any $\delta \in [0, 1]$ we have*

$$\Pr[\max_{v\in\mathcal{V}} \mathtt{deg}(v) \geq (1 + \delta)d] \leq n \cdot e^{-\frac{\delta^2 d}{3}}. \tag{3.2}$$

Proof. Let us look at a particular node $v \in \mathcal{V}$. For each $u \in \mathcal{V}$ we introduce a random variable $X_{v,u}$ indicating if there is $\{v, u\} \in E$. We have that

$$\mathbb{E}[X_{v,u}] = \rho, \tag{3.3}$$

which again implies that

$$\mathbb{E}[\mathtt{deg}(v)] = \sum_{u\in\mathcal{V}} \mathbb{E}[X_{v,u}] = n \cdot \phi = d. \tag{3.4}$$

Chernoff (Lemma 1.3.1) now implies that

$$\Pr[\mathtt{deg}(v) \geq (1 + \delta)d] \leq e^{-\frac{\delta^2 d}{3}}. \tag{3.5}$$

By a union bound, we get that

$$\Pr[\max_{v\in\mathcal{V}} \mathtt{deg}(v) \geq (1 + \delta)d] \leq \sum_{v\in\mathcal{V}} \Pr[\mathtt{deg}(v) \geq (1 + \delta)d] \leq ne^{-\frac{\delta^2 d}{3}}. \tag{3.6}$$

□

3.2.3 Universally Composable Security

The UC framework is a general framework for describing and proving cryptographic protocols secure. Its main selling point is that protocols can be described and proven secure in a modular manner while ensuring that the protocol in question remains secure independently of how one may compose the protocol in question with other protocols.

We build upon the journal version of UC [Can20] and refer to this for details about the framework.

In this section, we first provide a brief overview of the security notion of UC.[1] Next, we recap two peculiarities of the framework that are important for our modeling of delayed adversaries (Section 3.3).

3.2.3.1 Security Definition

A protocol is a collection of programs that is to be executed in a distributed setting. In UC, the ideal behavior of a protocol is described via. another program dubbed an *ideal functionality*. Contrary to a protocol, an ideal functionality is intended to be executed on just a single machine with which multiple different parties interact. Intuitively, a protocol is secure if, for any adversary that performs an attack against the protocol, there exists another adversary attacking the ideal functionality such that the observable behavior of the protocol and the functionality are indistinguishable.

To make the notion of "observable behavior" precise, the UC framework introduces an environment $\mathcal{Z}$, which is to provide inputs to the protocol/ideal functionality when it executes and finally outputs a bit which can be interpreted as a guess upon whether or not the environment is interacting with the real protocol or the ideal functionality. For the detailed execution semantics, we refer the reader to [Can20].

Definition 3.2.10 (Execution of protocols (informal))**.** Let Π be a protocol, $\mathcal{A}$ an adversary and $\mathcal{Z}$ an environment. The random variable $\texttt{EXEC}(\mathcal{Z}, \mathcal{A}, \Pi)$ is defined as the binary output that $\mathcal{Z}$ gives when executing Π against $\mathcal{A}$.

A functionality can also be viewed as protocol. This definition, therefore, allows one to formally define what it means for a protocol to implement a functionality.

Definition 3.2.11 (UC emulation)**.** Let Π be a protocol, $\mathcal{F}$ an ideal functionality, and $\approx$ mean that the statistical distance is negligible in the security parameter. The protocol Π emulates $\mathcal{F}$ if

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

Protocols are allowed to call ideal functionalities within their implementation. The UC framework comes with a *universal composition theorem* which ensures that if a protocol Π_1 UC-emulates an ideal functionality $\mathcal{F}_1$ using an ideal functionality $\mathcal{F}_2$, then exchanging all calls to $\mathcal{F}_2$ within Π_1 with calls a protocol Π_2 that UC-emulates $\mathcal{F}_2$ preserves the security.

Below we will refer to a protocol that calls an ideal functionality as being in the "hybrid world" and independent ideal functionalities as being in the "ideal world".

3.2.3.2 Corruptions

The UC framework has no built-in semantics for the corruption of parties in a protocol. Instead, it is up to each individual protocol description to describe the semantics of corruptions whenever the adversary signals that a specific party should be corrupted.

[1]We do not aim to provide an exhaustive presentation of the framework and all of its subtleties but merely wish to present just enough intuition such that this chapter can be understood. For details, please consult the original work [Can20].

Having no built-in corruption model in UC makes the composition theorem independent of a particular corruption model. This allows several different corruption models to be captured within the framework. Some machinery is, however, standard for many different types of corruptions.

The corruption aggregation ITI. The intuition behind UC security is to translate an attack on the protocol to an attack on the specification (the ideal functionality) and thereby show that an adversary does not gain any capabilities interacting with an implementation that another adversary did not have interacting with the ideal functionality. That is to show that any attack is not really an attack as it was already allowed by the specification. This translation between attacks is what is known as a simulator.

For this intuition to make sense when active corruptions are possible, the translation between attacks on the protocol and the specification must necessarily be *corruption preserving*. That is, it should not require more corruptions to attack the ideal functionality than it takes to attack the real protocol. To ensure this, an additional Interactive Turing Machine Instance (ITI) called the *corruption aggregation* ITI is run aside the parties in the protocol. Whenever a party is corrupted, it registers as corrupted by the corruption aggregation ITI. The environment can then query the corruption aggregation ITI to get an overview of who is currently corrupted. Similarly, the ideal functionality makes information about who is corrupted available to the environment. Note that the corruption aggregation ITI is only present for modeling purposes and thus not present when deploying a protocol. In that way, if the simulator corrupts differently than the adversary, the environment can immediately distinguish. A depiction of the flow of information about corruptions can be found in Figure 3.1.

Identity masking function and PIDs. The UC framework allows for very fine-grained control over what knowledge about corruptions is leaked to the environment by parameterizing the corruptions using an *identity-masking-function*, which parties will apply to the information that they send to the corruption aggregation ITI. This can allow an adversary to corrupt only sub-protocols of a party instead of an entire party. We leave this out of the definitions below for clarity as we will always consider corruptions of entire parties (known as PID-wise corruptions within the framework).

Standard corruption models within UC. Canetti presents, among others, how *adaptive corruptions* and *static corruptions* can be modeled within UC. A brief recap of this modeling is provided below.

Instant adaptive corruptions: When an adversary inputs *Corrupt* on the backdoor tape of a party, this party is immediately overtaken by the adversary, and the environment is notified via the corruption aggregation ITI

Static corruptions: If a party receives *Corrupt* on the backdoor tape as the first message, this party is immediately overtaken by the adversary, and the environment is notified via the corruption aggregation ITI. If a *Corrupt* is received later, it is ignored.

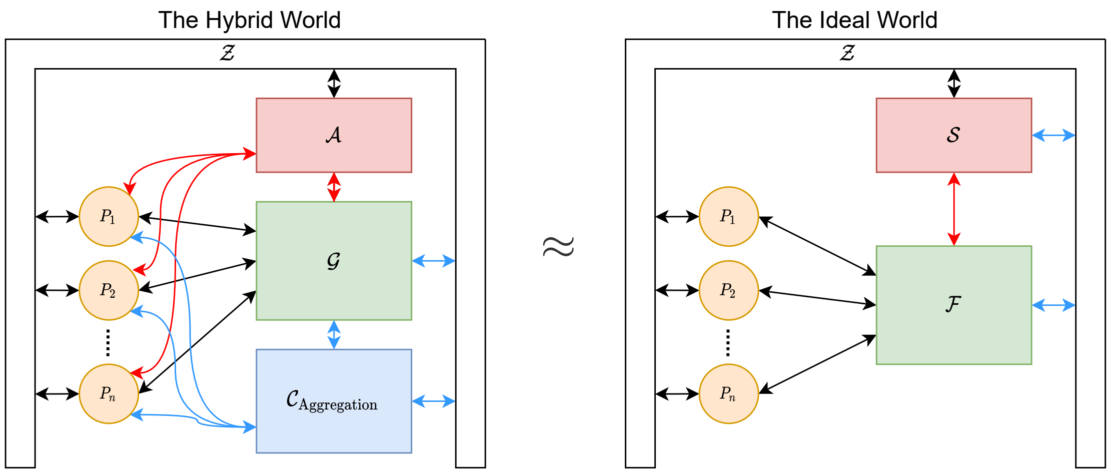


Figure 3.1: A depiction of how corruptions are propagated in both the hybrid world and the ideal world. Corruption requests are passed along the red arrows, and corruption information is propagated along the blue arrows. In the hybrid world, the adversary can corrupt either a party $(p_1, \ldots, p_n)$ or a part of the sub-functionality $\mathcal{G}$. If the adversary corrupts a party p_i by sending a message on the backdoor tape of this party, the party directly informs the corruption aggregation ITI, $\mathcal{C}_{\text{Aggregation}}$, which makes this information available to the environment. If the adversary corrupts a sub-part of the functionality $\mathcal{G}$, this information is recorded in $\mathcal{G}$. Furthermore, when the corruption aggregation ITI, $\mathcal{C}_{\text{Aggregation}}$, is queried for information by the environment, it firsts queries $\mathcal{G}$ for its corruption status and merges this information with its own information, before responding the environment with the aggregated information. For this behavior to be mimicked in the ideal world, the simulator $\mathcal{S}$ needs to make corruption requests to the functionality $\mathcal{F}$ s.t. the information that $\mathcal{F}$ makes available to the environment matches the information the environment obtains by queering $\mathcal{C}_{\text{Aggregation}}$ in the hybrid world. Furthermore, the simulator needs to make corruption information available to the environment that corresponds to the information available from the functionality $\mathcal{G}$ in the hybrid world.

3.2.3.3 Time

There is no built-in notion of time in UC. However, the framework's flexibility allows for modeling a notion of time using an ideal functionality. In this work, we adopt the notion of time presented in TARDIS [Bau+21].

In TARDIS, time is modeled via a global functionality dubbed a *ticker* (written $\bar{\mathcal{G}}_{\text{Ticker}}$). The ticker's job is to keep track of time and enforce that any party has enough time to perform the actions it wishes to perform between any two time steps. It does so by allowing parties to register by the functionality and only allows the environment to progress time once it has heard that this is okay from all registered parties.

Functionalities can query the ticker and get an answer as to whether or not time has passed since the last time they asked the ticker. Importantly, this query can *only* be made by functionalities and not parties. That is, this modeling of time does not tie the protocols to be designed under a specific synchrony assumption, as parties are oblivious to time. The only way that they can observe the passing of time is by asking functionalities. This parallels the real world: we do not have raw access to time, only clocks. The level of information functionalities provided to parties about time determines possible assumptions about synchrony.

The ticker functionality, as described in TARDIS, is referenced below for completeness.

Functionality $\bar{\mathcal{G}}_{\text{Ticker}}$

The functionality keeps track of a set of registered parties $\mathcal{P}$, a set of registered functionalities F, a set of activated parties L_P and a set of functionalities L_F that have been informed about the current tick. Initially, $\mathcal{P} = F = L_P = L_F := \varnothing$.

Party registration: Upon receiving (*Register*) from honest party p_i add p_i to $\mathcal{P}$ and send (*Registered*) to p_i.

Functionality registration: Upon receiving (*Register*) from functionality $\mathcal{F}$, add $\mathcal{F}$ to F and send (*Registered*) to $\mathcal{F}$.

Tick: Upon receiving (*Tick*) from the environment, do the following: If $\mathcal{P} == L_P$, reset $L_P := \varnothing$ and $L_F := \varnothing$ and send (*Ticked*) to the adversary. Else send (*NotTicked*) to the environment.

Ticked request: Upon receiving (*Ticked?*) from $\mathcal{F} \in F$ if $\mathcal{F} \in L_F$, send (*NotTicked*) to $\mathcal{F}$, else add F to L_F and send (*Ticked*) to $\mathcal{F}$.

Record party activation: Upon receiving (*Activated*) from party $p_i \in \mathcal{P}$, add $p_i \in L_P$ and send (*Recorded*) to p_i.

Ticked?-convention. In the remainder of this chapter, we adopt the convention (also used in [Bau+21]) that when describing ideal functionalities, we omit *Ticked?* queries to $\bar{\mathcal{G}}_{\text{Ticker}}$ from the description. Functionalities are instead assumed to make this query whenever they are activated and, in case of a positive answer, perform whatever action described by **Tick**. We adopt the convention that, for brevity, we leave out the registration of functionalities and parties by the global ticker. All of the functionalities and protocols we consider will, upon initialization as the first thing registered by the global ticker.

How to prevent fast-forwarding? A thing to note about the ticker is that it does not notify ideal functionalities when time progresses. Instead, it is up to functionalities to query the ticker to determine whether or not time has passed. Nor does the ticker wait to hear from functionalities before progressing time.

One could worry that this allows the environment to *fast-forward time* by activating the parties without activating ideal functionalities, thereby preventing them from enforcing timing-based properties. How would a time-bounded channel enforce that messages are delivered timely if it is not activated often enough to notice that time passes?

This kind of “attack” is prevented separately in the real and ideal world:

Real world: In the real world, it is up to the protocol designer to ensure that the attack cannot happen. This can be done by ensuring parties activate the respective functionalities before passing (*Activated*) to the ticker. Intuitively, this corresponds to the fact that if a protocol is expected to work correctly, then sub-protocols that depend upon a time should be activated regularly such that they can check whether or not time has passed.

Ideal world: This attack is particularly troublesome in the ideal world. Here, the environment can tell dummy parties to pass on time, and there is no mechanism to prevent this. Note, however, that when the ticker progresses time, it notifies the adversary about this. In the ideal world, the adversary is a simulator and is therefore programmed when proving the protocol secure. The simulator can, therefore, ensure to activate functionalities correspondingly to their activation pattern in the real protocol. Intuitively, if one thinks of the simulator as a translation of attacks on the real world to the ideal world, then it is a part of the translation of an attack to ensure that activations are provided in a similar pattern in the ideal world.

Global functionalities within plain UC. TARDIS' ticker functionality is technically defined within the GUC framework [Can+07]. However, as pointed out in [Bad+20], the GUC framework has not been updated since its introduction, even though it relies on the UC framework, which has been revised and updated several times. Furthermore, [Bad+20] points out that several technical subtleties of the composition theorem of GUC are under-specified, which, at best, leaves its correctness unproven. The compatibility with the latest version of UC, which we use in this work, is thus unclear.

However, [Bad+20] introduces machinery to handle "global subroutines", which can be used to model similar global setup assumptions to global functionalities, and extends the composition theorem of UC to cover such "global subroutines" directly within the version of UC also adapted for this work. They also show examples of global functionalities that can instead be modeled as global subroutines. One of their examples [Bad+20, Section 4.3] of such a transformation is that they show that [Bad+17] that implements a transaction ledger using a global clock (similar to the one from [Kat+13]), instead could have been done directly within UC, by modeling the clock as a global subroutine instead of a global functionality. We note that $\bar{\mathcal{G}}_{\text{Ticker}}$ is *regular* (informally, it does not spawn new ITIs) and as all of the protocols considered in this work are $\bar{\mathcal{G}}_{\text{Ticker}}$-*subroutine respecting* (informally, all subroutines except $\bar{\mathcal{G}}_{\text{Ticker}}$ only communicate with ITIs within the session). Therefore, we can use the same approach as [Bad+20, Section 4.3] (in particular, can adopt the same *identity bound* for the environment to ensure that the ticker works as expected) to keep our modeling within plain UC.

3.3 Delayed Adversaries within UC

In this section, we describe the semantics of delayed corruptions within the UC framework. First, we introduce the semantics for δ-delayed corruptions via *corruption shells*. Next, we revisit the standard adaptive corruptions using corruption shells. Finally, we relate the standard notion of adaptive corruptions to a 0-delayed adversary.

We define the notion of a delayed adversary precisely within the UC-model via what we call *δ-delayed corruptions* or a *δ-delayed adversary.* For such an adversary, executing a corruption takes at least δ time. The delay can be thought of as either the time it takes to hack into the system or the time it takes to physically orchestrate an attack on the specific property that hosts the system. To capture this within UC, we introduce an additional token that an adversary has to use when wanting to corrupt a party. The two corruption tokens that can be passed to a party are the *Precorrupt* token and the *Corrupt* token. When receiving a *Precorrupt* token, the party notes the time it received this token, t, and ignores all *Corrupt* tokens received before $t + \delta$. When a *Corrupt* token is received at or after time $t + \delta$, the party becomes corrupted in the usual manner.

Below we give a more detailed description of how this corruption model can be captured within the UC framework.

3.3.1 The δ-delay Shell

Including code that models corruption behavior in each protocol and ideal functionality, description is tedious and error-prone. Therefore, we separate the concern of describing corruption behaviors from describing the protocol by introducing protocol transformers, dubbed *shells*, which extend a protocol that does not handle corruption tokens into one that obeys a particular corruption behavior. In particular, we provide the following two shells for δ-delayed corruptions:

$\mathcal{D}^{\delta}_{\mathsf{Real}}$: This a wrapper around a protocol Π. It ensures that the protocol respects δ-delayed corruptions. The wrapper preserves the functionality of Π but additionally ensures that corruptions are executed as expected.

$\mathcal{D}^{\delta}_{\mathsf{Ideal}}$: This is a wrapper around an ideal functionality $\mathcal{F}$. It ensures that the functionality respects δ-delayed corruptions and preserves the functionality of $\mathcal{F}$ but additionally ensures that corruptions are executed as expected.

Both shells intuitively work in the same way: They track when *Precorrupt* tokens are delivered and only accept corruption tokens for a particular party δ time later. Having two different shells is necessary as the protocol shell needs to wrap the individual ITMs executing the protocol, whereas the ideal shell needs to wrap only the ITM running the ideal functionality.

Additionally, both shells allow the first message sent to a specific party to initialize the precorruption time. The delay shells for real parties ensure to use this initialization option when an inner protocol sends a message to a sub-routine for the first time. This ensures that the time of precorruption is inherited when new sub-routines are spawned and thereby induces the natural behavior for PID-wise corruptions, i.e., that any sub-routine can be corrupted no later than the routine that spawned it. The initialized precorruption time is allowed to be negative. This allows the environment to start the protocol in a state where some parties are precorrupted in the past and hence be able to immediately corrupt these parties at the start of the protocol (similar to letting some parties be statically corrupted).

The shells that wrap the individual party's ITMs do not have access to query the ticker for the time, whereas the ideal shells can do this freely. We solve this by letting the $\mathcal{D}_{\mathsf{Real}}$ spawn a *corruption-clock* (written $\mathcal{F}_{\text{CorruptionClock}}$) which precisely allows the shells to access time. Importantly, this does not reintroduce a global synchrony assumption, as our shells prevent the inner protocols from communicating with the corruption clock. Therefore, the corruption clock is only an artifact of our modeling and will not appear when running the protocol.

Functionality $\mathcal{F}_{\text{CorruptionClock}}$

The functionality maintains a counter `Time`. Initially, `Time` $:= 0$.

Time?: When receiving ($\mathit{Time?}$) from a party $p_i \in \mathcal{P}$ it returns (Time, `Time`) to p_i.

Tick: It updates `Time` $:=$ `Time` $+ 1$.

When describing $\mathcal{D}_{\mathsf{Real}}$, we will leave out calls to $\mathcal{F}_{\text{CorruptionClock}}$ for brevity, but these happen each time the shell uses any notion of time.

We amend the corruption aggregation ITI presented in [Can20] also to make information about the precorruptions an adversary has used available to the environment (and, similarly, the ideal functionalities). This prevents a simulator from using more precorruption tokens or corrupting faster than the real adversary.

Aside from ensuring the protocol corruption delays are respected, the $\mathcal{D}_{\mathsf{Ideal}}$ additionally propagates both precorruption and corruption-tokens to the "inner functionality" (the functionality that the shell is a wrapper around). This is done to ensure that the simulator appended to the ideal functionalities can gain functionality-specific powers when performing a corruption. For example, it might be that a channel does not need to respect delivery guarantees when the sender gets corrupted (for an example of this, see Section 3.5.1).

Below we provide formal descriptions of both shells.

Function $\mathcal{D}^{\delta}_{\mathsf{Real}}(\Pi)$

The shell wraps each party $p_i \in \mathcal{P}$ in a small wrapper that maintains a variable `PrecorruptionTime`$_i$. Initially, `PrecorruptionTime`$_i$ $:= \bot$. When receiving precorruptions and corruptions, the wrapper has the behavior described below. The wrapper also filters out any communication with $\mathcal{F}_{\text{CorruptionClock}}$, and on all other inputs, it simply forwards the inputs/outputs to/from the original protocol.

Initialization: If p_i receives ($\mathit{Initialize}, \tau$) as the first message, then the party updates `PrecorruptionTime`$_i$ $:= \tau$ and if $\tau \neq \bot$ then also notifies the corruption-aggregation ITI.

Precorruption: If $p_i \in \mathcal{P}$ receives $\mathit{Precorrupt}$ at time τ, then the party first notifies the corruption-aggregation ITI by sending ($\mathit{Precorrupt}, p_i$) to this machine. It then updates `PrecorruptionTime`$_i$ $:= \tau$.

Corruption: When p_i receives $\mathit{Corrupt}$ at time τ, then p_i checks if `PrecorruptionTime`$_i$ $+ \delta \leq \tau$. If that is not the case, the request is ignored. Otherwise, the party first notifies the corruption-aggregation ITI by sending ($\mathit{Corrupt}, p_i$) to this machine, and then it corrupts p_i by forwarding

Corrupt to Π. Each time p_i is activated after this, it sends its entire local state of the inner protocol to the adversary and forwards all messages m (assuming that m includes both content and recipient) that are written on the backdoor tape of p_i.

Whenever the shell of p_i detects that the inner protocol sends a message to a new sub-routine for the first time, it sends $(\mathit{Initialize}, \texttt{PrecorruptionTime}_i)$ to the subroutine before forwarding the message of the inner protocol.
Furthermore, the shell starts a separate corruption aggregation ITI. It maintains two lists, `Precorrupted`, and `Corrupted`, that initially are both empty. The corruption aggregation ITI has the following behavior:

Precorruption Registration: When receiving a $(\mathit{Precorrupt}, p)$ from a party p it sets $\texttt{Precorrupted} := p :: \texttt{Precorrupted}$.

Corruption Registration: When receiving a $(\mathit{Corrupt}, p)$ from a party p it sets $\texttt{Corrupted} := p :: \texttt{Corrupted}$.

Corruption Status: When receiving *CorruptionStatus* from the environment it queries all sub-functionalities of the protocol for their corruption status and updates the `Precorrupted` and `Corrupted`-lists accordingly. Finally, it sends $(\texttt{Precorrupted}, \texttt{Corrupted})$ back to the environment.

Function $\mathcal{D}^{\delta}_{\mathsf{Ideal}}(\mathcal{F})$

The shell wraps the functionality in a wrapper that maintains two lists `Precorrupted` and `Corrupted` that initially are both empty. Furthermore, it has a map $\texttt{PrecorruptionTimeMap} : \mathcal{P} \to \textsc{Time}$ and a counter to keep track of time `Time`, which is initially instantiated as 0. When receiving precorruptions, corruptions, and corruption-status requests, it has the following behavior, and on all other inputs/outputs, it forwards the inputs to/from $\mathcal{F}$.

Initialization: If the functionality receives $(\mathit{Initialize}, \tau)$ at the port belonging to p as the first message for this party, then the party updates $\texttt{PrecorruptionTimeMap}[p] := \tau$. If $\tau \neq \bot$, then it also updates $\texttt{Precorrupted} := p :: \texttt{Precorrupted}$, and forwards $(\mathit{Initialize}, \tau)$ to the inner functionality.

Precorruption: When receiving $(\mathit{Precorrupt}, p)$ and p is a valid PID of a dummy party, then it adds the current time, `Time`, to $\texttt{PrecorruptionTimeMap}[p] := \texttt{Time}$ and updates $\texttt{Precorrupted} := p :: \texttt{Precorrupted}$. Furthermore, it propagates $(\mathit{Precorrupt}, p)$ to $\mathcal{F}$.

Corruption: When receiving $(\mathit{Corrupt}, p)$ where p is a valid PID of a dummy party, then the functionality checks if $\texttt{PrecorruptionTimeMap}[p] + \delta \leq \texttt{Time}$. If that is the case, it updates $\texttt{Corrupted} := p :: \texttt{Corrupted}$ and returns to the adversary all the values received from p and output to p so far. From now on,

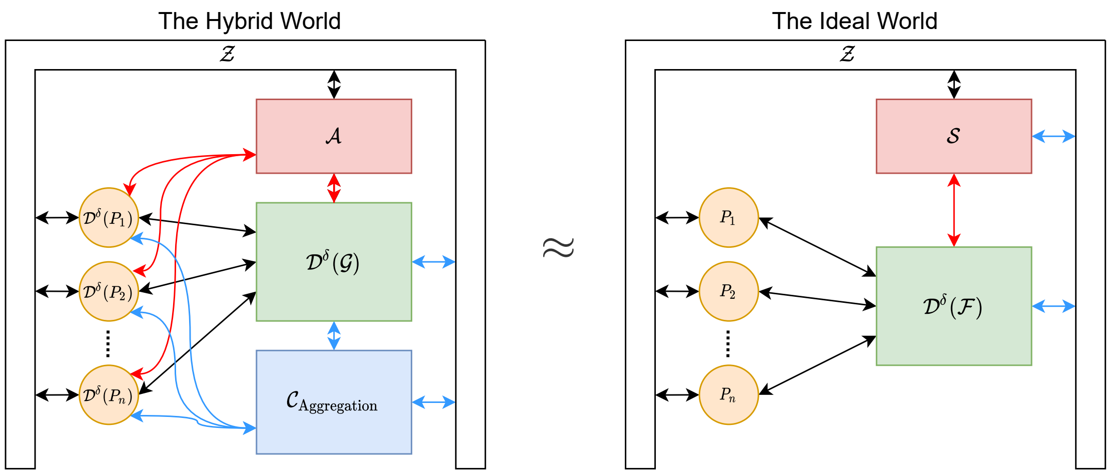

The Hybrid World
$\mathcal{Z}$
$\mathcal{A}$
$\mathcal{D}^\delta(P_1)$
$\mathcal{D}^\delta(P_2)$
$\mathcal{D}^\delta(P_n)$
$\mathcal{D}^\delta(\mathcal{G})$
$\mathcal{C}_{\text{Aggregation}}$
$\approx$
The Ideal World
$\mathcal{Z}$
$\mathcal{S}$
P_1
P_2
P_n
$\mathcal{D}^\delta(\mathcal{F})$

Figure 3.2: A depiction of the security statement for a protocol that implements an ideal functionality $\mathcal{F}$ using the functionality $\mathcal{G}$ against a δ-delayed adversary.

inputs from p are ignored but are given via the adversary's backdoor tape. Furthermore, it propagates $(\mathit{Corrupt}, p)$ to $\mathcal{F}$.

If the request is sent too early, it is ignored.

Inputs: If the functionality receives (Input, p, v) from the adversary and $p \in$ `Corrupted`, then v is forwarded to $\mathcal{F}$ as if it was directly input by p to $\mathcal{F}$.

Corruption Status: When receiving $\mathit{CorruptionStatus}$ from the environment it sends $(\texttt{Precorrupted}, \texttt{Corrupted})$ back to the environment.

Tick: The functionality updates $\texttt{Time} \coloneqq \texttt{Time} + 1$.

The additional (Input, p, v) command accepted by the ideal shell allows an adversary to input a message v on behalf of party p if p is corrupted. This follows how standard byzantine corruptions are treated and modeled in the UC framework.

We next formally define what it means for a protocol to securely implement a functionality against a δ-delayed adversary, see also Figure 3.2 for a graphical depiction.

Definition 3.3.1 (UC-security against delayed adversaries)**.** Let $\delta \in \mathbb{N}$. We say that a protocol Π securely implements an ideal functionality $\mathcal{F}$ *against a δ-delayed adversary* when $\mathcal{D}^\delta_{\mathsf{Real}}(\Pi)$ securely implements $\mathcal{D}^\delta_{\mathsf{Ideal}}(\mathcal{F})$ in the usual UC sense [Can20], i.e., if

$$\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z}, \texttt{EXEC}(\mathcal{Z}, \mathcal{A}, \mathcal{D}^\delta_{\mathsf{Real}}(\Pi)) \approx \texttt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{D}^\delta_{\mathsf{Ideal}}(\mathcal{F})).$$

Note that security against a delayed adversary is defined both for functionalities that have special behavior defined for receiving precorruptions and functionalities that do not have any such behavior defined, as the default for protocols/functionalities is to ignore unrecognized inputs.

3.3.2 Relating Corruption Models

In this section, we relate the notion of a 0-delayed adversary to the standard notion of an adaptive adversary in UC. We further show that any protocol secure against a fast adversary is also secure against a slower adversary. These results allow us to reuse cryptographic constructions, which are already proven secure modularly when implementing larger constructions.

Byzantine corruptions and 0*-delayed corruptions.* To showcase the generality of the δ-delayed corruption model, we relate this model to the standard model of adaptive Byzantine corruptions defined in UC. To quantify how these notions relate, we introduce two Byzantine shells similar to the delay shells. The byzantine shells are meant to precisely encapsulate the corruption model as presented in [Can20]. We believe these are of independent interest as by using these, it can be avoided to clutter the protocol and functionality description with a specific corruption model.

Function $\mathcal{B}_{\mathsf{Real}}(\Pi)$

The shell adds the following behavior to each party $p_i \in \mathcal{P}$. If any other inputs are received than the ones below, it is the original code of the party that is executed.

Corruption: If $p_i \in \mathcal{P}$ receives $\mathit{Corrupt}$ then the party first notifies the corruption-aggregation ITI by sending $(\mathit{Corrupt}, p_i)$ to this machine.

Each time p_i is activated after this, it sends its entire local state of the inner protocol to the adversary and forwards all messages m (assuming that m includes both content and recipient) that are written on the backdoor tape of p_i.

Furthermore, the shell runs a separate corruption-aggregation ITI. It maintains a list `Corrupted`, which initially is set to be the empty list and has the following behavior:

Registration: When receiving a $(\mathit{Corrupt}, p)$ from a party p it sets `Corrupted` $:= p ::$ `Corrupted`.

Corruption Status: When receiving $\mathit{CorruptionStatus}$ from the environment it queries all sub-functionalities of the protocol for their corruption status and updates the `Corrupted`-list accordingly. Finally, it sends `Corrupted` back to the environment.

Function $\mathcal{B}_{\mathsf{Ideal}}(\mathcal{F})$

The functionality maintains a list of corrupted parties, `Corrupted`, initially set to be the empty list. Upon receiving the following

Corruption: If the functionality receives $(\mathit{Corrupt}, p)$ from the adversary and p is a valid PID of the dummy parties, it updates `Corrupted` $:= p ::$ `Corrupted` and returns to the adversary all the values received from p and output to p so far.

From now on, inputs from p are ignored but are given via the adversary's backdoor tape. Furthermore it propagates $(\mathit{Corrupt}, p)$ to $\mathcal{F}$.

Inputs: If the functionality receives (Input, p, v) from the adversary and $p \in$ `Corrupted` then v is forwarded to $\mathcal{F}$ as if it was directly input by p to $\mathcal{F}$.

Corruption Status: When receiving $\mathit{CorruptionStatus}$ from the environment it sends `Corrupted` back to the environment.

Security against 0-delayed adversary implies security in the standard model and vice versa if the implemented functionality ignores precorruption and initialization tokens. We encapsulate this intuition in the theorem below.

Theorem 3.3.2. *Let* Π *be a protocol and* $\mathcal{F}$ *an ideal functionality that ignores precorruptions and initializations.* $\mathcal{B}_{\mathsf{Real}}(\Pi)$ *securely implements* $\mathcal{B}_{\mathsf{Ideal}}(\mathcal{F})$ *if and only if* $\mathcal{D}^0_{\mathsf{Real}}(\Pi)$ *securely implements* $\mathcal{D}^0_{\mathsf{Ideal}}(\mathcal{F})$*.*

Formally,

$$\begin{aligned}
&\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z}, \mathtt{EXEC}(\mathcal{Z}, \mathcal{A}, \mathcal{B}_{\mathsf{Real}}(\Pi)) \approx \mathtt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{B}_{\mathsf{Ideal}}(\mathcal{F})) \\
&\iff \forall \mathcal{A}'\ \exists \mathcal{S}' \forall \mathcal{Z}', \mathtt{EXEC}(\mathcal{Z}', \mathcal{A}', \mathcal{D}^0_{\mathsf{Real}}(\Pi)) \approx \mathtt{EXEC}(\mathcal{Z}', \mathcal{S}', \mathcal{D}^0_{\mathsf{Ideal}}(\mathcal{F})).
\end{aligned} \tag{3.7}$$

Proof Sketch. We prove the two directions of the implication individually.

"$\Longrightarrow$": We let $\mathcal{A}'$ be any adversary and construct an adversary $\mathcal{A}$ by wrapping the original adversary $\mathcal{A}'$ with a shell that forwards all inputs/outputs except precorruptions to/from $\mathcal{A}'$. Whenever $\mathcal{A}$ receives a $\mathit{Precorrupt}$ directed to p_i from $\mathcal{A}'$ it forwards $(\mathit{Precorrupt}, p_i)$ to the environment instead. We now use the LHS of Equation (3.7) to obtain a simulator $\mathcal{S}$ s.t.

$$\forall \mathcal{Z}, \mathtt{EXEC}(\mathcal{Z}, \mathcal{A}, \mathcal{B}_{\mathsf{Real}}(\Pi)) \approx \mathtt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{B}_{\mathsf{Ideal}}(\mathcal{F})). \tag{3.8}$$

Given $\mathcal{S}$ we construct $\mathcal{S}'$ by running $\mathcal{S}$ inside $\mathcal{S}'$. Each time $\mathcal{S}$ outputs $(\mathit{Precorrupt}, p_i)$ to the environment then $\mathcal{S}'$ outputs $(\mathit{Precorrupt}, p_i)$ to $\mathcal{D}^0_{\mathsf{Ideal}}(\mathcal{F})$. All other inputs and outputs are directly forwarded to and from $\mathcal{S}$. Note that precorruptions are ignored by $\mathcal{F}$, and therefore $\mathcal{F}$ does not change its behavior based upon these.

Let us now, for the sake of contradiction, assume that there exists some environment $\mathcal{Z}'$ that can distinguish against $\mathcal{A}'$ and $\mathcal{S}'$, i.e.,

$$\mathtt{EXEC}(\mathcal{Z}', \mathcal{A}', \mathcal{D}^0_{\mathsf{Real}}(\Pi)) \not\approx \mathtt{EXEC}(\mathcal{Z}', \mathcal{S}', \mathcal{D}^0_{\mathsf{Ideal}}(\mathcal{F})) \tag{3.9}$$

Let us now show how to construct an environment, $\mathcal{Z}$, that can distinguish for the byzantine setting and thereby contradict Equation (3.8).

We build $\mathcal{Z}$ by running $\mathcal{Z}'$ inside and forwarding all inputs and outputs to $\mathcal{Z}'$. $\mathcal{Z}$ only deviates from $\mathcal{Z}'$ in the two cases below:

- Whenever a $\mathit{CorruptionStatus}$ command is issued by $\mathcal{Z}'$ to the corruption aggregation ITI, we amend the answer with an additional list of precorruptions we have received from $\mathcal{A}$ so far.

- Whenever a $(\mathit{Initialize}, \tau)$ command is sent to some party, it is not forwarded by $\mathcal{Z}$ but instead recorded as a precorruption of this party. This does not change the protocol's behavior nor the ideal functionality, as these are ignored.

In particular, $\mathcal{Z}$ forwards the guess on which world it is placed in from $\mathcal{Z}'$.

We observe that

$$\mathtt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{B}_{\mathsf{Ideal}}(\mathcal{F})) \approx \mathtt{EXEC}(\mathcal{Z}', \mathcal{S}', \mathcal{D}^0_{\mathsf{Ideal}}(\mathcal{F})), \tag{3.10}$$

and

$$\mathtt{EXEC}(\mathcal{Z}, \mathcal{A}, \mathcal{B}_{\mathsf{Real}}(\Pi)) \approx \mathtt{EXEC}(\mathcal{Z}', \mathcal{A}', \mathcal{D}^0_{\mathsf{Real}}(\Pi)). \tag{3.11}$$

Together with Equation (3.9), this contradicts Equation (3.8) and thus concludes the case.

"$\Longleftarrow$": The proof of this case mirrors the other case. We are now given $\mathcal{A}$ and construct $\mathcal{A}'$ by sending *Precorrupt*-tokens just before *Corrupt*-tokens. From the RHS of Theorem 3.3.2, we get a simulator $\mathcal{S}'$ which we use to construct $\mathcal{S}$ by forwarding everything except *Precorrupt*-tokens. Finally, we assume for the sake of contradiction that there exists a $\mathcal{Z}$ that can distinguish, build an environment $\mathcal{Z}'$ using this (removing *Precorrupt*-tokens and initializations), and derive a contradiction similar to the other case. □

Note that the above theorem allows reusing constructions that are proven secure against a standard adaptive adversary when building complex systems that are to be secure against a 0-delayed adversary.

Lifting security to weaker adversaries. If protocols that are proven secure within different corruption models are composed, it gets hard to identify the final security guarantee provided by the composed construction. Intuitively, one would presume that a protocol that is proven secure against an adversary able to do "fast" corruptions is also secure against an adversary only able to do "slow" corruptions. Using precise shells to quantify corruption speed lets us capture this intuition in the lemma below.

Theorem 3.3.3 (Lifting Security to Slower Corruptions)**.** *Let $\delta, \delta' \in \mathbb{N}$, s.t. $\delta \leq \delta'$, let Π be a protocol, and let $\mathcal{F}$ be an ideal functionality. If $\mathcal{D}^{\delta}_{\mathsf{Real}}(\Pi)$ securely implements $\mathcal{D}^{\delta}_{\mathsf{Ideal}}(\mathcal{F})$, then $\mathcal{D}^{\delta'}_{\mathsf{Real}}(\Pi)$ securely implements $\mathcal{D}^{\delta'}_{\mathsf{Ideal}}(\mathcal{F})$.*

Formally,

$$\begin{aligned} &\forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{Z}, \mathtt{EXEC}(\mathcal{Z}, \mathcal{A}, \mathcal{D}^{\delta}_{\mathsf{Real}}(\Pi)) \approx \mathtt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{D}^{\delta}_{\mathsf{Ideal}}(\mathcal{F})) \\ &\Longrightarrow \forall \mathcal{A}', \exists \mathcal{S}', \forall \mathcal{Z}', \mathtt{EXEC}(\mathcal{Z}', \mathcal{A}', \mathcal{D}^{\delta'}_{\mathsf{Real}}(\Pi)) \approx \mathtt{EXEC}(\mathcal{Z}', \mathcal{S}', \mathcal{D}^{\delta'}_{\mathsf{Ideal}}(\mathcal{F})). \end{aligned} \tag{3.12}$$

Proof. Let H be the hypothesis (LHS of the implication), and let $\mathcal{A}'$ be an adversary. We define $\mathsf{Filter}(\mathcal{A}, \delta)$ as a wrapper around an adversary that filters out corruption requests that are too early w.r.t. δ.

Using H, we know that there exists a simulator $\mathcal{S}$ s.t.

$$\forall \mathcal{Z}, \mathtt{EXEC}(\mathcal{Z}, \mathsf{Filter}(\mathcal{A}', \delta'), \mathcal{D}^{\delta}_{\mathsf{Real}}(\Pi)) \approx \mathtt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{D}^{\delta}_{\mathsf{Ideal}}(\mathcal{F})). \tag{3.13}$$

Let us now show,

$$\forall \mathcal{Z}, \texttt{EXEC}(\mathcal{Z}, \mathsf{Filter}(\mathcal{A}', \delta'), \mathcal{D}^{\delta}_{\mathsf{Real}}(\Pi)) \approx \texttt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{D}^{\delta'}_{\mathsf{Ideal}}(\mathcal{F})). \tag{3.14}$$

Assume for the sake of contradiction that there exists an environment $\mathcal{Z}$ that can distinguish in Equation (3.14). We use this to build an environment $\mathcal{Z}'$ which can distinguish in Equation (3.13) with at least as big an advantage. $\mathcal{Z}'$ works by forwarding everything to and from $\mathcal{Z}$. Except if, at any point in time, there is a *Precorrupt*-token followed by a *Corrupt* send with strictly less than δ' between them, then $\mathcal{Z}'$ immediately guesses that it is in the ideal case.

Every time this happens, the environment is correct, and every time this does not occur, the execution is precisely similar to that of Equation (3.13) this implies Equation (3.14).

We now define $\mathcal{S}' \triangleq \mathcal{S}$ and let $\mathcal{Z}'$ be any environment. We specialize Equation (3.14) with $\mathcal{Z}'$ and obtain

$$\texttt{EXEC}(\mathcal{Z}', \mathsf{Filter}(\mathcal{A}', \delta'), \mathcal{D}^{\delta}_{\mathsf{Real}}(\Pi)) \approx \texttt{EXEC}(\mathcal{Z}', \mathcal{S}, \mathcal{D}^{\delta'}_{\mathsf{Ideal}}(\mathcal{F})). \tag{3.15}$$

Furthermore,

$$\begin{aligned} \texttt{EXEC}(\mathcal{Z}', \mathsf{Filter}(\mathcal{A}', \delta'), \mathcal{D}^{\delta}_{\mathsf{Real}}(\Pi)) &\approx \texttt{EXEC}(\mathcal{Z}', \mathsf{Filter}(\mathcal{A}', \delta'), \mathcal{D}^{\delta'}_{\mathsf{Real}}(\Pi)) && (3.16)\\ &\approx \texttt{EXEC}(\mathcal{Z}', \mathcal{A}', \mathcal{D}^{\delta'}_{\mathsf{Real}}(\Pi)). && (3.17) \end{aligned}$$

Equation (3.16) holds as if early corruptions are ignored, then $\mathcal{D}^{\delta}_{\mathsf{Real}}(\Pi)$ and $\mathcal{D}^{\delta'}_{\mathsf{Real}}(\Pi)$ are identically distributed. Equation (3.17) holds as it is not observable by the environment if the filter or the shell ignores the corruption. Together Equations (3.15) and (3.17) finishes the proof. □

Note that if one considered a simpler model with just one corruption token and a subsequent automatic effectuation of the corruption a certain time after such token was input (instead of a model like ours with separate tokens for precorruptions and corruptions), then Theorem 3.3.3 would not hold. The reason is that in such a model, a fast adversary would not be able to imitate a slow adversary. Hence, in such a model, a fast adversary would not be strictly "stronger" than a slow adversary.

Theorems 3.3.2 and 3.3.3 together imply that any protocol that is secure against a standard adaptive adversary in UC is also secure against any δ-delayed adversary.

3.4 Concrete Bounds for Diameters of Erdős–Rényi Graphs

The proofs in this section are inspired by proofs in [BHK20].

3.4.1 Logarithmic Diameter

This section is dedicated to proving that an Erdős–Rényi graph with a constant average degree d has a logarithmic diameter except with a probability negligible in the degree d.

Lemma 3.4.1 (Erdős–Rényi graphs with logarithmic diameter). *Let $n \in \mathbb{N}$, $d \in \mathbb{R}$, $\gamma, \delta_1, \delta_2 \in [0,1]$, and $\rho := \frac{d}{n}$. Furthermore, let $\alpha \in \mathbb{R}$, let $G = (\mathcal{V}, E) \overset{\$}{\leftarrow} \mathbb{G}(n, \rho)$, and let $t_0 := \frac{\log\left(\frac{\gamma n}{(1-\delta_1)d}\right)}{\log((1-\delta_2)\alpha)} + 1$. If*

$$e^{-d\gamma} + \frac{\gamma\alpha}{1-\gamma} \leq 1 \quad \text{and } (1-\delta_2)\cdot\alpha > 1, \tag{3.18}$$

then

$$\Pr[\neg\phi_{Diam}(G, t_0+1)] \leq n\left(e^{-\frac{\delta_1^2 d}{2}} + t_0 e^{-\frac{\delta_2^2\alpha(1-\delta_1)d}{2}}\right) + e^{-n\cdot(d\gamma^2-2)}. \tag{3.19}$$

Proof. First, we bound the probability that a single node cannot reach a constant fraction of all nodes within a logarithmic number of steps. For $v \in \mathcal{V}$ let $D_v := (|\Gamma^{t_0}(v)| < \gamma \cdot n)$ i.e., the event that v does not reach at least a γ-fraction of the nodes within t_0 steps. Our goal is now to bound $\Pr[D_v]$. Let

$$A_0 := (\mathtt{deg}(v) > (1-\delta_1)d) \tag{3.20}$$

i.e., the event that v's degree is less than $(1-\delta_1)d$, and let

$$B_i := (|\theta(v, i+1)| > (1-\delta_2)\alpha|\theta(v,i)|), \quad C_i := \left(|\Gamma^i(v)| > \gamma n\right), \quad \text{and } A_i := B_i \vee C_i, \tag{3.21}$$

for $i = 1, \ldots, t_0 - 1$. We note that

$$t_0 = \frac{\log\left(\frac{\gamma n}{(1-\delta_1)\cdot d}\right)}{\log((1-\delta_2)\alpha)} + 1 \iff (1-\delta_1)d((1-\delta_2)\alpha)^{t_0-1} = \gamma n. \tag{3.22}$$

Therefore, if A_0 and $B_1, \ldots, B_{t_0-1}$ holds, then

$$\begin{aligned}
|\Gamma^{t_0}(v)| &= \sum_{i=0}^{t_0} |\theta(v,i)| \\
&= 1 + \sum_{i=1}^{t_0} |\theta(v,i)| \\
&= 1 + \sum_{i=0}^{t_0-1} |\theta(v,i+1)| \\
&\geq 1 + \sum_{i=0}^{t_0-1} ((1-\delta_2)\alpha)^i |\theta(v,1)| \\
&\geq 1 + (1-\delta_1)d \sum_{i=0}^{t_0-1} ((1-\delta_2)\alpha)^i \\
&\geq (1-\delta_1)d((1-\delta_2)\alpha)^{t_0-1} \\
&= \gamma n.
\end{aligned} \tag{3.23}$$

Similarly, if just some C_i holds then we get that $|\Gamma^{t_0}(v)| > \gamma n$. Therefore, by contraposition, we have that

$$\left(\bigwedge_{i=0}^{t_0-1} A_i \Longrightarrow \neg D_v\right) \iff \left(D_v \Longrightarrow \bigvee_{i=0}^{t_0-1} \neg A_i\right). \tag{3.24}$$

Hence, we get the following:

$$\begin{aligned}
\Pr[D_v] &\le \Pr\left[\bigcup_{i=0}^{t_0-1} \neg A_i\right] \\
&\le \sum_{i=0}^{t_0-1} \Pr\left[\neg A_i \mid \bigcap_{j<i} A_j\right] \\
&= \Pr[\neg A_0] + \sum_{i=1}^{t_0-1} \Pr\left[\neg A_i \mid \bigcap_{j<i} A_j\right] \\
&= \Pr[\neg A_0] + \sum_{i=1}^{t_0-1} \Pr\left[\neg B_i \cap \neg C_i \mid \bigcap_{j<i} A_j\right] \qquad (3.25)\\
&= \Pr[\neg A_0] + \sum_{i=1}^{t_0-1} \Pr\left[\neg B_i \mid \bigcap_{j<i} A_j \cap \neg C_i\right] \cdot \Pr\left[\neg C_i \mid \bigcap_{j<i} A_j\right] \\
&\le \Pr[\neg A_0] + \sum_{i=1}^{t_0-1} \Pr\left[\neg B_i \mid \bigcap_{j<i} A_j \cap \neg C_i\right] \\
&= \Pr[\neg A_0] + \sum_{i=1}^{t_0-1} \Pr\left[\neg B_i \mid \bigcap_{1\le j<i} B_j \cap \neg C_i \cap A_0\right].
\end{aligned}$$

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

Claim 3.4.2 (Fast expansion to small fraction)**.** *For any $i \in \{1, \ldots, t_0-1\}$ we have*

$$\Pr\left[\neg B_i \mid \bigcap_{1\le j<i} B_j \cap \neg C_i \cap A_0\right] \le e^{-\frac{\delta_2^2 \alpha(1-\delta_1)d}{2}}. \tag{3.26}$$

Proof. We look at the probability space where $\bigcap_{1\le j<i} B_j \cap \neg C_i \cap A_0$ holds. Let $r := |\theta(v,i)|$ and let $U := \mathcal{V} \setminus \Gamma^i(v)$. For each $u \in U$, we introduce a random variable X_u which describes if u is in $\theta(v, i+1)$. As the probability that there is an edge between any two nodes is independent of other edges,

$$\Pr[X_u = 1] = 1 - (1-\rho)^r \ge 1 - e^{-\rho r}. \tag{3.27}$$

The size of $\theta(v, i+1)$ is the sum of these independent variables, i.e.,

$$|\theta(v, i+1)| = \sum_{u\in U} X_u. \tag{3.28}$$

As we are looking at the case where $\neg C_i$, we have $|U| \ge (1-\gamma)n$ which by linearity of expectations gives us that

$$\mathbb{E}[|\theta(v, i+1)|] \ge n(1-\gamma)\left(1 - e^{-\rho r}\right). \tag{3.29}$$

For $\alpha \in \mathbb{N}$, we subtract $\alpha \cdot r$ on each side of the inequality above and get

$$\begin{aligned}\mathbb{E}[|\theta(v,i+1)|] - \alpha r &\geq n(1-\gamma)\left(1 - e^{-\rho r} - \frac{\alpha r}{n(1-\gamma)}\right) \\ &= n(1-\gamma)\left(1 - e^{-d\frac{r}{n}} - \frac{\alpha r}{n(1-\gamma)}\right).\end{aligned} \tag{3.30}$$

We let $x = \frac{r}{n}$ and set $f(x) = 1 - e^{-dx} - x\frac{\alpha}{(1-\gamma)}$. We differentiate this twice and find $f''(x) = -d^2 e^{-dx} \leq 0$, which implies that f is concave, which again means that the minimum values are at one of the endpoints of the function. As $x \in [0,\gamma]$ it is enough to check that $f(0) \geq 0$ and $f(\gamma) \geq 0$ which will imply that $\mathbb{E}[|\theta(v,i+1)|] \geq \alpha \cdot |\theta(v,i)|$.

$$f(0) = 1 - e^{-d\cdot 0} - 0 = 0. \tag{3.31}$$

$$f(\gamma) = 1 - e^{-d\gamma} - \frac{\gamma\alpha}{1-\gamma} \geq 0 \iff e^{-d\gamma} + \frac{\gamma\alpha}{1-\gamma} \leq 1. \tag{3.32}$$

We now use Chernoff (Lemma 1.3.1) to bound the probability that this is not the case which means that for any $\delta_2 \in [0,1]$, we get that

$$\Pr[|\theta(v,i+1)| \leq (1-\delta_2)\alpha|\theta(v,i)|] \leq e^{-\frac{\delta_2^2\alpha|\theta(v,i)|}{2}}. \tag{3.33}$$

However, $\bigcap_{j<i} B_j \cap A_0$ and $(1-\delta_2)\cdot\alpha \geq 1$ ensures that

$$\begin{aligned}|\theta(v,i)| &\geq ((1-\delta_2)\alpha)^i \cdot (1-\delta_1)d \\ &\geq (1-\delta_1)d.\end{aligned} \tag{3.34}$$

Hence, within the probability space where $\bigcap_{1\leq j<i} B_j \cap \neg C_i \cap A_0$ holds we have that

$$\Pr[|\theta(v,i+1)| \leq (1-\delta_2)\alpha|\theta(v,i)|] \leq e^{-\frac{\delta_2^2\alpha(1-\delta_1)d}{2}}. \tag{3.35}$$

□

Using Claim 3.4.2. and Chernoff (Lemma 1.3.1) to bound A_0 we get that

$$\begin{aligned}\Pr[D_v] &\leq e^{-\frac{\delta_1^2 d}{2}} + \sum_{i=1}^{t_0-1} e^{-\frac{\delta_2^2\alpha(1-\delta_1)d}{2}} \\ &\leq e^{-\frac{\delta_1^2 d}{2}} + t_0 e^{-\frac{\delta_2^2\alpha(1-\delta_1)d}{2}}.\end{aligned} \tag{3.36}$$

Furthermore, by the union-bound, we get that

$$\begin{aligned}\Pr[\text{exists } v \in \mathcal{V} \text{ s.t. } D_v] &= \Pr\left[\bigcup_{v\in\mathcal{V}} D_v\right] \\ &\leq \sum_{v\in\mathcal{V}} \Pr[D_v] \\ &\leq n\cdot\left(e^{-\frac{\delta_1^2 d}{2}} + t_0 e^{-\frac{\delta_2^2\alpha(1-\delta_1)d}{2}}\right).\end{aligned} \tag{3.37}$$

We now continue to show that for any two non-overlapping sets $S, S' \subseteq V$ where $|S| \geq \gamma n$ and $|S'| \geq \gamma n$, we have with very high probability that there is an edge between S and S'.

$$\begin{aligned}\Pr[\text{No edges from } S \text{ to } S'] &\leq ((1-\rho)^{\gamma n})^{\gamma n} \\ &= \left(\left(1-\frac{d}{n}\right)^{n}\right)^{\gamma^2 n} \\ &\leq e^{-d\cdot n\gamma^2}\end{aligned} \tag{3.38}$$

We now bound the probability that there exist any two such sets of size γn with no edges between them. There are less than 2^{2n} such pairs of sets. Hence, by the union bound, we get that

$$\begin{aligned}&\Pr[\text{exist two non-overlapping sets of size} \geq \gamma n \text{ that are not connected}] \\ &\leq 2^{2n} e^{-d\cdot n\gamma^2} \\ &\leq e^{-n\cdot(d\gamma^2-2)}.\end{aligned} \tag{3.39}$$

A final union-bound on the probabilities for all of the bad events concludes the proof, which shows that such a graph has diameter t_0+1 except with negligible probability. □

3.4.2 Diameter 2

Below we show that an Erdős–Rényi graph can achieve a constant diameter (a diameter of just 2 to be precise) by selecting a square root number of neighbors.

Lemma 3.4.3 (Erdős–Rényi graphs with diameter 2). *Let $n \in \mathbb{N}$, $k \in \mathbb{R}$, and $\rho := \sqrt{\frac{k}{n}}$. For $G = (\mathcal{V}, E) \xleftarrow{\$} \mathbb{G}(n, \rho)$ then*

$$\Pr[\neg\phi_{Diam}(G, 2)] \leq n^2 \cdot e^{-k\cdot\frac{(n-2)}{n}}. \tag{3.40}$$

Proof. We look at a pair of vertices $v, u \in \mathcal{V}$ and let $X_{v,u}$ indicate if u is not reachable from v in two steps. Let $w \in \mathcal{V}$ be an intermediary node. We have $\Pr[\{v, w\} \in E] = \rho$ and $\Pr[\{w, u\} \in E] = \rho$ and thus the probability that either of these is missing is $1-\rho^2$. As there are $n-2$ possible intermediary nodes, we get by the exponential inequality that

$$\begin{aligned}\Pr[X_{v,u} = 1] &= (1-\rho^2)^{n-2} \\ &\leq e^{-\rho^2\cdot(n-2)} \\ &= e^{-k\cdot\frac{(n-2)}{n}}.\end{aligned} \tag{3.41}$$

There are $\binom{n}{2}$ such pairs of nodes which by the union bound gives that:

$$\begin{aligned}\Pr[\text{exists } v, u \in \mathcal{V} \text{ s.t.} X_{v,u} = 1] &\leq \sum_{v,u\in\mathcal{V}} \Pr[X_{v,u} = 1] \\ &= \binom{n}{2} \cdot e^{-\rho^2\cdot(n-2)} \\ &\leq n^2 \cdot e^{-k\cdot\frac{(n-2)}{n}}\end{aligned} \tag{3.42}$$

If there are no such pairs, then all vertices can be reached from all vertices in at most 2 steps. □

3.5 Functionalities

In this section, we define a time-bounded channel between parties and a flooding functionality. The functionalities that we present are:

MessageTransfer*:* This functionality allows one party to send messages to another. This is modeling a point-to-point channel.

Flood*:* This functionality allows all honest parties to disseminate to all other parties.

Conventions for ideal functionalities. Our functionalities need to maintain a counter which is incremented each time a tick happens (similarly to what $\mathcal{D}_{\mathsf{Ideal}}$ does). For clarity of presentation, we describe our functionalities without explicitly mentioning this but instead describe them as having direct access to time. Furthermore, we define the functionalities without specifying the corruption model as we will use the shells described in Section 3.3 to make the corruption-model explicit when implementing the functionalities.

Additionally, the behavior of both our functionalities depends on which parties are precorrupted and which are corrupted. Therefore they both maintain two sets: `Precorrupted` and `Corrupted`, which are initially empty. These are updated by the following activation rules, which we do not make explicit in the functionalities below for clarity of presentation.

Precorrupt: Upon receiving $(\mathit{Precorrupt}, p_i)$ or an initialization that changes party p_i's status to precorrupted, it sets $\texttt{Precorrupted} := \texttt{Precorrupted} \cup \{p_i\}$.

Corrupt: Upon receiving $(\mathit{Corrupt}, p_i)$ it sets $\texttt{Corrupted} := \texttt{Corrupted} \cup \{p_i\}$.

Furthermore, both of our ideal functionalities are parameterized by a message that can be propagated, which we denote Messages.

3.5.1 MessageTransfer

In this section, we present a basic functionality that allows a party to send messages to other parties. This is similar to the point-to-point channel presented in [Bau+21], but instead of hardcoding whether we assume AMS (as done in [Bau+21]) or not, we introduce an additional parameter which is the time an honest party needs to stay honest for ensuring delivery of the message.

Functionality $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_s, p_r)$

The functionality is parameterized by two parties p_s (the sender) and p_r (the receiver), and a time σ which parties need to stay honest for the delivery guarantee Δ to apply. It maintains a mailbox for p_r, $\texttt{Mailbox} : \textsc{Messages}$.

Initialize: Initially, $\texttt{Mailbox} := \varnothing$.

Send: After receiving (Send, m) from p_s it leaks (Leak, p_s, m) to the adversary.

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

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

At any time, the functionality automatically enforces the following property:

1. Let m be a message that is input for the first time by an honest party $p_s \notin \texttt{Corrupted}$ at some time τ. If $p_s \notin \texttt{Corrupted}$ at time $\tau + \sigma$, then by time $\tau + \Delta$ it is ensured that $m \in \texttt{Mailbox}$.

The property is ensured by the functionality automatically making the minimal possible additional calls with $\mathcal{SetMessage}$.

Note that building a construction using $\mathcal{F}^{0,\Delta}_{\mathsf{MessageTransfer}}$ exactly corresponds to assuming AMS whereas assuming $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}$ against a δ-delayed adversary with $\delta < \sigma$ corresponds to not assuming AMS.

3.5.2 Flood

The ideal functionality that we present here provides the guarantees of a flooding network, i.e., that all information some honest party knows is disseminated to all other parties within a bounded time.

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

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

Furthermore, it keeps track of a set of messages for each party $\texttt{Mailbox} : \mathcal{P} \to$ MESSAGES. These sets contain the messages that each party will receive after fetching.

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

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

Get Messages: After receiving ($\mathcal{GetMessages}$) from p_i it outputs $\texttt{Mailbox}[p_i]$ to party p_i.

Set Message: After receiving ($\mathcal{SetMessage}, m, p_i$) 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 two properties:

1. Let m be a message that is input for the first time to an honest party $p_i \notin \texttt{Precorrupted} \cup \texttt{Corrupted}$ at some time τ. By time $\tau + \Delta$ it is ensured that $\forall p_j \in \mathcal{P} \setminus (\texttt{Corrupted} \cup \texttt{Precorrupted})$ it holds that $m \in \texttt{Mailbox}[p_j]$.
2. Let m be a message at some time τ is in the mailbox of an honest party $p_i \notin \texttt{Precorrupted} \cup \texttt{Corrupted}$ i.e., $m \in \texttt{Mailbox}[p_i]$. By the time $\tau + \Delta$ it

is distributed to all honest mailboxes, i.e., for any party $p_j \in \mathcal{P} \setminus (\texttt{Corrupted} \cup \texttt{Precorrupted})$ it holds that $m \in \texttt{Mailbox}[p_j]$.

The functionality automatically ensures the properties, making the minimal possible additional calls with *SetMessage*.

3.6 Implementations of Flood

In this section we will present the following protocols that implement Flood:

$\Pi_{\mathsf{NaiveFlood}}$: Everybody simply sends to everybody.

Π_{ERFlood}: Everybody sends to each other party with some fixed probability ρ.

We provide two types of implementations for Flood. A naive approach where everybody sends to everybody and a more efficient one where each party sends to their neighbors with probability ρ. The latter construction allows us to reuse the theoretic foundation of Erdős–Rényi graphs in the distributed systems setting and achieve a variety of properties.

3.6.1 Naive Flood

We present here a protocol that implements Flood with a message complexity that is quadratic in the number of messages input to the system.

The protocol $\Pi_{\mathsf{NaiveFlood}}$ works straightforwardly by a peer sending and relaying any non-relayed message to all other parties. As everybody sends to everybody, the protocol achieves a minimal diameter and resilience against *relatively fast* adaptive adversaries at the cost of significant communication overhead and neighborhood.

Protocol $\Pi_{\mathsf{NaiveFlood}}$

Each pair of parties $p_i, p_j \in \mathcal{P}$ has access to a channel $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$. Each party $p_i \in \mathcal{P}$ keeps track of a set of relayed messages $\texttt{Relayed}_i$.

Initialize: Initially, all parties initialize their channel between them and set $\texttt{Relayed}_i := \varnothing$.

Send: When p_i receives (Send, m) they now forward inputs (Send, m) to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ for all $p_j \in \mathcal{P}$ and set $\texttt{Relayed}_i := \texttt{Relayed}_i \cup \{m\}$.

Get Messages: When p_i receives $(\mathit{GetMessages})$ they let M be the union of the messages they achieve by calling $(\mathit{GetMessages})$ to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ for all $p_j \in \mathcal{P}$, and outputs M.

Furthermore, once in every activation each honest p_i let M be the union of the messages they achieve by calling $(\mathit{GetMessages})$ to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$. For any $m \in M \setminus \texttt{Relayed}_i$, p_i inputs (Send, m) to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ for all $p_j \in \mathcal{P}$, and sets $\texttt{Relayed}_i := \texttt{Relayed}_i \cup \{m\}$.

An obvious attack on this protocol an adversary might try to perform is to try to corrupt the sender between the time τ that a message is sent and time $\tau + \sigma$ where the delivery guarantee from the underlying $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}$ applies. An adversary that succeeds with this can violate both Properties 1 and 2 of $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$. However, σ-delayed adversaries do not have sufficient time to succeed with this as the properties only need to be upheld for parties that are neither corrupted nor precorrupted when they try to send the message. Below we explicitly[2] prove that against such adversaries, the naive protocol realizes $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$.

Lemma 3.6.1. *Let $\sigma, \Delta \in \mathbb{N}$. The protocol $\Pi_{\mathsf{NaiveFlood}}$ perfectly realizes $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$ in the $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}$-hybrid model against a σ-delayed adversary.*

Proof. We construct a simulator $\mathcal{S}$.

1. $\mathcal{S}$ simulates all parties $p_i \in P$ inside it self.
2. When receiving (Leak, p_i, m) from $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$ the simulator inputs (Send, m) to p_i (running inside $\mathcal{S}$).
3. When receiving $(\mathit{SetMessage}, m)$ from the adversary on the port belonging to functionality $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$, $\mathcal{S}$ forwards $(\mathit{SetMessage}, m, p_j)$ to $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$.
4. Whenever $\mathcal{A}$ corrupts some $p_i \in \mathcal{P}$, $\mathcal{S}$ corrupts p_i and sends the simulated internal state to $\mathcal{A}$. From then on the simulated p_i (inside $\mathcal{S}$) follows $\mathcal{A}$'s instructions.
5. Whenever the $\bar{\mathcal{G}}_{\text{Ticker}}$ notifies $\mathcal{S}$ about the passing of time, $\mathcal{S}$ ensures to activate $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$.

As protocol, functionality, and simulator are all deterministic, it is enough to argue that the I/O behavior of $\mathcal{A}$ interacting with $\Pi_{\mathsf{NaiveFlood}}$ is equal to the I/O behavior of $\mathcal{S}$ interacting with $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$ to argue perfect indistinguishability. The send command is invoked at the exact same times in the real execution, and in the execution inside $\mathcal{S}$, this produces the same behavior. Furthermore, for any send command that is invoked at time τ by an honest party (neither precorrupted nor corrupted), there will be a set-message command within $\tau + \Delta$ for all honest parties in the real protocol as a σ-delayed adversary does not have time to violate the delivery property of the underlying $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$, and therefore Property 1 is upheld. Similarly, the relaying of messages in the real protocol ensures that messages will be delivered by the adversary according to the properties of $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$ in the real protocol (inside $\mathcal{S}$ and therefore also in the ideal) which ensures Property 2. $\square$

3.6.2 Efficient Flood

We now present a more efficient version of Flood. The idea is simple: Instead of relaying messages to *all* parties, each party flips a coin for each neighbor that decides

[2]In [Nie03, Chapter 3, p. 111], it is shown that it is enough to argue correct realization to achieve secure realization for any protocol which leaks all I/O behavior to the adversary. One may be led to believe that this result directly applies to $\Pi_{\mathsf{NaiveFlood}}$, but as $(\mathit{GetMessages})$ inputs (and corresponding outputs) are hidden from the adversary, this is not the case.

if a particular message should be forwarded to this party. Compared to the naive implementation of Flood presented in the previous section, the protocol presented here will have significantly smaller neighborhoods at the cost of a larger diameter in the communication graph (the parameter Δ of Flood). Furthermore, the construction can only tolerate adversaries slightly more delayed than those the naive protocol can tolerate.

The protocol Π_{ERFlood} works by letting all parties relay and send messages to a different random subset of parties for each message to be sent/relayed. By letting the random subset be large enough, we ensure that we establish a connected graph with a low diameter for all messages. As the subset of parties each party chooses to send to is random, the protocol achieves quite some robustness against adaptive adversaries, as a slightly delayed adversary cannot predict whom to corrupt to eclipse some specific parties.

Protocol $\Pi_{\mathsf{ERFlood}}(\rho)$

Each pair of parties $p_i, p_j \in \mathcal{P}$ has access to a channel $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$. Each party $p_i \in \mathcal{P}$ keeps track of a set of relayed messages $\texttt{Relayed}_i$: Messages.

Initialize: Initially, all parties initialize their channel between them and set $\texttt{Relayed}_i := \varnothing$.

Send: When p_i receives (Send, m), they input (Send, m) to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ with probability ρ for each party $p_j \in \mathcal{P}$. Finally they set $\texttt{Relayed}_i := \texttt{Relayed}_i \cup \{m\}$.

Get Messages: When p_i receives $(\mathit{GetMessages})$ they let M be the union of the messages they achieve by calling $(\mathit{GetMessages})$ to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ for all $p_j \in \mathcal{P}$, and outputs M.

Furthermore, once in each activation each honest p_i let M be the union of the messages they achieve by calling $(\mathit{GetMessages})$ to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ for all $p_j \in \mathcal{P}$. For any $m \in M \setminus \texttt{Relayed}_i$, p_i inputs (Send, m) to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ with probability ρ for all $p_j \in \mathcal{P}$, and sets $\texttt{Relayed}_i := \texttt{Relayed}_i \cup \{m\}$.

Depending on the parameter ρ, the protocol Π_{ERFlood} can achieve various properties. We provide two different instantiations that use the channel $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}$ and all works against a $(\sigma + \Delta)$-delayed adversary. Before going into detail with the actual proof, we provide some intuition for why the protocol is secure against exactly a $(\sigma + \Delta)$-delayed adversary. The central intuition is that such an adversary cannot influence how the communication graph between the honest parties is created. If a party decides to send a message at some time τ, then the set of parties that receives this message will have completed forwarding the message at time $\tau + \sigma + \Delta$, which is the earliest point on this party can be corrupted based upon this party's role in the specific communication graph. Therefore an adversary cannot use the adaptive corruptions to disrupt the propagation of a message.

Each of the instantiations presented below provides a trade-off between the graph's diameter, the neighborhood's average size, and the probability that the graph has these properties. Instantiation 1 ensures a diameter of 2 with a neighborhood of just $\Omega\left(\sqrt{n\kappa}\right)$ and Instantiation 2 ensures a logarithmic diameter with a neighborhood of average size $\Omega\left(\kappa\right)$.

Theorem 3.6.2. *Let $\Delta \in \mathbb{N}$ be any delay, let $\sigma \in \mathbb{N}$, let $t < n$ be the maximum number of parties an adversary can corrupt, and let $d \in \mathbb{R}$. The protocol $\Pi_{\mathsf{ERFlood}}(\rho)$ securely implements $\mathcal{F}_{\mathsf{Flood}}^{\Delta'}$ against a $(\sigma+\Delta)$-delayed adversary using $\mathcal{F}_{\mathsf{MessageTransfer}}^{\sigma,\Delta}$. More precisely, when r is an upper bound on the number of different messages input (either via Send or via SetMessage), the statistical distance between the real and ideal executions is bounded by the probability p_{bad} for either of the following instantiations:*

1. *Let $\rho := \sqrt{\frac{d}{h}}$ and let $\Delta' := 2\Delta$ then*

$$p_{bad} \leq r \cdot (t+1) \cdot n^2 \cdot e^{-d \cdot \frac{(h-2)}{h}}. \tag{3.43}$$

2. *Let $\alpha \in \mathbb{R}$, $\gamma, \delta_1, \delta_2 \in [0,1]$, and $\rho := \frac{d}{h}$. Furthermore, let $t_0 := \frac{\log\left(\frac{\gamma n}{(1-\delta_1)d}\right)}{\log((1-\delta_2)\alpha)} + 1$ and $\Delta' := \Delta \cdot (t_0 + 1)$. If*

$$e^{-d\gamma} + \frac{\gamma\alpha}{1-\gamma} \leq 1, \quad \frac{\gamma n}{(1-\delta_1)d} > 1, \quad \textit{and} \quad (1-\delta_2)\cdot\alpha > 1, \tag{3.44}$$

then

$$p_{bad} \leq r \cdot (t+1) \cdot \left(n \cdot \left(e^{-\frac{\delta_1^2 d}{2}} + t_0 e^{-\frac{\delta_2^2 \alpha(1-\delta_1)d}{2}}\right) + e^{-h\cdot(d\gamma^2-2)}\right). \tag{3.45}$$

Proof Sketch. For an adversary, we construct a simulator similar to how it is done in the proof of Lemma 3.6.1. The only times this is not a perfect simulation is when one of the properties of $\mathcal{F}_{\mathsf{Flood}}^{\Delta'}$ are violated in Π_{ERFlood}, which will never happen when the environment interacts with $\mathcal{F}_{\mathsf{Flood}}^{\Delta'}$. The main idea of the proof is to argue about the probability that a message m, that is input via either Send or SetMessage, is not propagated to all parties within Δ' time. We will argue about this via seven random experiments:

$\mathsf{FloodToER}_1$: An experiment where an adversary interacts with an oracle to learn edges in a directed graph. Only nodes that have an edge to them can have their edges revealed to the adversary, but the adversary can inject additional edges to be able to reveal more nodes. The adversary can remove up to t nodes, but at the point of removal, the adversary cannot have learned any edges connecting to the removed node. If there is a cut in the graph at any point, the adversary can stop the game.

$\mathsf{FloodToER}_2$: An experiment similar to $\mathsf{FloodToER}_1$ except now the edges are undirected.

$\mathsf{FloodToER}_3$: An experiment similar to $\mathsf{FloodToER}_2$ except the adversary cannot stop the game before all parties have been revealed.

$\mathsf{FloodToER}_4$: An experiment similar to $\mathsf{FloodToER}_3$ except the adversary cannot inject edges between parties.

$\mathsf{FloodToER}_5$: An experiment similar to $\mathsf{FloodToER}_4$ except that the oracle secretly and uniformly predetermines the size of the returned graph, $s \in \{h, \ldots, n\}$. The adversary can, however, still decide whether or not to remove a particular node, given that it does not violate the size that the oracle has determined.

$\mathsf{FloodToER}_6$: An experiment similar to $\mathsf{FloodToER}_5$ except now the oracle also predetermines a Erdős–Rényi graph of the predetermined size and embeds this into the final graph that is returned.

Erdős–Rényi: An experiment that chooses a graph of a certain size and includes each edge independently with probability ρ.

Let $\lambda := \frac{\Delta'}{\Delta}$. We now argue via the following steps:

1. If there is an adversary that prevents timely delivery of m in the real world with some probability, then there exists an adversary that can make $\mathsf{FloodToER}_1$ return a graph where the distance from the sender to some node is larger than λ with at least as high a probability.
2. If any adversary can make $\mathsf{FloodToER}_1$ return a graph with a diameter larger than λ with probability p, then there exists some adversary that can make $\mathsf{FloodToER}_2$ return a graph where the distance from the sender to some node is larger than λ with at least as high a probability.
3. If any adversary can make $\mathsf{FloodToER}_2$ return a graph with a diameter larger than λ with probability p, then there exists some adversary that can make $\mathsf{FloodToER}_3$ return a graph where the distance from the sender to some node is larger than λ with at least as high a probability.
4. If any adversary can make $\mathsf{FloodToER}_3$ return a graph with a diameter larger than λ with probability p, then there exists some adversary that can make $\mathsf{FloodToER}_4$ return a graph with a diameter larger than λ with at least as high a probability.
5. If any adversary can make the $\mathsf{FloodToER}_4$ game return a graph with a diameter larger than λ with probability p, then the same adversary can make $\mathsf{FloodToER}_5$ return a graph with a diameter larger than λ with probability at least $p \cdot (t + 1)$.
6. The experiments $\mathsf{FloodToER}_5$ and $\mathsf{FloodToER}_6$ are distributed identically.
7. The probability that $\mathsf{FloodToER}_6$ returns a graph with larger diameter than λ must be less than the probability that an Erdős–Rényi graph with the *worst* size has a larger diameter than λ.
8. We can now use the Erdős–Rényi graph results from Section 3.2.2 (in particular Lemmas 3.4.1 and 3.4.3) to bound the probability that an adversary can prevent the delivery of m in the real world.

We finally do a union bound over the number of different messages input to the functionality. The detailed proof can be found in Section 3.6.3. □

As the results in Theorem 3.6.2 are hard to interpret we additionally provide the following corollary which instantiates some of the many constants and makes some simplifying but non-optimal estimates. We emphasize that if one wants to optimize for a particular use-case (i.e., small diameter or very small failure probability) then Theorem 3.6.2 can be used to obtain tighter bounds.

As the results, in Theorem 3.6.2 are hard to interpret, we additionally provide the following corollary, which instantiates some of the many constants and makes some simplifying but non-optimal estimates. We emphasize that if one wants to optimize for a particular use-case (i.e., small diameter or tiny failure probability), then Theorem 3.6.2 can be used to obtain tighter bounds.

Corollary 3.6.3. *Let $\Delta \in \mathbb{N}$ be any delay, let $\sigma \in \mathbb{N}$, let $t < n$ be the maximum number of parties an adversary can corrupt, and let $d \in \mathbb{R}$ be the security parameter. The protocol $\Pi_{\mathsf{ERFlood}}(\rho)$ securely implements $\mathcal{F}_{\mathsf{Flood}}^{\Delta'}$ against a $(\sigma + \Delta)$-delayed adversary using $\mathcal{F}_{\mathsf{MessageTransfer}}^{\sigma,\Delta}$. More precisely, when r is an upper bound on the number of different messages input (either via $\mathscr{S}end$ or via $\mathscr{S}et\mathscr{M}essage$), the statistical distance between the real and ideal executions is bounded by the probability p_{bad} for either of the following instantiations:*

1. *Let $\rho := \sqrt{\frac{d}{h}}$ and let $\Delta' := 2\Delta$ then*
$$p_{bad} \leq r \cdot (t+1) \cdot n^2 \cdot e^{-d \cdot \frac{(h-2)}{h}}. \tag{3.46}$$

2. *Let $\rho := \frac{d}{h}$, and $\Delta' := \Delta \cdot \left(5 \log\left(\frac{n}{2d}\right) + 2\right)$, if $\frac{n}{2d} > 1$ then*
$$p_{bad} \leq r \cdot (t+1) \cdot \left(7n \log\left(\frac{n}{2d}\right) e^{-\frac{d}{18}} + e^{-\frac{h(d-18)}{9}}\right). \tag{3.47}$$

Proof. Instantiation 1 immediately follows from Theorem 3.6.2 (Instantiation 1). To derive instantiation Instantiation 2 we again use Theorem 3.6.2 (Instantiation 2) and select
$$\delta_1 := \delta_2 := \gamma := \frac{1}{3} \quad \text{and} \quad \alpha := \frac{7}{4}.$$
With these parameters, we see that Equation (3.44) is fulfilled when $d \geq 1$. Furthermore, we see that
$$\begin{aligned} p_{\text{bad}} &\leq r \cdot (t+1) \cdot \left(n \cdot \left(e^{-\frac{d}{18}} + \left(5 \log\left(\frac{n}{2d}\right) + 1\right) e^{-\frac{7d}{108}}\right) + e^{-\frac{h(d-18)}{9}}\right) \\ &\leq r \cdot (t+1) \cdot \left(7n \log\left(\frac{n}{2d}\right) e^{-\frac{d}{18}} + e^{-\frac{h(d-18)}{9}}\right). \end{aligned} \tag{3.48}$$
□

In Section 3.2.2.2 we provided Lemma 3.2.9, which shows that the number of neighbors any party will need to send to when they send/relay a message in $\Pi_{\mathsf{ERFlood}}(\rho)$ concentrates around $n \cdot \rho$. This follows from Chernoff and the union bound. Concretely, for Instantiation 1, to make the distinguishing probability negligible in a security parameter κ independent of the number of parties, we get that the number of neighbors needed is upper-bounded by $O\left(\sqrt{(\kappa + \log(n)) \cdot n}\right)$. For Instantiation 2, the number of necessary neighbors is upper-bounded by $O(\kappa + \log(n))$.

Remark 3.6.1. Within the UC framework, it is required that the number of parties is a polynomial of the security parameter. Therefore, using this, the above bounds on the number of parties could be simplified to just $O\big(\sqrt{\kappa \cdot n}\big)$ and $O(\kappa)$ respectively. However, to make the dependency on n explicit, we have left this simplification out.

A note on changing from TCP to UDP. Results about Erdős–Rényi graphs can be transferred to a setting without reliable message transmission. Let us, instead of reliable transmission, assume that there is an independent failure probability β for each message that is sent via $\mathcal{F}_{\mathsf{MessageTransfer}}$ and ρ is an instantiation of $\Pi_{\mathsf{ERFlood}}(\rho)$ that ensures a specific diameter assuming reliable transfer. If we let $\rho' := \frac{\rho}{1-\beta}$ then $\Pi_{\mathsf{ERFlood}}(\rho')$ with the unreliable transfer is ensured to have the same diameter as $\Pi_{\mathsf{ERFlood}}(\rho)$ with the reliable transfer. This is because the probability for a successful propagation from party p_i to p_j will then be $\rho' \cdot (1-\beta) = \rho$, which ensures that we in this more difficult setting inherent the original results for $\Pi_{\mathsf{ERFlood}}(\rho)$.

3.6.3 Reducing from Π_{ERFlood} to Erdős–Rényi Graphs

In this section, we prove Theorem 3.6.2. Our goal is to show that the probability that a message from Π_{ERFlood} is not delivered timely is only a tiny factor off from the probability that an Erdős–Rényi graph has a large diameter. This allows us to prove bounds on the delivery time for a message by porting results about the diameter of a Erdős–Rényi-graph, and we, therefore, believe that this is a technique of general interest.

We will show this by relating a series of random experiments via simulations. Our methodology for going from one game to the next is to construct a new adversary (using the old adversary), which has as good a probability of attacking the game as the old one has. A depiction of this approach can be found in Figure 3.3.

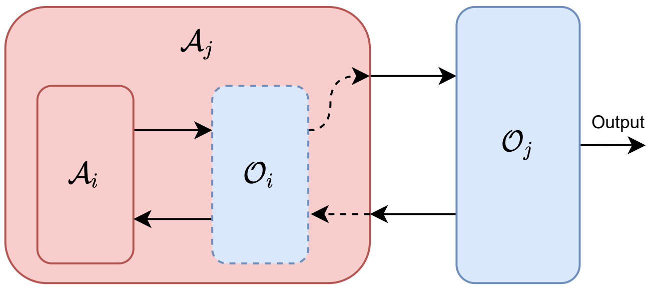


Figure 3.3: A depiction of the proof methodology for bounding that "something bad" happens when $\mathcal{A}_i$ interacts with the oracle $\mathcal{O}_i$ by the probability that there exists an $\mathcal{A}_j$ which can make "something bad" when interacting with $\mathcal{O}_j$. The idea is that $\mathcal{A}_j$ runs $\mathcal{A}_i$ inside itself while interacting with a simulated $\mathcal{O}_i$ which outputs are correlated with the outputs from the oracle $\mathcal{O}_j$.

In Table 3.1, we provide an overview of the different experiments we consider and which properties are bounded in the game.

Game	Description	Bounded Property
Π_{Gossip}	The real protocol.	Everybody receives a message no later than Δ' after it was sent.
$\mathsf{FloodToER}_1$	Directed graph with variable size, inject of edges, a specific sender, and early stopping.	Maximum distance from sender to any node is less than $\frac{\Delta'}{\Delta}$.
$\mathsf{FloodToER}_2$	Undirected graph with variable size, inject of edges, a specific sender, and early stopping.	Maximum distance from sender to any node is less than $\frac{\Delta'}{\Delta}$.
$\mathsf{FloodToER}_3$	Undirected graph with variable size, inject of edges, and a specific sender.	Maximum distance from sender to any node is less than $\frac{\Delta'}{\Delta}$.
$\mathsf{FloodToER}_3$	Undirected graph with variable size, inject of edges, and a specific sender.	Diameter of graph is less than $\frac{\Delta'}{\Delta}$.
$\mathsf{FloodToER}_4$	Undirected graph with variable size.	Any property.
$\mathsf{FloodToER}_5$	Undirected graph with a fixed size.	Monotone property.
$\mathsf{FloodToER}_6$	Undirected graph with a fixed size that explicitly embeds an Erdős–Rényi graph.	Any property.
Erdős–Rényi	Undirected Erdős–Rényi graph.	Monotone property that is preserved under renaming.

Table 3.1: Overview of the different games in our reduction to bound the statistical difference between the protocol $\Pi_{\mathsf{ERFlood}}(\rho)$ and the ideal functionality $\mathcal{F}^{\Delta'}_{\mathsf{Flood}}$ against a $(\sigma + \Delta)$-delayed adversary using $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}$. $\mathsf{FloodToER}_3$ appears twice as we, in separate lemmas, bound the distance from a particular sender by the graph's diameter.

3.6.3.1 Relating Games

We now present the games and prove several lemmas about how the different games relate. The first game we consider directly reflects an adversary's capabilities when a message has been input for the first time to a particular sender p_s in the protocol Π_{Gossip}.

Game $\mathsf{FloodToER}_1(\mathcal{P}, \rho, \mathcal{A}, p_s)$

The game is parameterized by a set of parties $\mathcal{P}$, an edge probability ρ, an adversary $\mathcal{A}$, and a node that is the original sender p_s. The adversary plays a game against an oracle, $\mathcal{O}_1$, which we define below.
$\mathcal{O}_1$ maintains five sets: a set of nodes that can be removed by the adversary `Killables`, a set of nodes that had their edge set revealed `Revealed`, a set of nodes that cannot be removed but have not yet had their edges revealed `Pending`,

a set of removed nodes $\texttt{Killed}$, and a set of directed edges $\texttt{Edges}$.
Initially, $\texttt{Revealed} := \varnothing$, $\texttt{Pending} := \{p_s\}$, $\texttt{Killables} := \mathcal{P} \setminus \{p_s\}$, $\texttt{Killed} := \varnothing$, and $\texttt{Edges} := \varnothing$. The oracle accepts the following inputs from the adversary:

Reveal: On input (Reveal, p_i) the oracle checks if $p_i \in \texttt{Pending}$ and otherwise ignores the input. The oracle now continues by adding p_i to $\texttt{Revealed}$ and removing p_i from $\texttt{Pending}$. Furthermore, it adds an edge (p_i, p_j) with probability ρ to $\texttt{Edges}$ for all $p_j \in \texttt{Pending} \cup \texttt{Killables} \cup \texttt{Revealed}$. Additionally, for any $p_j \in \texttt{Killables}$ it checks if $(p_i, p_j) \in \texttt{Edges}$ and if so moves p_j to $\texttt{Pending}$.

Finally, the set of edges is returned to the adversary.

Kill: On input (Kill, p_i), the oracle checks if $p_i \in \texttt{Killables}$ and if $|\texttt{Killed}| < t$. The oracle then removes p_i from $\texttt{Killables}$ and sets $\texttt{Killed} := \texttt{Killed} \cup \{p_i\}$. If not, the input is ignored.

Inject: On input $(\mathit{Inject}, p_i, p_j)$ the oracle checks if $p_i \in \texttt{Revealed}$ and $p_j \in \texttt{Killables}$. If that is the case it adds an edge (p_i, p_j) to $\texttt{Edges}$ and p_j is moved to $\texttt{Pending}$. If not, the input is ignored.

Stop: When receiving (Stop) the oracle checks if $\texttt{Pending} == \varnothing$. If that is the case, the oracle stops the game and returns $G = (\texttt{Revealed} \cup \texttt{Killables}, \texttt{Edges})$. If not, the input is ignored.

Next, we relate this game to the execution of the actual protocol, Π_{Gossip}.

Lemma 3.6.4. *Let $\Delta, \Delta', \sigma \in \mathbb{N}$, and $\rho \in [0,1]$. Let $\mathcal{A}$ be a $(\sigma + \Delta)$-delayed adversary, $\mathcal{Z}$ an environment, m a message that is input (either by the send command or by letting it be sent from some dishonest party) to some honest party P_s for the first time at time τ in the protocol $\Pi_{\mathsf{Gossip}}(\rho)$ using $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}$ against $\mathcal{A}$ and $\mathcal{Z}$.*

Let p_{bad} be the probability that some party at time $\tau + \Delta'$ is honest and has not yet received m. Furthermore let $\lambda := \frac{\Delta'}{\Delta}$. There exists an adversary $\mathcal{A}'$ s.t. if $G_{\mathsf{FloodToER}_1} \xleftarrow{\$} \mathsf{FloodToER}_1(\mathcal{P}, \rho, \mathcal{A}', p_s)$ then

$$p_{bad} \leq \Pr[\neg\phi_{Dist}(p_s, G_{\mathsf{FloodToER}_1}, \lambda)]. \tag{3.49}$$

Proof. The adversary $\mathcal{A}'$ simulates the execution of $\Pi_{\mathsf{Gossip}}(\rho)$ against $\mathcal{A}$, and $\mathcal{Z}$ inside its head and monitors the execution. $\mathcal{A}'$ maintains the same sets as $\mathcal{O}_1$ and updates them according to the outputs of $\mathcal{O}_1$.

- Initially, $\mathcal{A}'$ inputs (Kill, p_i) for all corrupted or precorrupted parties to the oracle.
- Whenever $\mathcal{A}$ inputs $\mathit{Precorrupt}$ to some party p_i at time τ' then $\mathcal{A}'$ checks if m is guaranteed to be delivered to p_i before time $\tau' + \Delta$. If that is not the case, then $\mathcal{A}'$ inputs (Kill, p_i) to the oracle.
- Whenever a party p_i which has not previously been removed from the graph is activated the first time after the message is delivered in one of their inboxes, then $\mathcal{A}'$ runs the following two ordered checks:

1. If $p_i \in$ `Killables` then $\mathcal{A}'$ finds the party p_j in `Revealed` that is furthest away from the sender p_s. It then inputs $(\mathit{Inject}, p_j, p_i)$.
2. If $p_i \in$ `Pending` then $\mathcal{A}'$ inputs (Reveal, p_i) to the oracle and receives a set of edges E. $\mathcal{A}'$ now makes p_i input (Send, m) to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ for any party p_j for which there is an edge in $(p_i, p_j) \in E$. Additionally, for any party $p_j \in$ `Killed`, (Send, m) is input to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ with probability ρ.

- If at any point `Pending` $== \varnothing$ then $\mathcal{A}'$ inputs Stop to the oracle.
- At the time $\tau + \Delta' + 1$, the adversary $\mathcal{A}'$ tries to finish the game by revealing all pending nodes continuously until the set of pending nodes is empty, and then it inputs Stop to the oracle.

Let $G_{\mathsf{FloodToER_1}} = (\mathcal{V}, E) \xleftarrow{\$} \mathsf{FloodToER_1}(\mathcal{P}, \rho, \mathcal{A}', p_s)$ and let the set of nodes that are honest at time $\tau + \Delta'$ be denoted $\mathcal{H}$.

First, we observe that all inputs $\mathcal{A}$ obtained from $\mathcal{A}'$ are distributed identically to those the adversary would see in a real execution of the protocol. It is, therefore, enough to argue that if there exists an honest party that has not received the message at time $\tau + \Delta'$ then $\neg\phi_{\mathsf{Dist}}(p_s, G_{\mathsf{FloodToER_1}}, \lambda)$.

We first observe the following invariant.

Claim 3.6.5. *Let p_i be a party that is added to* `Pending` *at time τ' then party p_i will at the latest be in* `Revealed` *at time $\tau' + \Delta$.*

Proof. Let us look at how p_i was added to `Pending`.

If $p_i = p_s$ then the claim is trivially true as the game must have started at τ' and the time won't progress before p_s is activated. Therefore $\mathcal{A}'$ will also input (Reveal, p_s) at time τ'.

Otherwise, p_i can only be added to `Pending` when an edge from a party p_j in the graph has been added to the set of edges. Let us make a case distinction on how this edge was added.

(Reveal, p_j)*:* When an edge is added by a reveal command, this makes p_j input (Send, m) to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ at time τ'. However, any party that has not been removed from the graph is not precorrupted early enough to prevent the party from forwarding the message to their neighbors, as $\mathcal{A}$ is $(\sigma + \Delta')$-delayed. Therefore, it is ensured that p_i at the latest will have m in its inbox at time $\tau' + \Delta$ which ensures that a (Reveal, p_i) will be input and p_i moved to `Revealed`.

$(\mathit{Inject}, p_j, p_i)$*:* This command only happens when p_i is immediately afterwards revealed. □

We now make the following claim and prove it.

Claim 3.6.6. *For any $r \in \mathbb{N}$ it holds that*

A) if a party $p_j \in \Gamma^r(p_s)$ then it has at the latest been revealed at time $\tau + r \cdot \Delta$;

B) and if the game has not stopped at time $\tau' := \tau + r \cdot \Delta$ then there exists some party p_j which has been revealed before or at τ' and is at least r distance away from the sender, i.e., $p_j \notin \Gamma^{r-1}(p_s)$. [3]

Proof. We prove this by induction in r. For $r = 0$ we have that $\Gamma^0(p_s) = \{p_s\}$, and that p_s is revealed at time τ by definition which ensures both Claim A) and B).

Let us now assume that the claim holds for r and let us prove that it also holds for $r + 1$. Let us first prove Claim B).. By the induction hypothesis, we know that there exists a party p_i that has been revealed before or at time $\tau + r \cdot \Delta$ and $p_i \notin \Gamma^{r-1}(p_s)$. The induction hypothesis also shows that all parties in $\Gamma^r(p_s)$ have been revealed before or at time $\tau + r \cdot \Delta$. Therefore any party that is revealed in the time-span $(\tau + r \cdot \Delta; \tau + (r+1) \cdot \Delta]$ cannot be in $\Gamma^r(p_s)$, and it suffices to show that some party is revealed in the time-span. Assume for the sake of contradiction that no party has been revealed in the time-span $(\tau + r \cdot \Delta; \tau + (r + 1) \cdot \Delta]$, this implies that either `Pending` $= \varnothing$ for the time-span which would make $\mathcal{A}'$ stop the game and we are done, or there is some party $p_j \in$ `Pending` at time $\tau + r \cdot \Delta$. However, Claim 3.6.5. guarantees that this party is revealed before or at time $\tau + (r + 1) \cdot \Delta$, which proves the claim.

We now turn our attention to prove Claim A). for the induction case. Let $p_i \in \Gamma^{r+1}(p_s)$ and let us consider two cases:

$p_i \in \Gamma^r(p_s)$*:* For this case the induction hypothesis gives us that p_i was revealed before $\tau + r \cdot \Delta$. As this is undoubtedly also before $\tau + (r + 1) \cdot \Delta$, we are done.

$p_i \in \Gamma^{r+1}(p_s) \setminus \Gamma^r(p_s)$*:* There must exist some party $p_j \in \Gamma^r$ s.t. $(p_j, p_i) \in E$. This edge may have been added by $\mathcal{A}'$ having issued one of the following two commands:

- $(\mathcal{Reveal}, p_j)$*:* As $p_j \in \Gamma^r$ the induction hypothesis ensures that this command is issued before or at time $\tau + r \cdot \Delta$. Moreover, note that this implies that $p_i \in$ `Pending` at time $\tau + r \cdot \Delta$. Claim 3.6.5. therefore ensures that p_i is revealed not later than $\tau + (r + 1) \cdot \Delta$.
- $(\mathit{Inject}, p_j, p_i)$*:* As $\mathcal{A}'$ always follows an inject up with a reveal, we are done if the inject happens before or at $\tau + (r + 1) \cdot \Delta$. If the inject happens after that time, then Claim B).[4] ensures that there exists some party p_v that has been revealed and is not in $\Gamma^{r-1}(p_s)$. However, $\mathcal{A}'$ only issues $(\mathit{Inject}, p_j, p_i)$ when p_j is the party furthest away from the sender. Therefore we can also conclude that $p_j \notin \Gamma^{r-1}(p_s)$ and therefore party p_i cannot be in $\Gamma^r(p_s)$ which is a contradiction to our original assumption. □

Let p_i be an honest party that has not received the message at time $\tau + \Delta'$ in the simulated execution. First, we observe that $\mathcal{H} \subseteq \mathcal{V}$, as only precorrupted nodes are ever killed. Hence, $p_i \in \mathcal{V}$. It therefore suffices to show that $p_i \notin \Gamma^\lambda(p_s)$ to show $\neg\phi_{\mathsf{Dist}}(p_s, G_{\mathsf{FloodToER}_1}, \lambda)$. We now make a case distinction on whether or not p_i has been revealed by $\mathcal{A}'$.

$p_i \notin$ `Revealed`*:* It must be that $p_i \in$ `Killables` when $\mathcal{A}'$ gave the input $\mathcal{Stop}$ to the oracle. Furthermore, only parties that have no incoming edges can be in `Killables`,

[3] We define $\Gamma^{-1}(p_i) := \varnothing$ for any p_i.

[4] Note that the argument is not cyclic as the proof of Claim B). does not rely on Claim A)..

as whenever an edge is added to a killable party, this party will be moved to `Pending` by the oracle. Therefore $p_i \notin \Gamma^{\lambda}(p_s)$.

$p_i \in$ `Revealed`*:* The time that p_i was revealed must be $\tau + \Delta' + 1$. The reason is that before this time, parties are only revealed by $\mathcal{A}'$ when they got m in their inbox in the protocol, which would contradict the assumption that the delivery guarantee was violated for p_i. However, Claim 3.6.6. A) ensures that any party in $\Gamma^{\lambda}(p_s)$ must have been revealed before time $\tau + \lambda \cdot \Delta$. To show that $p_i \notin \Gamma^{\lambda}(p_s)$ it is therefore suffices to show that $\Delta' + 1 \geq \lambda \cdot \Delta$. However, $\lambda \cdot \Delta = \frac{\Delta'}{\Delta} \cdot \Delta = \Delta'$ and the inequality is therefore trivially satisfied by definition of λ. □

Next, we present an undirected version of the same game but where edges are only added to all parties that have not been revealed before. Furthermore, an adversary can reveal any node and not just those with an edge to them in this game.

Game $\mathsf{FloodToER}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$

The game is parameterized by a set of parties $\mathcal{P}$, an edge probability ρ, an adversary $\mathcal{A}$, and a node that is the original sender p_s. The adversary plays a game against an oracle, $\mathcal{O}_2$, which we define below.
$\mathcal{O}_2$ maintains five sets: a set of nodes that can be removed by the adversary `Killables`, a set of nodes that had their edge set revealed `Revealed`, a set of nodes that cannot be removed but have not yet had their edges revealed `Pending`, a set of removed nodes `Killed`, and a set of *undirected* edges `Edges`.
Initially, `Revealed` $:= \varnothing$, `Pending` $:= \{p_s\}$, `Killables` $:= \mathcal{P} \setminus \{p_s\}$, `Killed` $:= \varnothing$, and `Edges` $:= \varnothing$. The oracle accepts the following inputs from the adversary:

Reveal: On input (Reveal, p_i) the oracle checks if $p_i \in$ `Pending`$\cup$`Killables` and otherwise ignores the input. The oracle now continues by adding p_i to `Revealed` and removes p_i from `Pending` or `Killables` (depending on where it originally was). Furthermore, it adds an edge $\{p_i, p_j\}$ with probability ρ to `Edges` for all $p_j \in$ `Pending`$\cup$`Killables`. Additionally, for any $p_j \in$ `Killables` it checks if $\{p_i, p_j\} \in$ `Edges` and if so moves p_j to `Pending`.

Finally, the set of edges is returned to the adversary.

Kill: On input (Kill, p_i), the oracle checks if $p_i \in$ `Killables` and if $|$`Killed`$| \leq t$. The oracle then removes p_i from `Killables` and sets `Killed` $:=$ `Killed`$\cup\{p_i\}$. If not, the input is ignored.

Inject: On input $(\mathit{Inject}, p_i, p_j)$ the oracle checks if $p_i \in$ `Revealed` and $p_j \in$ `Killables`. If that is the case adds an edge $\{p_i, p_j\}$ to `Edges` and then p_j is moved to `Pending`. If not, the input is ignored.

Stop: When receiving (Stop) the oracle checks if `Pending` $== \varnothing$. If that is the case, the oracle stops the game and returns $G = ($`Revealed` $\cup$ `Killables`, `Edges`$)$. If not, the input is ignored.

Using a simulation argument, we now relate $\mathsf{FloodToER}_1$ to $\mathsf{FloodToER}_2$.

Lemma 3.6.7. *Let $\mathcal{A}$ be an adversary, $p_s \in \mathcal{P}$, $\lambda \in \mathbb{N}$ and $\rho \in [0,1]$. Furthermore let $G_{\mathsf{FloodToER}_1} \overset{\$}{\leftarrow} \mathsf{FloodToER}_1(\mathcal{P}, \rho, \mathcal{A}, p_s)$. There exists an adversary $\mathcal{A}'$ s.t. if $G_{\mathsf{FloodToER}_2} \overset{\$}{\leftarrow} \mathsf{FloodToER}_2(\mathcal{P}, \rho, \mathcal{A}', p_s)$ then*

$$\Pr[\neg\phi_{Dist}(p_s, G_{\mathsf{FloodToER}_1}, \lambda)] \leq \Pr[\neg\phi_{Dist}(p_s, G_{\mathsf{FloodToER}_2}, \lambda)]. \tag{3.50}$$

Proof. Again we define $\mathcal{A}'$ in terms of $\mathcal{A}$ by letting $\mathcal{A}'$ play the role of an oracle when interacting with $\mathcal{A}$. We let $\texttt{Revealed}_2$, $\texttt{Pending}_2$, $\texttt{Killables}_2$, $\texttt{Killed}_2$ and $\texttt{Edges}_2$ be the sets maintained by $\mathcal{O}_2$, and let $\mathcal{A}'$ maintain similar sets indexed with 1 which will be presented to $\mathcal{A}$ similar to the sets the oracle $\mathcal{O}_1$ in $\mathsf{FloodToER}_1$ presents to $\mathcal{A}$.

The adversary additionally maintains a set $\texttt{HiddenEdges}$ which will be a set of edges that have been constructed in the game $\mathsf{FloodToER}_2$ but not yet revealed to $\mathcal{A}$, and a map, $\texttt{Flipped} : \mathcal{P} \times \mathcal{P} \to \mathbb{B}$, that keeps track of which edges that have already been added with probability ρ. Initially, $\texttt{Revealed}_1 := \varnothing$, $\texttt{Pending}_1 := \{p_s\}$, $\texttt{Killables}_1 := \mathcal{P} \setminus \{p_s\}$, $\texttt{Killed}_1 := \varnothing$, $\texttt{HiddenEdges} = \varnothing$ and $\texttt{Flipped}$ the empty map. Furthermore, we let Γ_1 be the neighborhood function for $G_{\mathsf{FloodToER}_1}$, and let Γ_2 be the neighborhood function for $G_{\mathsf{FloodToER}_2}$. Similarly, we subscript the distance function with either 1 or 2 to indicate which graph the function calculates the distance on.

- On input $(\mathcal{Reveal}, p_i)$ from $\mathcal{A}$ the adversary makes a set $E_{\texttt{Tmp}} := \varnothing$ and does the following:

  1. For $p_j \in \texttt{Pending}$ the adversary checks if the distance $\texttt{dist}_2(p_s, p_i)$ changes if the edge $\{p_i, p_j\}$ was added to $\texttt{Edges}_2$. If so it sets $\texttt{Flipped}(p_j, p_i) := \top$ and otherwise $\texttt{Flipped}(p_j, p_i) := \bot$.
  2. For $p_j \in \texttt{Pending}$, if $\texttt{Flipped}(p_j, p_i)$ the adversary $\mathcal{A}'$ sets $E_{\texttt{Tmp}} := E_{\texttt{Tmp}} \cup \{(p_i, p_j)\}$ with probability ρ.
  3. $\mathcal{A}'$ forwards $(\mathcal{Reveal}, p_i)$ to the oracle of $\mathsf{FloodToER}_2$. Let E be the set of edges returned on that request. Now, for every new edge, $\{p_i, p_j\} \in E$, the adversary does the following:
     - If $\texttt{Flipped}(p_j, p_i)$ then $\texttt{HiddenEdges} := \texttt{HiddenEdges} \cup \{(p_j, p_i)\}$.
     - Else the adversary adds an edge (p_i, p_j) to $E_{\texttt{Tmp}}$.
  4. The adversary $\mathcal{A}'$ assigns a new variable $E_{\texttt{Tmp}} := E_{\texttt{Tmp}} \cup \{(p_i, p_j) \mid p_j \in \mathcal{P} \wedge (p_i, p_j) \in \texttt{HiddenEdges}\}$.
  5. Now, for all $p_j \in \texttt{Revealed}$ where $\neg\texttt{Flipped}(p_i, p_j)$ an edge (p_i, p_j) is added to $E_{\texttt{Tmp}}$ with probability ρ.
  6. Finally, $\texttt{Edges}_1 := \texttt{Edges}_1 \cup E_{\texttt{Tmp}}$, the sets maintained by $\mathcal{A}'$ are updated similarly to how the oracle in $\mathsf{FloodToER}_1$ would have updated them, and the set $\texttt{Edges}_1$ is returned to $\mathcal{A}$.

- On input $(\mathcal{Inject}, p_i, p_j)$ the adversary forwards the request to the oracle. If the oracle adds an edge $\{p_i, p_j\}$ to $\texttt{Edges}_2$, then the adversary $\mathcal{A}'$ adds an edge (p_i, p_j) to $\texttt{Edges}_1$.

- All other inputs are forwarded directly to the oracle, and $\mathcal{A}'$ updates the sets accordingly.

Let us first prove that all inputs $\mathcal{A}$ obtained from $\mathcal{A}'$ are distributed identically to those the adversary would see when interacting with the oracle from game $\mathsf{FloodToER}_1$. We state this in the claim below.

Claim 3.6.8. *Let c be the number of commands executed by $\mathcal{A}'$ and let any set with superscript c denote the set after executing the c'th command. For any $c \in \mathbb{N}$ we have that* $\texttt{Revealed}_1^c = \texttt{Revealed}_2^c$, $\texttt{Pending}_1^c = \texttt{Pending}_2^c$, $\texttt{Killables}_1^c = \texttt{Killables}_2^c$, *and* $\texttt{Killed}_1^c = \texttt{Killed}_2^c$. *Moreover, the output $\mathcal{A}$ receives after inputting the c'th command to $\mathcal{A}'$ is identically distributed to the output $\mathcal{A}$ would have received when inputting the same c commands to* $\mathsf{FloodToER}_1$.

Proof. We prove this by induction in the number of commands. For the base case $c = 0$ we see that $\texttt{Pending}_1^0 = \texttt{Pending}_2^0 = \{p_s\}$, $\texttt{Killables}_1^0 = \texttt{Killables}_2^0 = \mathcal{P} \setminus \{p_s\}$, and all other sets are empty.

Let us now assume that the statement holds after having executed c' commands and show that it also holds after having executed $c = c' + 1$ commands. We make a case distinction based on the command given by $\mathcal{A}$:

$(\mathcal{Reveal}, p_i)$: The adversary, $\mathcal{A}$ expects that for any $p_j \in \texttt{Revealed}_1^{c'} \cup \texttt{Pending}_1^{c'} \cup \texttt{Killables}_1^{c'}$ the probability to see (p_i, p_j) should be ρ. We again analyse this case-wise based upon which set p_j was in just before command number c was executed:

$p_j \in \texttt{Killables}_1^{c'}$: As sets are synchronized we have that $p_j \in \texttt{Killables}_2^{c'}$. The oracle therefore returns an edge $\{p_i, p_j\}$ with probability ρ. If the oracle returns such an edge, the edge never changes the distance to party p_j as this is the first edge ever added to it, and therefore $\mathcal{A}'$ directly adds an edge (p_i, p_j) to $\texttt{Edges}_1$.

$p_j \in \texttt{Pending}_1^{c'}$: Let us calculate the probability to see $(p_i, p_j) \in E_{\texttt{Tmp}}$ just before $\texttt{Edges}_1$ is returned to $\mathcal{A}$. By the law of total probabilities for conditional events, we have that

$$\begin{aligned}
&\Pr[(p_i, p_j) \in E_{\texttt{Tmp}}] \\
&= \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \texttt{Flipped}(p_j, p_i)] \cdot \Pr[\texttt{Flipped}(p_j, p_i)] \\
&\quad + \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \neg\texttt{Flipped}(p_j, p_i)] \cdot \Pr[\neg\texttt{Flipped}(p_j, p_i)].
\end{aligned} \tag{3.51}$$

Let E be the edges returned by the oracle in $\mathsf{FloodToER}_2$. For any $p_j \in \texttt{Pending}$ with $\texttt{Flipped}(p_j, p_i)$ the adversary $\mathcal{A}'$ adds an edge (p_i, p_j) to $E_{\texttt{Tmp}}$ with probability ρ. If $\neg\texttt{Flipped}(p_j, p_i)$ then $(p_i, p_j) \in E_{\texttt{Tmp}}$ if and only if E. Therefore we have that

$$\begin{aligned}
&\Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \texttt{Flipped}(p_j, p_i)] \\
&= \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \neg\texttt{Flipped}(p_j, p_i)] \\
&= \Pr[(p_i, p_j) \in E] \\
&= \rho.
\end{aligned} \tag{3.52}$$

Furthermore, as $\Pr[\texttt{Flipped}(p_j, p_i)] + \Pr[\neg\texttt{Flipped}(p_j, p_i)] = 1$ we can conclude that the probability to see (p_i, p_j) in the outputted edges is ρ.

$p_j \in \texttt{Revealed}_1^{c'}$*:* This implies that there previously has been a reveal command $(\mathcal{Reveal}, p_j)$ that was input by $\mathcal{A}$. When this command was input was, the only time $\texttt{Flipped}(p_j, p_i)$ was changed. Let us calculate the probability to see $(p_i, p_j) \in E_{\texttt{Tmp}}$ just before $\texttt{Edges}_1$ is returned to $\mathcal{A}$. By the law of total probabilities for conditional events, we have that

$$\begin{aligned}
&\Pr[(p_i, p_j) \in E_{\texttt{Tmp}}] \\
&= \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \texttt{Flipped}(p_j, p_i)] \cdot \Pr[\texttt{Flipped}(p_j, p_i)] \\
&\quad + \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \neg\texttt{Flipped}(p_j, p_i)] \cdot \Pr[\neg\texttt{Flipped}(p_j, p_i)].
\end{aligned} \tag{3.53}$$

If $\texttt{Flipped}(p_j, p_i)$ then an edge (p_i, p_j) was added to $\texttt{HiddenEdges}$ with probability ρ when $(\mathcal{Reveal}, p_j)$ was given as input. Such an edge always ends in $E_{\texttt{Tmp}}$. On the other hand, if $\neg\texttt{Flipped}(p_j, p_i)$ then an edge (p_i, p_j) is directly added to $E_{\texttt{Tmp}}$ with probability ρ by $\mathcal{A}'$. Hence,

$$\begin{aligned}
&\Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \texttt{Flipped}(p_j, p_i)] \\
&= \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \neg\texttt{Flipped}(p_j, p_i)] \\
&= \rho.
\end{aligned} \tag{3.54}$$

Furthermore, as $\Pr[\texttt{Flipped}(p_j, p_i)] + \Pr[\neg\texttt{Flipped}(p_j, p_i)] = 1$ we can conclude that the probability to see (p_i, p_j) in the outputted edges is ρ.

$(\mathcal{Kill}, p_i)$*:* As the sets are in synchrony for c' commands any kill command that would be valid in $\mathsf{FloodToER}_1$ is also valid in $\mathsf{FloodToER}_2$. Therefore the sets stay synchronized.

$(\mathcal{Inject}, p_i, p_j)$*:* As the sets are in synchrony for c' commands any inject command that would be valid in $\mathsf{FloodToER}_1$ is also valid in $\mathsf{FloodToER}_2$. Therefore the sets stay synchronized.

$(\mathcal{Stop})$*:* This command doesn't change any of the involved sets.

□

Given the above claim it suffices to show that $\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToER}_1}, \lambda)$ implies $\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToER}_2}, \lambda)$. To show this we let $G_{\mathsf{FloodToER}_1} = (\mathcal{V}_1, E_1)$ and let $G_{\mathsf{FloodToER}_2} = (\mathcal{V}_2, E_2)$.

Now, let $p_i \in \mathcal{V}_1$ s.t. $p_i \notin \Gamma_1^{\lambda}(p_s)$ and let us show that $p_i \in \mathcal{V}_2$ and $p_i \notin \Gamma_2^{\lambda}(p_s)$. First, we observe that $\mathcal{V}_1 = \mathcal{V}_2$ as we at all times have that all sets are kept synchronized. This ensures that $P_i \in \mathcal{V}_2$. What is left is thus only to show that $p_i \notin \Gamma_2^{\lambda}(p_s)$. We show the lemma below.

Claim 3.6.9. *For any $k \in \mathbb{N}$ we have that $\Gamma_2^k(p_s) \subseteq \Gamma_1^k(p_s)$.*

Proof. We proceed by induction in k. For the base case, $k = 0$, we have that $\Gamma_1^0(p_s) = \Gamma_2^0(p_s) = \{p_s\}$. For the induction case, $k = k' + 1$, let p_i be party in $\Gamma_2^k(p_s)$. We make a case distinction:

$p_i \in \Gamma_2^{k-1}(p_s)$*:* By the induction hypothesis we have that $p_i \in \Gamma_1^{k-1}$. By definition we furthermore have $\Gamma_1^{k-1} \subseteq \Gamma_1^k$ which concludes the case.

$p_i \in \Gamma_2^k(p_s) \setminus \Gamma_2^{k-1}(p_s)$*:* There must exists some party $p_j \in \Gamma_2^{k-1}(p_s)$ where $\{p_j, p_i\} \in$ $\texttt{Edges}_2$ was the first edge that made $p_i \in \Gamma_2^k(p_s)$. Now note that by the induction hypothesis we have $p_j \in \Gamma_1^{k-1}(p_s)$. Let us distinguish how this edge was added to $\texttt{Edges}_2$:

- $(\mathcal{Reveal}, p_i)$*:* We note that if $\{p_j, p_i\}$ was added to $\texttt{Edges}_2$ with this command, then $p_j \in \texttt{Pending}_2$ when the command was executed. We note that if $\{p_i, p_j\}$ changes the distance from the sender to p_i, then it is saved in $\texttt{HiddenEdges}$ as the directed edge (p_j, p_i). Later, when p_j is revealed (which it must necessarily be at some point before the game is stopped), then we get that $(p_j, p_i) \in \texttt{Edges}_1$ and therefore that $p_j \in \Gamma_1^k(p_s)$.
- $(\mathcal{Reveal}, p_j)$*:* If this command added the edge, then as it was the first edge that made $p_i \in \Gamma_2^k(p_s)$ this cannot have changed the distance of p_j, and therefore it must be that $\neg\texttt{Flipped}(p_i, p_j)$. Therefore the edge (p_j, p_i) is also added to $\texttt{Edges}_1$ and again we get that $p_j \in \Gamma_1^k(p_s)$ by the induction hypothesis.
- $(\mathit{Inject}, p_j, p_i)$*:* This implies that $\mathcal{A}'$ adds (p_j, p_i) to $\texttt{Edges}_1$, which by the induction hypothesis implies that $p_j \in \Gamma_1^k(p_s)$.
- $(\mathit{Inject}, p_i, p_j)$*:* This command is only valid if p_j is in $\texttt{Killables}_2$. However, any party that is killable has no edges going to it. Therefore this contradicts that this is the *first* edge that was revealed that made $p_i \in \Gamma_2^k(p_s)$.

□

Now assume for the sake of contradiction that $p_i \in \Gamma_2^\lambda(p_s)$. Claim 3.6.9. implies that $p_i \in \Gamma_1^\lambda(p_s)$ which contradicts with the original assumption which was that $p_i \notin \Gamma_1^\lambda(p_s)$. □

Next, we present a version of the game that cannot be stopped before all notes are either revealed or killed.

Game $\mathsf{FloodToER}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$

The game is identical to $\mathsf{FloodToER}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$ except that the oracle in this game ignores *any* $(\mathcal{Stop})$ inputs from the adversary. All other inputs are treated identically to how the oracle from $\mathsf{FloodToER}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$ treats them.
When $\texttt{Killables} \cup \texttt{Pending} = \varnothing$ the game ends by the oracle returning $G = (\texttt{Revealed}, \texttt{Edges})$.

Lemma 3.6.10. *Let $\mathcal{A}$ be an adversary, $p_s \in \mathcal{P}$, $\lambda \in \mathbb{N}$ and $\rho \in [0,1]$. Furthermore let $G_{\mathsf{FloodToER}_2} \xleftarrow{\$} \mathsf{FloodToER}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$. There exists an adversary $\mathcal{A}'$ s.t. if $G_{\mathsf{FloodToER}_3} \xleftarrow{\$} \mathsf{FloodToER}_3(\mathcal{P}, \rho, \mathcal{A}', p_s)$ then*

$$\Pr[\neg\phi_{Dist}(p_s, G_{\mathsf{FloodToER}_2}, \lambda)] \leq \Pr[\neg\phi_{Dist}(p_s, G_{\mathsf{FloodToER}_3}, \lambda)]. \tag{3.55}$$

Proof. We define $\mathcal{A}'$ in terms of $\mathcal{A}$ by letting $\mathcal{A}'$ play the role of an oracle when interacting with $\mathcal{A}$.

- On input (Stop), $\mathcal{A}'$ checks if this stop command would make the oracle from $\mathsf{FloodToER}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$ end the game (i.e., if `Pending` $== \varnothing$). If that is the case, the adversary inputs (Reveal, p_i) for any party p_i that was in `Killables` at that time. Otherwise, the input is ignored.

- All other inputs and responses are forwarded directly between $\mathcal{A}$ and the oracle.

First, we observe that all inputs $\mathcal{A}$ receives from $\mathcal{A}'$ are distributed identically to those that the adversary would expect to see from the oracle in $\mathsf{FloodToER}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$. It therefore suffices to show that $\neg\phi_{\mathsf{Dist}}(p_s, G_{\mathsf{FloodToER}_2}, \lambda)$ implies $\neg\phi_{\mathsf{Dist}}(p_s, G_{\mathsf{FloodToER}_3}, \lambda)$. We make a case distinction based on if there has been any (Stop) input from $\mathcal{A}$ that is not ignored by $\mathcal{A}'$ before $\mathsf{FloodToER}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$ ends. If no such input is given, we have that $G_{\mathsf{FloodToER}_2} = G_{\mathsf{FloodToER}_3}$, and we are done.

If there is a (Stop) input from $\mathcal{A}$ that is not ignored by $\mathcal{A}'$ before $\mathsf{FloodToER}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$ ends, then there must have been a point in time, τ, where `Pending` $= \varnothing$ and `Killables` $\neq \varnothing$. Let $\texttt{Revealed}^\tau$ denote the set of revealed parties at this time, and $\texttt{Killables}^\tau$ denote the set of killable nodes at the time. There will not be any edges between any $p_i \in \texttt{Revealed}^\tau$ and any party $p_j \in \texttt{Killables}^\tau$ as the oracle does not add edges to parties that have already been revealed. As $\texttt{Killables}^\tau \neq \varnothing$ there exists some party $p_i \in \texttt{Killables}^\tau$ which will be a party of the final graph. The party p_s must however be in $\texttt{Revealed}^\tau$ as it is initially pending and $\texttt{Pending}^\tau = \varnothing$. This allows us to conclude that $\texttt{dist}(p_s, p_k) = \infty$, which completes the proof. □

That the distance from a specific node is less than some distance, λ, is a strictly weaker property than the diameter of a graph being less than λ. The below lemma follows immediately.

Lemma 3.6.11. *Let $\mathcal{A}$ be an adversary, $p_s \in \mathcal{P}$, $\lambda \in \mathbb{N}$, $\rho \in [0, 1]$, and let* $G_{\mathsf{FloodToER}_3} \overset{\$}{\leftarrow} \mathsf{FloodToER}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$. *We have that*

$$\Pr[\neg\phi_{\mathit{Dist}}(p_s, G_{\mathsf{FloodToER}_3}, \lambda)] \leq \Pr[\neg\phi_{\mathit{Diam}}(G_{\mathsf{FloodToER}_3}, \lambda)]. \tag{3.56}$$

Next, we present a version of the game where no specific node starts being pending, and the adversary can no longer inject additional edges into the graph.

Game $\mathsf{FloodToER}_4(\mathcal{P}, \rho, \mathcal{A})$

The game is identical to $\mathsf{FloodToER}_3$ except two things:

- Initially, `Pending` $:= \varnothing$ and `Killables` $:= \mathcal{P}$.

- The oracle ignores *any* $(\mathit{Inject}, p_i, p_j)$ inputs from the adversary.

All other inputs are treated identically to how the oracle from $\mathsf{FloodToER}_3$ treats them.

Before relating $\mathsf{FloodToER}_4(\mathcal{P}, \rho, \mathcal{A})$ to $\mathsf{FloodToER}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$ we define a special kind of graph properties called *monotone* properties.

Definition 3.6.12 (Monotone graph property)**.** Let $G = (\mathcal{V}, E)$ be a graph and ϕ be a property. We say that a property is *monotone* if for any additional set of edges $A \subseteq \mathcal{V} \times \mathcal{V}$ we have that

$$\phi(G) \implies \phi(\mathcal{V}, E \cup A).$$

Lemma 3.6.13. *Let $\mathcal{A}$ be an adversary, $p_s \in \mathcal{P}$, $\lambda \in \mathbb{N}$, $\rho \in [0,1]$, and let ϕ be a monotone graph property. Furthermore let $G_{\mathsf{FloodToER}_3} \overset{\$}{\leftarrow} \mathsf{FloodToER}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$. There exists an adversary $\mathcal{A}'$ s.t. if $G_{\mathsf{FloodToER}_4} \overset{\$}{\leftarrow} \mathsf{FloodToER}_4(\mathcal{P}, \rho, \mathcal{A}')$ then*

$$\Pr[\neg\phi(G_{\mathsf{FloodToER}_3})] \leq \Pr[\neg\phi(G_{\mathsf{FloodToER}_4})]. \tag{3.57}$$

Proof. We define $\mathcal{A}'$ in terms of $\mathcal{A}$ by letting $\mathcal{A}'$ play the role of an oracle when interacting with $\mathcal{A}$. We let $\texttt{Revealed}_4$, $\texttt{Pending}_4$, $\texttt{Killables}_4$, $\texttt{Killed}_4$ and $\texttt{Edges}_4$ be the sets maintained by the oracle in game $\mathsf{FloodToER}_4$, and let $\mathcal{A}'$ maintain similar sets indexed with 3 which will be presented to $\mathcal{A}$ similar to the sets the oracle in $\mathsf{FloodToER}_3$ presents to $\mathcal{A}$. Initially $\texttt{Pending}_3 := \{p_s\}$ and $\texttt{Killables}_3 = \mathcal{P} \setminus \{p_s\}$.

- On input $(\mathit{Inject}, p_i, p_j)$, $\mathcal{A}'$ checks if $p_i \in \texttt{Revealed}_3$ and $p_j \in \texttt{Killables}_3$. If that is the case the adversary adds $\{p_i, p_j\}$ to $\texttt{Edges}_3$ and moves p_j to $\texttt{Pending}_3$. Otherwise, the input is ignored.
- All other inputs are checked if they should be ignored with the sets indexed by 3 that are maintained by $\mathcal{A}'$ (using the rules the oracle uses in $\mathsf{FloodToER}_3$). If these rules should not ignore them, then inputs and responses are forwarded directly between $\mathcal{A}$ and the oracle.

We observe that all inputs $\mathcal{A}$ receives from $\mathcal{A}'$ are distributed identically to those that the adversary would expect to see from the oracle in $\mathsf{FloodToER}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$. We have this because, at any time during the execution, we have that $\texttt{Killables}_3 \subseteq \texttt{Killables}_4$ and therefore, any input that passes the check from the $\mathcal{A}'$ is a valid input to $\mathsf{FloodToER}_4$.

Moreover, we have that $\texttt{Edges}_4 \subseteq \texttt{Edges}_3$. Therefore if $\phi(G_{\mathsf{FloodToER}_4})$ then, as ϕ is monotone, $\phi(G_{\mathsf{FloodToER}_3})$. □

The game $\mathsf{FloodToER}_4(\mathcal{P}, \rho, \mathcal{A})$ has the property that the probability that any two nodes will have an edge between them given that they are in the final graph returned by the game is more than or equal to ρ. This is, however, not sufficient to be able to reduce the game to the Erdős–Rényi setting. The reason is that the adversary can dynamically choose the size of the graph. A hypothetical example of this being useful is when all but two nodes have been revealed, and the remaining two nodes are killable. If the adversary's goal is to obtain a graph with an isolated node, it is better to kill one and reveal the other than reveal/kill both.

To bound this advantage, we define an additional random experiment, which fixes the size of the random graph *a priori* and in particular without the adversary's influence.

Game $\mathsf{FloodToER}_5(\mathcal{P}, \rho, \mathcal{A})$

The game is identical to $\mathsf{FloodToER}_4$ except two things:

- Initially, the oracle makes a uniform guess on the size of the final graph,

$s \xleftarrow{\$} \mathcal{U}(\{h, \ldots, n\})$[a].

- If at any point in time either $|\texttt{Revealed}| + |\texttt{Pending}| > s$ or $n - |\texttt{Killed}| < s$, then the oracle sets $\texttt{Revealed} := \{p_1, \ldots, p_s\}$ and $\texttt{Edges} := \texttt{Revealed} \times \texttt{Revealed}$ before immediately ending the game by outputting $G = (\texttt{Revealed}, \texttt{Edges})$.[b]

All other behavior is identical to the oracle from $\mathsf{FloodToER}_4$.

[a] $\mathcal{U}(S)$ denotes the uniform distribution on a set S.
[b] If the game ends by this rule we say that the game ended by *quick quit*.

In this game, it is worth noticing that the game can only terminate by *quick quit* if the guess the oracle did on the size was incorrect. If the guess on the final size of the graph was, too low *quick quit* can be activated by the adversary directly revealing a node or by revealing a node which creates edges to a node that should be in the final kill set and cannot be moved away from there. If the guess on the final size of the graph was too high *quick quit* happens by the adversary trying to kill a party that cannot be swapped with any party in the final kill set.

Lemma 3.6.14. *Let ϕ be a graph property. For any adversary $\mathcal{A}$, let $G_{\mathsf{FloodToER}_4} \xleftarrow{\$} \mathsf{FloodToER}_4(\mathcal{P}, \rho, \mathcal{A})$, and let $G_{\mathsf{FloodToER}_5} \xleftarrow{\$} \mathsf{FloodToER}_5(\mathcal{P}, \rho, \mathcal{A})$. We have that*

$$\Pr[\phi(G_{\mathsf{FloodToER}_4})] \leq \Pr[\phi(G_{\mathsf{FloodToER}_5})] \cdot (t+1). \tag{3.58}$$

Proof. Let $\mathcal{O}_4$ and $\mathcal{O}_5$ be the oracles from the respective games.

The core idea of the proof is now to observe that given that $\mathcal{O}_5$ by luck made a good guess on the final size of the kill set, then the random experiments have equal distributions and that the adversary cannot influence whether or not the guess is correct.

Formally, we let $\texttt{GG}$ (abbreviating "good guess") be the event that $\mathcal{O}_5$ did not terminate by *quick quit* (corresponding to that the guess on the final size of the graph was "correct"). By the law of total probabilities, we have

$$\begin{aligned}\Pr[\phi(G_{\mathsf{FloodToER}_5})] &= \Pr[\phi(G_{\mathsf{FloodToER}_5}) \mid \texttt{GG}] \cdot \Pr[\texttt{GG}] + \Pr[\phi(G_{\mathsf{FloodToER}_5}) \mid \neg\texttt{GG}] \cdot \Pr[\neg\texttt{GG}] \\ &\geq \Pr[\phi(G_{\mathsf{FloodToER}_5}) \mid \texttt{GG}] \cdot \Pr[\texttt{GG}].\end{aligned} \tag{3.59}$$

We now note that the adversary cannot influence the probability that a guess is correct, as the behavior of $\mathcal{O}_5$ is precisely equal to the behavior of $\mathcal{O}_4$ until the game terminates. When this happens, it is too late for the adversary to influence. Therefore, as $\mathcal{O}_5$'s guess on the size of the final kill set is picked uniformly at random, the probability of seeing $\texttt{GG}$ is

$$\Pr[\texttt{GG}] = (t+1)^{-1}. \tag{3.60}$$

By Equations (3.59) and (3.60) it is sufficient to show that

$$\Pr[\phi(G_{\mathsf{FloodToER}_5}) \mid \texttt{GG}] = \Pr[\phi(G_{\mathsf{FloodToER}_4})]. \tag{3.61}$$

Now, note that if GG then the outputs that $\mathcal{O}_5$ provides to $\mathcal{A}$ are distributed exactly as the outputs of $\mathcal{O}_4$. Therefore the adversary's inputs must be identically distributed as well. □

Game $\mathsf{FloodToER}_6(\mathcal{P}, \rho, \mathcal{A})$

The game is parameterized by a set of parties $\mathcal{P}$, an edge probability ρ, and an adversary $\mathcal{A}$. The adversary plays a game against an oracle which we define below. The oracle maintains five sets: a set of nodes that can be removed by the adversary `Killables`, a set of nodes that had their edge set revealed `Revealed`, a set of nodes that cannot be removed but have not yet had their edges revealed `Pending`, a set of removed nodes `Killed`, and a set of undirected edges `Edges`.
Before the game starts, the oracle makes a uniform guess on the size of the final graph, $s \xleftarrow{\$} \mathcal{U}(\{h, \dots, n\})$, and then samples a graph $G_s = (\mathcal{V}, E) \xleftarrow{\$} \mathbb{G}(s, \rho)$.
Initially, $\texttt{Revealed} := \varnothing$, $\texttt{Pending} := \varnothing$, $\texttt{Killables} := \mathcal{P}$, $\texttt{Killed} := \varnothing$, $\texttt{Edges} := \varnothing$. Additionally, the oracle maintains a map $\texttt{Naming} : \mathcal{P} \to \{1, \dots, s\}$ and a set of available names `AvailableNames`. The map starts out being empty and $\texttt{AvailableNames} := \{1, \dots, s\}$.
The oracle accepts the following inputs from the adversary:

Reveal: On input $(\mathcal{R}eveal, p_i)$ the oracle checks if $p_i \in \texttt{Pending} \cup \texttt{Killables}$ and otherwise ignores the input. If $p_i \in \texttt{Pending} \cup \texttt{Killables}$ then the oracle does the following:

1. If $p_i \in \texttt{Killables}$ then the oracle samples $\eta \xleftarrow{\$} \mathcal{U}(\texttt{AvailableNames})$, sets $\texttt{Naming}[p_i] := \eta$, and sets $\texttt{AvailableNames} := \texttt{AvailableNames} \setminus \{\eta\}$.
2. The oracle now continues by adding p_i to `Revealed` and removes p_i from `Pending` or `Killables` (depending on where it originally was).
3. Now, for any $e \in \{\{\texttt{Naming}[p_i], j\} \in E \mid j \in \{1, \dots, s\}\}$ let $e = \{\texttt{Naming}[p_i], j\}$ for some j and do the following:
   a) If $\texttt{Naming}^{-1}[j] == \bot$ then the oracle samples $p \xleftarrow{\$} \mathcal{U}(\texttt{Killables})$, sets $\texttt{Naming}[p] := j$, and sets $\texttt{AvailableNames} := \texttt{AvailableNames} \setminus \{j\}$, and moves p from `Killables` to `Pending`.
   b) Set $\texttt{Edges} := \texttt{Edges} \cup \{\{p_i, \texttt{Naming}^{-1}[j]\}\}$.
4. Now for each $\{1, \dots, n - s - |\texttt{Killed}|\}$, the oracle flips a coin that comes out head with probability ρ. If head, then the oracle samples a party $p \xleftarrow{\$} \mathcal{U}(\texttt{Killables})$ and a name $\eta \xleftarrow{\$} \mathcal{U}(\texttt{AvailableNames})$, removes η from `AvailableNames`, updates the naming $\texttt{Naming}[p] := \eta$, moves p from `Killables` to `Pending` and adds an edge $\{p_i, p\}$ to `Edges`.
5. Finally, the set of edges is returned to the adversary.

Kill: On input $(\mathcal{K}ill, p_i)$, the oracle checks if $p_i \in \texttt{Killables}$ and if $|\texttt{Killed}| \le t$. The oracle then removes p_i from `Killables` and sets $\texttt{Killed} := \texttt{Killed} \cup \{p_i\}$. If not, the input is ignored.

If at any point in time, either $|\texttt{Revealed}|+|\texttt{Pending}| > s$ or $n-|\texttt{Killed}| < s$ or the oracle tries to make a draw from an empty set, then the oracle sets $\texttt{Revealed} \coloneqq \{p_1, \dots, p_s\}$ and $\texttt{Edges} \coloneqq \texttt{Revealed} \times \texttt{Revealed}$ before immediately ending the game by outputting $G = (\texttt{Revealed}, \texttt{Edges})$.
When $\texttt{Killables} \cup \texttt{Pending} = \varnothing$ the game ends by the oracle returning $G = (\texttt{Revealed}, \texttt{Edges})$.

Lemma 3.6.15. *Let $\rho \in [0,1]$. For any adversary $\mathcal{A}$,*

$$\mathsf{FloodToER}_5(\mathcal{P}, \rho, \mathcal{A}) \approx \mathsf{FloodToER}_6(\mathcal{P}, \rho, \mathcal{A}). \tag{3.62}$$

Proof. Let $\mathcal{O}_5$ be the oracle from $\mathsf{FloodToER}_5$ and let $\mathcal{O}_6$ be the oracle from $\mathsf{FloodToER}_6$. We let $\texttt{Revealed}_5$, $\texttt{Pending}_5$, $\texttt{Killables}_5$, $\texttt{Killed}_5$ and $\texttt{Edges}_5$ be the sets maintained by the $\mathcal{O}_5$, and let similarly named sets indexed with 6 be the one maintained by $\mathcal{O}_6$. Moreover, let $G = (\mathcal{V}, E)$ be the Erdős–Rényi graph that $\mathcal{O}_6$ holds. Let c be the number of commands input by $\mathcal{A}$ and let any set with superscript c denote the set after executing the c'th command.

We now state and prove two claims needed for the main result.

Claim 3.6.16. *For any $c \in \mathbb{N}$ if $\mathcal{O}_6$ did not stop early we have that $\forall p_i \in \texttt{Pending}_6^c \cup \texttt{Revealed}_6^c, \texttt{Naming}^c[p_i] \in \mathcal{V}$.*

Proof. The claim follows by induction in c as each time a party is moved from $\texttt{Killables}_6$ to $\texttt{Revealed}_6$ a name is assigned to the party, and each time a party is moved from $\texttt{Killables}$ to $\texttt{Pending}$ it is also assigned a name. □

Claim 3.6.17. *For any $c \in \mathbb{N}$ if $\mathcal{O}_6$ did not stop early we have that $|\texttt{AvailableNames}^c| = s - (|\texttt{Pending}_6^c| + |\texttt{Revealed}_6^c|)$.*

Proof. We do induction in c. For the base case $c = 0$, we have that

$$|\texttt{AvailableNames}^0| = |\{1, \dots, s\}| = s. \tag{3.63}$$

We now argue about the induction case $c = c' + 1$. As $(\mathcal{K}ill, p_i)$ inputs don't change any of the sets, it is sufficient to argue about the case where the c'th command is of the form $(\mathcal{R}eveal, p_i)$. If $p_i \notin \texttt{Pending}_6^{c'} \cup \texttt{Killables}_6^{c'}$, the command is ignored, and none of the sets changes. We make a case distinction on the remaining two possibilities:

$p_i \in \texttt{Pending}_6^{c'}$*:* First p_i is moved from pending parties to revealed parties which does not change the invariant. Afterward, for each party, p_j that is moved from killable parties to pending parties, a name is also removed from the available names and thus maintains the invariant.

$p_i \in \texttt{Killables}_6^{c'}$*:* First p_i is assigned a name (which decreases the number of available names), and next it is moved from killables to revealed parties. This maintains the invariant. The remaining operations are similar to the case $p_i \in \texttt{Pending}_6^{c'}$.

□

Claim 3.6.18. *For any $c \in \mathbb{N}$ we have that if neither $\mathcal{O}_5$ nor $\mathcal{O}_6$ stopped early, then* $\texttt{Revealed}_1^c = \texttt{Revealed}_2^c$, $\texttt{Pending}_1^c = \texttt{Pending}_2^c$, $\texttt{Killables}_1^c = \texttt{Killables}_2^c$, *and* $\texttt{Killed}_1^c = \texttt{Killed}_2^c$. *Moreover, the output $\mathcal{A}$ receives in the two games is identically distributed.*

Proof. We do induction in the number of commands. For the base case, $c = 0$, the claim is trivially true as all sets are initialized to be the same. Let us now assume the claim holds for $c' \in \mathbb{N}$ and let us show that it holds for $c = c' + 1$. If the command input is $(\mathcal{Kill}, p_i)$, then the claim follows by the induction hypothesis. It is, therefore, sufficient to look at the case when $(\mathcal{Reveal}, p_i)$ is input. Furthermore, as sets are updated identically based on the output of the command for both oracles, it is sufficient to argue about the output distribution of the edges. We make a case distinction based upon where p_i is located before the command is executed:

$p_i \in \texttt{Revealed}_5^{c'}$*:* By the induction hypothesis we have that $p_i \in \texttt{Revealed}_6^{c'}$ and therefore $\mathcal{O}_5$ and $\mathcal{O}_6$ both ignore the command.

$p_i \in \texttt{Pending}_5^{c'}$*:* By the induction hypothesis we have that $p_i \in \texttt{Pending}_6^{c'}$. $\mathcal{O}_5$ adds an edge with probability ρ to all parties in $\texttt{Pending}_5^{c'} \cup \texttt{Killables}_5^{c'} \setminus \{p_i\}$ and moves all parties that had an edge added to it to $\texttt{Pending}_5^c$. Let us now argue that $\mathcal{O}_6$ does the same. We let $p_j \in \texttt{Pending}_5^{c'} \cup \texttt{Killables}_5^{c'} \setminus \{p_i\}$ and do a case distinction to show that $\Pr[\{p_i, p_j\} \in \texttt{Edges}_6^c] = \rho$.

$p_j \in \texttt{Pending}_5^{c'}$*:* By the induction hypothesis we have that $p_j \in \texttt{Pending}_6^{c'}$. Furthermore, Claim 3.6.16. ensures that $\texttt{Naming}[p_j] \in \mathcal{V}$. As G is an Erdős–Rényi graph there is a probability of ρ that $\{\texttt{Naming}[p_i], \texttt{Naming}[p_j]\} \in E$ and if it is the case it then $\{p_i, p_j\}$ will be added to $\texttt{Edges}_6^c$.

$p_j \in \texttt{Killables}_5^{c'}$*:* By the induction hypothesis we have that $p_j \in \texttt{Killables}_6^{c'}$. There are two different possibilities for an edge $\{p_i, p_j\}$ to be added to $\texttt{Edges}$. It can either be added based upon the edges from the underlying Erdős–Rényi graph that goes to a node that has not yet had assigned an edge, or the additional flips can add it afterward. For any node $v \in \texttt{AvailableNames}$ (which is precisely those that have not yet have a name assigned), we see that there is a probability of $\Pr[\texttt{Naming}[p_i], v] = \rho$. If this edge exists v is uniformly assigned to a node from $\texttt{Killables}_6^{c'}$ and $\texttt{Killables}_6$ is updated. With the additional $n - s - |\texttt{Killed}_6^{c'}|$ coin flips for extra edges that are mapped in the same way afterward, we have in total $|\texttt{AvailableNames}| + (n - s - |\texttt{Killed}_6^{c'}|)$ such random variables that might result in an edge $\{p_i, p_j\}$. By Claim 3.6.17. we have that

$$\begin{aligned}
&|\texttt{AvailableNames}| + (n - s - |\texttt{Killed}_6^{c'}|) \\
&= s - (|\texttt{Revealed}_6^{c'} + |\texttt{Pending}_6^{c'}||) + n + s - |\texttt{Killed}_6^{c'}| \\
&= n - |\texttt{Revealed}_6^{c'}| - |\texttt{Pending}_6^{c'}| - |\texttt{Killed}_6^{c'}| \\
&= |\texttt{Killables}_6^{c'}|.
\end{aligned} \tag{3.64}$$

As there are exactly $|\texttt{Killables}_6^{c'}|$ such random draws and for any outcome of the draws, a uniform bijective mapping is constructed into $\texttt{Killables}_6^{c'}$ we can conclude that $\Pr[\{p_i, p_j\} \in \texttt{Edges}_6^c] = \rho$.

$p_i \in \texttt{Killables}_5^{c'}$*:* Follows by a similar argument to the case $p_i \in \texttt{Pending}_5^{c'}$.

$p_i \in \texttt{Killed}_5^{c'}$*:* By the induction hypothesis we have that $p_i \in \texttt{Killed}_6^{c'}$ and therefore $\mathcal{O}_5$ and $\mathcal{O}_6$ both ignore the command.

□

Claim 3.6.18. ensures that the outputs of the two oracles are synchronized until one of them stops early. If neither stops, we are done. It is, therefore, sufficient to argue that $\mathcal{O}_5$ only stops early if and only if $\mathcal{O}_6$ stops early. However, as the sets are synchronized until one stops (Claim 3.6.18.) and $\mathcal{O}_6$ rules for early stopping is a superset of the rules $\mathcal{O}_5$ has for early stopping, it is sufficient to argue that if $\mathcal{O}_6$ stops by any of the additional stopping rules then it would anyway stop by one of the other rules (and thereby $\mathcal{O}_5$ would also stop).

By Claim 3.6.17., the set of available names is only empty when $|\texttt{Revealed}_6| + |\texttt{Pending}_6| = s$, and therefore $\mathcal{O}_6$ only tries to draw a name from the empty set if it would anyway right after having that $|\texttt{Revealed}_6| + |\texttt{Pending}_6| > s$.

Let us now argue that $\mathcal{O}_6$ only makes a draw from an empty set of killable if $n - |\texttt{Killed}_6| < s$. Let us assume that $\texttt{Killables} = \varnothing$ and $n - |\texttt{Killed}_6| \geq s$. We have that:

$$\begin{aligned} & n - |\texttt{Killed}_6| \\ & = |\texttt{Revealed}_6| + |\texttt{Pending}_6| + |\texttt{Killables}_6| + |\texttt{Killed}_6| - |\texttt{Killed}_6| \\ & = |\texttt{Revealed}_6| + |\texttt{Pending}_6|. \end{aligned} \tag{3.65}$$

However, as $\texttt{Naming}$ is injective, we have by Claim 3.6.16. that $|\texttt{Revealed}_6|+|\texttt{Pending}_6| \leq s$ which combined with our assumption gives us that $|\texttt{Revealed}_6| + |\texttt{Pending}_6| = s$. Moreover, as $\texttt{Naming}$ is injective there cannot be any nodes $j \in \mathcal{V}$ for which $\texttt{Naming}^{-1} == \perp$ and therefore there are no draws from $\texttt{Killables}$ when $\mathcal{O}_6$ samples edges from E. Finally, we have that $n - s - |\texttt{Killed}| = 0$, so the oracle does not flip any additional coins, which ensures that there are no draws from $\texttt{Killables}_6$. □

Let us now relate the probability that an adversary can produce a graph with an undesirable property in the *FloodToErdosSix* game to the probability that an Erdős–Rényi graph has the property. However, this is only true for a restricted class of properties we define below.

Definition 3.6.19 (Properties preserved under renaming of nodes)**.** Let $G = (\mathcal{V}, E)$ be a graph. We say that ϕ is *preserved under renaming of nodes* if for any injective function, $\texttt{r} : \mathcal{V} \to \mathcal{V}'$, we have that

$$\phi(G) \implies \phi((\{\texttt{r}(v) \mid v \in \mathcal{V}\}, \{\{\texttt{r}(v), \texttt{r}(u)\} \mid \{v, u\} \in E\})).$$

Lemma 3.6.20. *Let ϕ be a monotone graph property preserved under renaming and let $G_s \xleftarrow{\$} \mathbb{G}(s, \rho)$ for any $s \in \mathbb{N}$. For any adversary $\mathcal{A}$ let $G_{\mathsf{FloodToER}_6} \xleftarrow{\$} \mathsf{FloodToER}_6(\mathcal{P}, \rho, \mathcal{A})$ then*

$$\Pr[\neg\phi(G_{\mathsf{FloodToER}_6})] \leq \max_{s \in \{h, \ldots, n\}} \Pr[\neg\phi(G_s)]. \tag{3.66}$$

Proof. Let $G = (\mathcal{V}, E) \overset{\$}{\leftarrow} \mathsf{FloodToER}_6(\mathcal{P}, \rho, \mathcal{A})$. By the law of total probabilities, we have

$$\begin{aligned}\Pr[\neg\phi(G)] &= \sum_{s=h}^{n} \Pr[\neg\phi(G) \mid s = |G|] \cdot \Pr[s = |G|] \\ &\leq \left(\max_{s\in\{h,\ldots,n\}} \Pr[\neg\phi(G) \mid s = |G|] \right) \cdot \sum_{s=h}^{n} \Pr[s = |G|] \\ &= \max_{s\in\{h,\ldots,n\}} \Pr[\neg\phi(G) \mid s = |G|].\end{aligned} \tag{3.67}$$

Let $s_0 := \operatorname{argmax}_{s\in\{h,\ldots,n\}} \Pr[\neg\phi(G) \mid s = |G|]$. As

$$\max_{s\in\{h,\ldots,n\}} \Pr[\neg\phi(G_s)] \geq \Pr[\neg\phi(G_{s_0})], \tag{3.68}$$

it suffices to show that

$$\Pr[\neg\phi(G_{s_0})] \geq \Pr[\neg\phi(G) \mid s_0 = |G|]. \tag{3.69}$$

As $\Pr[\neg\phi(G_{s_0})] = 1 - \Pr[\phi(G_{s_0})]$ and $\Pr[\neg\phi(G) \mid s_0 = |G|] = 1 - \Pr[\phi(G) \mid s_0 = |G|]$ it suffices to show:

$$\Pr[\phi(G_{s_0})] \leq \Pr[\phi(G) \mid s_0 = |G|]. \tag{3.70}$$

Let $G' = (\mathcal{V}', E')$ be the Erdős–Rényi graph maintained internally by the oracle in $\mathsf{FloodToER}_6$. Clearly, $\Pr[\phi(G_{s_0})] = \Pr[\phi(G') \mid s_0 = |G|]$, as then G_{s_0} are actually drawn from the same distribution. It is, therefore, sufficient to show that $\phi(G') \implies \phi(G)$.

WLOG assume that there is some graph of size s_0 that has the property (otherwise Equation (3.70) is trivially fulfilled as both sides are equal to 0). Therefore, if the game ended early, then $\phi(G)$ as the entire graph is returned. Otherwise, if the game did not end early, then let $G'' = (\mathcal{V}'', E'') = (\mathtt{Naming}^{-1}(\mathcal{V}), \mathtt{Naming}^{-1}(E))$. As ϕ is preserved under renaming, we have that $\phi(G'')$. Furthermore, $\mathcal{V}'' = \mathcal{V}$ and $E'' \subseteq E$, which ensures $\phi(G)$ (as ϕ is monotone). □

3.6.3.2 Proving Π_{Gossip} Secure

We are now finally ready to prove our main result.

Theorem 3.6.2. *Let $\Delta \in \mathbb{N}$ be any delay, let $\sigma \in \mathbb{N}$, let $t < n$ be the maximum number of parties an adversary can corrupt, and let $d \in \mathbb{R}$. The protocol $\Pi_{\mathsf{ERFlood}}(\rho)$ securely implements $\mathcal{F}_{\mathsf{Flood}}^{\Delta'}$ against a $(\sigma + \Delta)$-delayed adversary using $\mathcal{F}_{\mathsf{MessageTransfer}}^{\sigma,\Delta}$. More precisely, when r is an upper bound on the number of different messages input (either via $\mathscr{Send}$ or via $\mathscr{SetMessage}$), the statistical distance between the real and ideal executions is bounded by the probability p_{bad} for either of the following instantiations:*

1. *Let $\rho := \sqrt{\frac{d}{h}}$ and let $\Delta' := 2\Delta$ then*

$$p_{bad} \leq r \cdot (t+1) \cdot n^2 \cdot e^{-d \cdot \frac{(h-2)}{h}}. \tag{3.43}$$

2. *Let $\alpha \in \mathbb{R}$, $\gamma, \delta_1, \delta_2 \in [0,1]$, and $\rho := \frac{d}{h}$. Furthermore, let $t_0 := \frac{\log\left(\frac{\gamma n}{(1-\delta_1)d}\right)}{\log((1-\delta_2)\alpha)} + 1$ and $\Delta' := \Delta \cdot (t_0 + 1)$. If*

$$e^{-d\gamma} + \frac{\gamma\alpha}{1-\gamma} \leq 1, \qquad \frac{\gamma n}{(1-\delta_1)d} > 1, \quad \textit{and} \quad (1-\delta_2)\cdot\alpha > 1, \tag{3.44}$$

then

$$p_{bad} \leq r \cdot (t+1) \cdot \left(n \cdot \left(e^{-\frac{\delta_1^2 d}{2}} + t_0 e^{-\frac{\delta_2^2 \alpha(1-\delta_1)d}{2}} \right) + e^{-h\cdot(d\gamma^2 - 2)} \right). \tag{3.45}$$

Proof. We let $\mathcal{A}$ be an adversary and construct a simulator $\mathcal{S}$ similar to how the simulator is constructed in the proof of Lemma 3.6.1.

1. $\mathcal{S}$ simulates all parties $p_i \in P$ running the protocl $\Pi_{\mathsf{ERFlood}}(\rho)$ inside itself.
2. When receiving (Leak, p_i, m) from $\mathcal{F}^{\Delta'}_{\mathsf{Flood}}$ the simulator inputs (Send, m) to p_i (running inside $\mathcal{S}$).
3. When receiving $(\mathit{SetMessage}, m)$ from the adversary on the port belonging to functionality $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$, $\mathcal{S}$ forwards $(\mathit{SetMessage}, m, p_j)$ to $\mathcal{F}^{\Delta'}_{\mathsf{Flood}}$.
4. Whenever $\mathcal{A}$ corrupts some $p_i \in \mathcal{P}$, $\mathcal{S}$ corrupts p_i and sends the simulated internal state to $\mathcal{A}$. From then on, the simulated p_i follows $\mathcal{A}$'s instructions.

We note that the only time this is not a perfect simulation is when one of the properties of the ideal functionality is violated in $\Pi_{\mathsf{ERFlood}}(\rho)$. In this case, the ideal functionality will enforce the properties itself, whereas the real protocol will not, and it will be trivial for an environment to distinguish.

To bound the probability that a property is violated, we let M be the set of messages that are sent throughout the execution of the protocol (either via *Send* or via *SetMessage*). For any message $m \in M$ that was input at time τ for the first time (either by the send command or by letting it be sent from some dishonest party) to some honest party, we let $\texttt{NoDelivery}(m)$ be the event that m was not delivered at time $\tau + \Delta'$. By a union bound, we have that

$$\begin{aligned} p_{\text{bad}} &\leq \Pr\left[\bigcup_{m\in M} \texttt{NoDelivery}(m)\right] \\ &\leq \sum_{m \in M} \Pr\left[\texttt{NoDelivery}(m)\right]. \end{aligned} \tag{3.71}$$

Now let us look at any message $m \in M$ that is input at time τ for the first time (either by the send command or by letting it be sent from some dishonest party) to some honest party p_s. Now, for $\lambda := \frac{\Delta'}{\Delta}$, Lemma 3.6.4 ensures that there exists an adversary $\mathcal{A}_1$ s.t. if $G_{\mathsf{FloodToER}_1} \xleftarrow{\$} \mathsf{FloodToER}_1(\mathcal{P}, \rho, \mathcal{A}_1, p_s)$ then

$$\Pr[\texttt{NoDelivery}(m)] \leq \Pr[\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToER}_1}, \lambda)]. \tag{3.72}$$

Lemma 3.6.7 ensures that there exists an adversary $\mathcal{A}_2$ s.t. if $G_{\mathsf{FloodToER}_2} \xleftarrow{\$} \mathsf{FloodToER}_2(\mathcal{P}, \rho, \mathcal{A}_2, p_s)$, then

$$\Pr[\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToER}_1}, \lambda)] \leq \Pr[\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToER}_2}, \lambda)]. \tag{3.73}$$

Lemmas 3.6.10 and 3.6.11 ensures that there exists an adversary $\mathcal{A}_3$ s.t. if $G_{\mathsf{FloodToER}_3} \xleftarrow{\$} \mathsf{FloodToER}_3(\mathcal{P}, \rho, \mathcal{A}_3, p_s)$ then

$$\begin{aligned}\Pr[\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToER}_2}, \lambda)] &\leq \Pr[\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToER}_3}, \lambda)] \\ &\leq \Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToER}_3}, \lambda)].\end{aligned} \tag{3.74}$$

As ϕ_{Dist} is monotone, Lemma 3.6.13 ensures that there exists an adversary $\mathcal{A}_4$ s.t. if $G_{\mathsf{FloodToER}_4} \xleftarrow{\$} \mathsf{FloodToER}_4(\mathcal{P}, \rho, \mathcal{A}_4)$ then

$$\Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToER}_3}, \lambda)] \leq \Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToER}_4}, \lambda)]. \tag{3.75}$$

Lemma 3.6.14 ensures that if $G_{\mathsf{FloodToER}_5} \xleftarrow{\$} \mathsf{FloodToER}_5(\mathcal{P}, \rho, \mathcal{A}_4)$ then

$$\Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToER}_4}, \lambda)] \leq \Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToER}_5}, \lambda)] \cdot (t+1). \tag{3.76}$$

Furthermore, as ϕ_{Diam} is also preserved under renaming, Lemmas 3.6.15 and 3.6.20 ensures that if $G_s \xleftarrow{\$} \mathbb{G}(s, \rho)$ then

$$\Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToER}_5}, \lambda)] \leq \max_{s \in \{h, \ldots, n\}} \Pr[\neg\phi_{\text{Diam}}(G_s, \lambda)]. \tag{3.77}$$

Combining Equations (3.71) to (3.77) and using that $|M| = r$ we get that

$$p_{\text{bad}} \leq r \cdot (t+1) \cdot \max_{s \in \{h, \ldots, n\}} \Pr[\neg\phi_{\text{Diam}}(G_s, \lambda)]. \tag{3.78}$$

We now look at the individual instantiations to bound $\max_{s \in \{h, \ldots, n\}} \Pr[\neg\phi_{\text{Diam}}(G_s)]$. For Instantiation 1 we obtain Equation (3.43) using Lemma 3.4.3. For Instantiation 2 we obtain Equation (3.45) using Lemma 3.4.1. □

4 Practical Provably Secure Flooding for Blockchains

4.1 Introduction

4.1.1 Motivation

Since Nakamoto proposed the first decentralized permissionless blockchain protocol [Nak08], a significant line of work has been done. In such protocols, one considers a setting where different parties are weighted according to how much of a resource they own (mining power, stake, space, etc.), and security relies on the fact that a certain fraction of the total weight (typically more than the majority, or two thirds) is owned by the honest parties.

Current blockchain protocols typically are proven secure assuming the availability of a *multicast* network, which allows each party to distribute a value among the parties within some delivery time Δ (see e.g., [Abr+20; CM19; DPS19; Dav+18; Din+20; GKL15; PS17a; PS18b]). However, very little attention has been devoted to the construction of provably secure multicast networks themselves.

In practice, the multicast network is typically implemented via a message-diffusion mechanism, where in order for a party P to distribute a message, P sends the message to a subset of its neighbors, who then forward the message to their neighbors and so on. The idea is that if the graph induced by the honest parties is connected, the message will reach all the honest parties, and if the graph has a low diameter, it will reach all honest parties after only a few iterations. Indeed, there have been works that study how to randomly select the neighbors so that the induced graph remains connected with a small diameter after removing corrupted nodes (see, e.g., [KMG03; MNT22b; RT19]).

Unfortunately, to the best of our knowledge, currently analyzed diffusion mechanisms do not consider weighted parties and, therefore, can only be proven secure when a certain constant fraction of the parties are honest (in particular, it is not enough to assume a fraction of the total weight is owned by honest parties). This means that when such a message diffusion mechanism is used to build a blockchain, the overall protocol relies on *both* the constant-honest-fraction-of-weight assumption *and* the constant-honest-fraction-of-parties assumption.

Note that for a fixed weight distribution, a bound on the corrupted weight also implies a bound on the number of parties that can be corrupted, where this maximum is achieved by greedily corrupting parties with the least weight first. Hence, current multicast protocols could, in principle, also be used, assuming only a bound on the corrupted weight. However, the message complexity of such protocols is inversely proportional to the guaranteed honesty ratio. That is, to still guarantee security under more corrupted parties, the remaining parties must send to more neighbors. In particular, this means

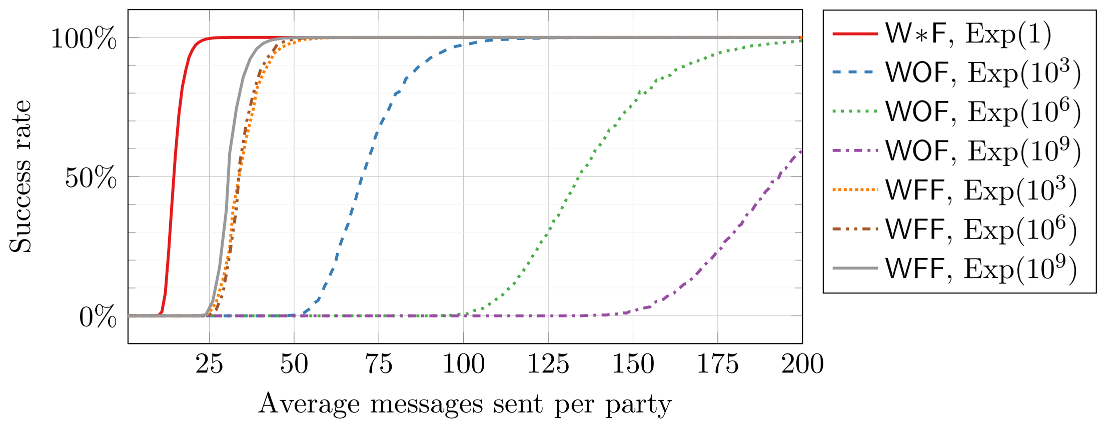


Figure 4.1: Comparison of our WFF protocol with a weight oblivious protocol WOF that chooses a fixed number of random neighbors independently of their weight. The simulations are for $n = 1024$ parties with exponential weight distributions Exp(r) where the heaviest party's weight is r times the lightest party's weight. Note that for Exp(1), WFF and WOF are identical. For each setting, we consider a 50% corruption threshold and a greedy corruption strategy where lighter parties are corrupted first. Each simulation was repeated 10 000 times, and the success rate measures how often all parties received a single sent message.

that in many of the current weight distributions where there are very few people owning a large fraction of the total weight but thousands of parties holding a tiny little fraction of the weight, the incurred concrete message complexity to achieve security significantly blows up (see an example in Figure 4.1, where even for large sizes of neighborhood sizes, the protocol fails).

The need for a practically efficient multicast network secure solely relying on the constant-honest-fraction-of-weight assumption is therefore apparent.

4.1.2 Our Contributions

In this work, we investigate provably secure protocols that implement a multicast network for the weighted setting, relying solely on the constant-honest-fraction-of-weight assumption. Additionally, we are interested in protocols that are concretely efficient. In short, we explore the following natural questions:

> *Is there a provably secure multicast protocol in the weighted setting, assuming only a constant fraction of honest weight? And if so, is there a practically efficient one?*

We answer both of these questions in the affirmative by presenting the first multicast protocol WFF (weighted fan-out flooding) that relies solely on the constant-honest-fraction-of-weight assumption and evaluate its practical efficiency by performing various simulations. More concretely, we prove the following theorem:

Theorem 4.1.1 (Informal)**.** *Let κ be a security parameter, n be the number of parties, and $\gamma \in [0, 1]$ be the fraction of the total weight that is guaranteed to belong to honest* *parties. Further, let $\delta_{Channel}$ be an upper bound on the delays of the underlying point-to-point channels. Then,* WFF *is a secure flooding protocol with maximal delay* $\Delta := \left(7 \cdot \log\left(\frac{6n}{\log(n)+\kappa}\right) + 2\right) \cdot \delta_{Channel}$ *and message complexity* $2n \cdot \frac{\log(n)+\kappa}{\gamma}$.

Note that the maximum delay and the message complexity in the theorem are independent of the weight distribution. By naturally assigning the weights to corresponding stake quantities, the achieved guarantees match those required in previous proof-of-stake blockchain protocols (see, e.g., [CM19; Dav+18; Din+20]). Therefore our protocol can be used to build a blockchain protocol from point-to-point channels without the need for any additional assumptions apart from those needed in the blockchain protocol itself.

Asymptotic optimality and practicality. Our protocol has the property that 1) parties accumulating large amounts of weight need to send to more parties, and 2) the number of parties each sends to increase logarithmically in the total number of parties. We prove that both properties are inherent for secure flooding protocols, meaning that Theorem 4.1.1 is asymptotically optimal. Concretely, for the first point, if a small set S (say, of constant size) accumulates more than a γ-fraction of the weight, then this set necessarily needs to send at least to a linear number $\Theta(n)$ of parties.

This means it is undesirable to have parties with small weights and also to have parties with massive weights. A simple way to mitigate this in practice is to exclude parties with less than W_{min} weight and cap the maximal weight to W_{max}. This means if we use the flooding for a proof-of-stake blockchain, parties with a huge amount of stake need to split their stake over several nodes such that none has more than W_{max} weight. Parties with tiny weights can still obtain data from other nodes by requesting data from them periodically. We discuss this further below.

Simulations. We use simulations to evaluate the practicality of our provably secure protocol. The simulations confirm our theoretical results and also show that our protocol is practical: Messages are diffused quickly to all parties with high success probability even when weights are unevenly distributed. On the other hand, as our simulations also show, prior protocols—oblivious of the parties' weights—fail completely for neighborhood sizes for which our provably secure protocol succeeds (see Figure 4.1). In particular, this means that our protocol achieves the necessary security guarantees using considerably fewer messages than current (weight-oblivious) protocols.

Outline of the chapter. In the remainder of this section, we will discuss our model and assumptions, provide a technical overview, and discuss related work. In Section 4.2, we introduce the model for which our results hold, as well as the notation and basic graph definitions used in the remainder of the chapter. In Section 4.3, we present our practical flooding protocol and prove it secure based on the constant-honest-fraction-of-weight assumption. In Section 4.4, we present theoretical lower bounds showing that our protocol is asymptotically optimal and discuss the practical implications of these bounds. In Section 4.5, we present two solutions for obtaining delivery to parties with zero weight. Finally, in Section 4.6, we evaluate the performance of our protocol using simulations.

4.1.3 Model and Assumptions

Network and corruption model. We assume all parties have access to an underlying network that allows them to establish point-to-point channels with other parties. We further assume each party p is publicly assigned a weight $W_p > 0$ and an adversary can corrupt parties accumulating at most a constant fraction $1 - \gamma$ of the total weight. For simplicity, we consider static corruptions in our proofs but using the techniques from [MNT22b], all our results can be extended to security against delayed adaptive adversaries (adversaries for which there is a delay from the time they decide to corrupt a party until the adversary effectively controls the party). Intuitively, if corruption takes longer than the duration from the earliest point in time an adversary can learn the neighbors of a party till the neighbors are guaranteed to have resent the message to other parties, then adaptivity does not help the adversary prevent delivery of any messages. However, significant overhead is involved in proving this because the adversary can still dynamically decide how many parties are left to guarantee delivery for (as evidenced in Chapter 3). This is why we only present proofs for a static adversary and refer to [MNT22b] for techniques for proving such a statement.

Realising public weights from resource assumptions. Proof-of-stake blockchains rely on a constant fraction of the stake being honest (typically more than $1/2$ [DPS19; Dav+18] or more than $2/3$ [CM19]). Furthermore, a blockchain itself provides a ledger accessible by all parties describing how much stake each party owns. Hence, it is immediate how to assign weights to parties by simply accessing the ledger to instantiate the weights for our protocols.

To achieve a weight distribution for blockchain protocols that rely on a constant fraction of the computational resources being honest [GKL15; PS17a; PS17b; PS18b], one can make use of the techniques for committee selection for such setting [PS17b; PS18b]. The idea behind this is that for long fragments of a chain with high chain quality, the distribution of block creators is similar to the distribution of computational resources among parties. Hence, this distribution translates directly to publicly available weight distribution to all parties. For techniques to achieve a high chain quality, see [PS17a].

Delivery to zero-weight parties. While we assume that all parties have positive weight and parties with zero weight cannot contribute to the security of the protocol, it is still desirable in practice to allow such parties to obtain the state of the system. This can be achieved, e.g., by letting such parties fetch missing data from other nodes. We discuss some options in Section 4.5.

Static versus dynamic weight. For simplicity, we consider for this work the static-weight setting, in which the weight of all parties remains fixed. This might appear unrealistic when weight is instantiated with the stake in a proof-of-stake. However, this is not a real limitation of our protocol when combined with such a blockchain. For example, in [Dav+18], to prove their protocol secure for a dynamic stake, the authors divide time into epochs where the stake used for producing blocks remains unchanged and additionally make assumptions on the speed that stake can move between epochs. In their proofs, they note that all parties agree on the stake distribution in a previous epoch. We note that our proofs only rely on static weight for the propagation of a single message, and the time it takes to propagate a message is minimal compared to such epochs.

Practicality of complete network. We note that the assumption that any two parties can establish a point-to-point connection with each other is a reasonable assumption, e.g., in the proof-of-stake setting: Parties who want to participate in the protocol first need to register their node, see, e.g., [Bad+18]. This registration process can include the node's IP address and further required information that allows other nodes to establish a connection with that node.

4.1.4 Technical Overview

Flooding protocol skeleton. Our protocol follows the basic structure of previous flooding protocols: When a party p receives a message m for the first time, p samples a set of neighbors N from the party set $\mathcal{P}$, according to some probability distribution $\mathcal{N}_p$. The party then forwards the message m to all parties in N. The crucial variable of this protocol is the distribution $\mathcal{N}_p$, i.e., how parties select their peers.

Remark 4.1.1. In most practical blockchain implementations, parties do not resample their peers for every message but keep the connections over an extended period of time [Hei+15; MHG18]. We note that our protocol can also be used in such a fashion, and all our results can be translated to such a setting. The reason for resampling peers often is that against a delayed adaptive adversary [MNT22b], security can only be guaranteed if the corruption delay is longer than the time peers keep their connections. Hence, resampling more often provides better security guarantees.

Dependency of neighborhood selection on weight distribution. It is clear that to achieve efficient results, one must use the overall weight distribution to decide whether a party p_i forwards the message to party p_j. What is perhaps less clear is what the required *amount* of dependency is. We here argue intuitively that the neighborhood selection must depend (at least) on both the weights of p_i and p_j: Consider a weight distribution where p_i's weight is overwhelming, and there are many parties with very little weight (including p_j). In this case, the adversary has a corruption budget to corrupt all parties except for p_i and p_j. Therefore, to guarantee that an honest p_j receives the message, p_i must send it to that party with probability 1. Consequently, the neighborhood selection distribution $\mathcal{N}_{p_i}$ must depend on p_i's weight. It follows via an analogous argument that p_i must send to p_j if the latter's weight is overwhelming. Hence, the probability of choosing p_j in $\mathcal{N}_p$ must also depend on p_j's weight.

A simple inefficient solution. From the above observations, we see that the neighborhood distribution must depend on both the weights W_i of p_i and W_j of p_j. A simple idea is to let each party p_i internally emulate W_{p_i} parties and then run a traditional unweighted flooding protocol among $W = \sum_p W_p$ nodes, where two nodes are connected with some probability ρ. By properties of Erdős–Rényi graphs, this leads to a secure flooding protocol [MNT22b]. Note that the probability that a node from p_i is connected to a node from p_j depends on both weights W_{p_i} and W_{p_j}, namely $1 - (1-\rho)^{W_{p_i} \cdot W_{p_j}}$.

However, the resulting protocol is highly inefficient since it has a message complexity that depends on the total sum of the weights W rather than the number of parties. Note that in current proof-of-stake systems, the total stake is in the order of billions, so any dependency on the total weight is highly undesirable.

Scaling invariance. The simple protocol from above is not only inefficient if the total weight is large, but it also has the undesirable property that the efficiency depends on the "unit" of the weight: If we multiply everybody's weight by 100, the overall number of messages increases substantially, even though this scaling does not affect the possible corruptions. We thus postulate that practical protocols should be invariant under such weight scalings.

A simple fix seems to be to normalize the weight distribution by dividing every party's weight by the weight of the lightest party. This, however, introduces two issues: First, since the number of internally emulated nodes must be an integer, this division leads to rounding issues, with implications for the security argument. Secondly, introducing an additional extremely light party now has a massive impact on efficiency, even though this additional party does not substantially change the possible corruptions.

A first theoretical protocol. Our first technical contribution is a new simple way to choose the neighbors in the flooding protocol. More precisely, we generalize the approach above and show that it is actually enough to emulate a number of nodes that is proportional to the total number of parties (rather than the total weight).

For that, we introduce the notion of an emulation-function $\mathtt{E} : \mathcal{P} \to \mathbb{N} \setminus \{0\}$. According to the emulation function, we let each party p internally emulate $\mathtt{E}(p) \geq 1$ different nodes in a graph consisting of $n_{\mathtt{E}} := \sum_p \mathtt{E}(p)$ nodes. As explained above, the basic idea is to create an Erdős–Rényi graph on the emulated graph with $n_{\mathtt{E}}$ nodes and edge-probability ρ. Then, we say that a party p_i forwards the message to p_j if *any* of the emulated nodes from p_i is connected to *any* of the emulated nodes from p_j. This means that the probability that p_i forwards the message to p_j is $1 - (1-\rho)^{\mathtt{E}(p_i) \cdot \mathtt{E}(p_j)}$.

We then consider the emulation function $\mathtt{E}(p) = \lceil \alpha_p \cdot n \rceil$, where α_p is p's fraction of the total weight. That is, we let each party emulate a number of nodes proportional to the number of parties scaled by the party's relative weight. Note that the ceiling ensures that each party emulates at least one node. We then prove that by choosing ρ appropriately such that the unweighted subgraph emulated by honest parties remains connected with low diameter, we obtain a flooding protocol with message complexity $O((\log(n) + \kappa) \cdot n \cdot \gamma^{-1})$ and time complexity $O(\log(n) \cdot \delta_{\text{Channel}})$.

A practical protocol. Although the method described above is intuitive and gives us asymptotically good complexities, it is very far from practical. In particular, the protocol requires every party to locally flip $\Omega(n)$ coins for each message. Similar to current protocols deployed in practice, we would like to have a protocol that instead chooses a *fixed* set of neighbors (possibly dependent on the weight distribution, but nothing else) and provide provable security for it.

We propose a protocol where each party p chooses to send to $K = k \cdot \mathtt{E}(p) = k \cdot \lceil \alpha_p \cdot n \rceil$ distinct parties (for a parameter k), according to a weighted sampling without replacement [BPS18]. More precisely, p chooses K parties, where the probability of choosing a certain tuple[1] of parties $(q_1, \dots, q_K)$ (among the set of parties $\mathcal{P} \setminus \{p\}$) is

$$\Pr[(q_1, \dots, q_K)] = \prod_{i=1}^{K} \frac{\mathtt{E}(q_i)}{n_{\mathtt{E}} - \mathtt{E}(p) - \mathtt{E}(q_1) - \cdots - \mathtt{E}(q_{i-1})}. \tag{4.1}$$

We show that this practical protocol has the same asymptotic guarantees as the first protocol above.

Importance of emulation function. Even though this protocol is so simple that it can be described in a few lines, it is by no means trivial. In fact, it is crucial for the correctness of the protocol that the emulation function is used to determine both the number of neighbors *and* the distribution of these neighbors.

To see that it is crucial to use the emulation function to decide how many neighbors each party should choose, consider a slight change to the protocol, namely, send to $K = k \cdot \alpha_p \cdot n$ parties (instead of $K = k \cdot \mathtt{E}(p)$). Now, consider a sender p with a small fraction of the total weight α_p, and let us estimate the parameter k to ensure that this p sends to at least one honest party. As any party potentially could be corrupt, it must be that p sends to more than just one neighbor. Hence, it must be that $k > \frac{1}{\alpha_p \cdot n}$, just to ensure this very minimal requirement. A rough bound on the message complexity of such protocol would be $\sum_{p'} k \cdot \alpha_{p'} \cdot n > \frac{1}{\alpha_p}$, which is impractical if α_p is small.

To see that it is crucial to weigh the selection of neighbors with the emulation function, we consider another small change to the protocol, namely to select parties weighted by their weight instead of the emulation function. Now, consider a weight distribution where just one party p has a tiny fraction of the total weight, and all others have roughly equal weight. Note that for any party choosing less than n neighbors, the probability that p is selected as a neighbor becomes arbitrarily small for a decreasing α_p. Hence, ensuring that p receives a message would induce a quadratic message complexity which is impractical.

Security proof. Proving the security of such a protocol in the weighted setting directly is non-trivial for two reasons: First, the choices of whether to send to a neighbor or not are not independent. Secondly, the fact that the choices are according to an arbitrary weight distribution makes the analysis considerably harder than traditional graph-theoretic results that consider the non-weighted setting. Instead of providing a direct graph-theoretic analysis, we give a security proof via a sequence of intermediate protocols, essentially relating the success probability of the first protocol above based on Erdős–Rényi graphs to the practical protocol. This leads to Theorem 4.1.1.

4.1.5 Current State of the Art and Related Work

Flooding networks in a Byzantine setting. [KMG03] was the first to relate probabilistic gossiping to the connectivity of the induced graph. They considered $(1 - \gamma) \cdot n$ out of n parties failing and showed that each party needs to forward a message with probability $\rho > \frac{\log(n) + \kappa}{\gamma \cdot n}$ to any other party to ensure that messages are delivered to all non-failing parties with a probability overwhelming in κ.

[1]The probability to choose the unordered neighborhood set $N = \{q_1, \dots, q_K\}$ is the sum over the probabilities of all permuted tuples.

[MNT22b] observed that against an adversary capable of adaptively corrupting up to t parties, any flooding network where each party sends to less than t neighbors is inherently insecure (an adversary can corrupt all neighbors of a sender). To mitigate this problem and achieve a protocol secure against a Byzantine adaptive adversary, [MNT22b] formalized the notion of a delayed adversary (informally introduced by [PS17b]) for which there is a delay from the time the adversary decides to corrupt a party until the adversary effectively controls the party. In this setting, they showed that against an adversary delayed for the time it takes to send a message plus the time it takes to resend a message, it is sufficient to, on average, send to $\Omega((\log(n) + \kappa) \cdot \gamma^{-1})$ neighbors to achieve a flooding protocol that with an overwhelming probability in κ has $O(\log(n))$ round complexity for n parties with at most $(1 - \gamma) \cdot n$ of the parties being corrupted. In this work, we match the theoretical performance of their flooding protocol with a practical protocol that only relies on a γ fraction of the weight remaining honest, which is more relevant in the blockchain setting.

Kadcast [RT19] is a recent flooding protocol designed explicitly for blockchains. Interestingly, they claim that structured networks are inherently more efficient than unstructured networks and propose a structured protocol with $O(\log n)$ neighbors and $O(\log n)$ steps to propagate a message, which is similar to what we achieve using an unstructured network. It is unclear how their protocol performs under Byzantine failures. Further, we note that structured networks are inherently vulnerable to attacks by adaptive adversaries.

A different line of work [MMR99; MPS01; MS03] considers how to propagate updates in a database using gossip where at most t of the processors may be corrupted. The setting is different from ours as they assume that at least t honest parties get the update as input initially, and only updates input to some honest processor can be accepted by the other processors.

Probabilistic communication has also been used to improve the communication complexity for both multi-party-computation (MPC) [Cha+15] and Byzantine broadcast [TLP22]. In [Cha+15], communication between honest parties are assumed to be hidden from the adversary. This is exploited by constructing a random communication network with an average polylogarithmic degree based on Erdős–Rényi graphs. They thereby achieve a MPC protocol with a low communication locality that is secure against a fully adaptive adversary. [TLP22] combines the classic broadcast protocol by Dolev and Strong [DS83] with gossiping based upon Erdős–Rényi-graphs to obtain the first broadcast algorithm with a sub-cubic communication complexity for a dishonest majority. Using similar techniques and assuming a trusted setup, they achieve an asymptotically optimal communication complexity for parallel broadcast.

A different line of work considers the problems of MPC and Agreement on incomplete communication networks [CGO10; CGO15; Dwo+88; GO08; JRV20; Kin+06; Upf94]. To circumvent complexity bounds for fully adaptive adversaries, the seminal work of [Dwo+88] introduced the problem of *almost-everywhere agreement* as a relaxation of agreement where not all nodes are required to be consistent, but a small number of nodes are allowed to be inconsistent. Since then, the relaxation has also been extended to MPC [GO08], and different solutions to this problem have been continuously improved [CGO10; CGO15; JRV20; Kin+06; Upf94]. Notably, [Kin+06] used probabilistic communication to increase the number of consistent parties, and [CGO15] used Erdős–Rényi graphs with a diameter of 2 to obtain a construction secure not only against adaptive corruptions but also an adversary allowed to remove some communication links adaptively. In our work, nodes are also of a bounded degree, but contrary to this line of work, we work in a slightly weaker adversarial model, which allows us to ensure correctness for *all* parties.

Attacks on the network layers of blockchains. Attacks on network layers of blockchains are not only a theoretical concern. In fact, several works [AZV17; Hei+15; MHG18; Tra+20] have shown that it has been possible to launch eclipsing attacks[2] against nodes in the Bitcoin network and the Ethereum network.

Bitcoin's peer-to-peer network lets each node maintain 8 outgoing connections and up to 117 incoming connections. This is clearly insecure when considering a resource-constrained adversary instead of a traditional adversary (as the probability of only connecting to adversarial nodes can be arbitrarily high). In addition to this inherent insecurity, [Hei+15] showed how to eclipse a node that is already a part of an existing honest network by exploiting a bias in how a peer selects its outgoing connections. They launched such an attack with only 4600 bots and achieved 85% success probability to eclipse a targeted node.

By default, a node in the Ethereum peer-to-peer network selects 13 outgoing connections contrary to the 8 that is the default in Bitcoin. Hence, one might be led to believe that it is more challenging to eclipse an Ethereum node than a Bitcoin node. However, in Ethereum, neighbors are selected using a distance measure based on nodes' public keys. Exploiting that in a prior version of the Ethereum client, a single computer was allowed to run several nodes, [MHG18] showed that just a single computer could be used to mount an attack by creating multiple carefully selected public keys.

[AZV17] showed that BGP-Hijacking could also be used to eclipse Bitcoin nodes. However, we note that such an attack is immediately observable as an adversary must publicly announce a false BGP prefix. In [Tra+20], it was shown that a stealthier version of such an attack could also be launched against a Bitcoin node by additionally influencing how a bitcoin node selects its outgoing connections. We note that such attacks are attacks on the internet's infrastructure and therefore fall outside the scope of our model.

We note that the attacks presented in [Hei+15; MHG18; Tra+20] all rely on exploiting the heuristics used to select outgoing connections for nodes in the peer-to-peer network. Hence, such attacks would not have been possible if a provably secure protocol (such as the one presented in this work) had been deployed instead of heuristics.

Detecting eclipse attacks. To mitigate attacks on the network layer, a line of work considers the possibility of detecting eclipse attacks [Ala+21; Xu+20; ZTA21]. [Xu+20] provide a method for using supervised learning to detect eclipsing attacks based on the metadata in packages. We note that this method is only as good as its data set for training and hence cannot be used to detect attacks in general. A different approach is to try to detect eclipse attacks based on the absence of new blocks [Ala+21; ZTA21]. However, this method has the drawback that it becomes arbitrarily slow as the fraction of resources controlled by an adversary approaches 50%, and even for small values, it

[2] An attack where an adversary tricks an honest party into talking only with adversarial parties. It is thereby possible for the adversary to manipulate the honest node in various ways.

takes upwards of 3 hours to detect. Finally, it has been considered to detect eclipse attacks using an additional overlay gossip protocol [Ala+21]. However, contrary to this work, this is not *proven* to work but instead demonstrated to work empirically.

Consequences of eclipse attacks. If a party is eclipsed, it is immediate that security proofs that rely on guaranteed message delivery no longer apply. Several works have shown that eclipse attacks not only invalidate the security proofs but actually invalidate the security of blockchain protocols [Hei+15; Nay+16; ZL19]. Eclipsing can be used to invalidate the total order that blockchain provides and thereby allow double-spend attacks [Hei+15], amplify the rewards from selfish mining [Nay+16], and dramatically speed up ”stake-bleeding”-attacks [ZL19].

The Generals' Scuttlebutt: Byzantine-Resilient Gossip Protocols [Cor+22]. Concurrent with and independent of our work, [Cor+22] considered the problem of designing a message diffusion mechanism based on the majority of honest stake assumption. The main focus of that paper is to design a network protocol specifically for the Ouroboros Praos consensus protocol [Dav+18]. To mitigate a specific denial-of-service attack possible in that protocol (and related proof-of-stake protocols), the authors propose a mechanism that relies on long-lived connections between parties to synchronize chains instead of generically diffusing messages. A consequence of these long-lived connections between parties is that an adaptive adversary can eclipse a set of honest parties. Because their ideal functionality allows such eclipsing, the functionality is different from the assumed functionality of [Dav+18] (and thereby the functionality implemented in this work), and the authors argue in [Cor+22] that security of [Dav+18] can be proven using this new functionality. In contrast, our work focuses on realizing the flooding functionality without eclipsing, which is assumed by most existing blockchain protocols. Hence, while some techniques are similar, the results of [Cor+22] are mostly orthogonal to our work.

4.2 Preliminaries and Model

4.2.1 Preliminaries

We will use κ to denote the security parameter of our protocols.

Graphs. We use standard notation for graphs and let $G = (\mathcal{V}, E)$ be a graph with nodes $\mathcal{V}$ and edges E. An edge can be either directed, in which case we will write (v, z) to denote the edge from v to z, or undirected, in which case we will write $\{v, z\}$ to denote the edge between the two nodes. We write $\mathtt{dist}(v, z)$ to denote the shortest distance between two nodes v and z. Further, we use the shorthand notation $\mathtt{MaxDist}(G, v) \triangleq \max_{z \in \mathcal{V}} \mathtt{dist}(v, z)$ for the maximum distance from v to any node in a graph $G = (\mathcal{V}, E)$, and the following notation $\mathtt{Diam}(G) \triangleq \max_{v \in \mathcal{V}} \mathtt{MaxDist}(G, v)$ for the diameter of a graph G.

We also define Erdős–Rényi graphs and digraphs.

Definition 4.2.1 (Erdős–Rényi (di)graphs)**.** An Erdős–Rényi (di)graph is a (di)graph $G = (\mathcal{V}, E)$ where all possible edges are present with an independent probability ρ. That is for any $v, z \in \mathcal{V}$, we have $\Pr[\{v, z\} \in E] = \rho$ for Erdős–Rényi graphs and

$\Pr[(v,z) \in E] = \rho$ for digraphs. To sample such a graph G with $|\mathcal{V}| = \eta$, we write $G \overset{\$}{\leftarrow} \mathcal{G}_{\mathrm{ER}}(\eta, \rho)$ and for the directed case $G \overset{\$}{\leftarrow} \mathcal{G}_{\overrightarrow{\mathrm{ER}}}(\eta, \rho)$.

Basic equalities and inequalities. For completeness, we restate some fundamental equalities and inequalities which we will use in our proofs.

Lemma 4.2.2 (Binomial formula)**.** *For $x, y \in \mathbb{R}$ and $n \in \mathbb{N}$*

$$(x+y)^n = \sum_{k=0}^{n} x^{n-k} \cdot y^k.$$

Lemma 4.2.3 (Bernoulli's inequality)**.** *For $r \in \mathbb{R} \setminus (0,1)$ and $x \geq -1$*

$$1 + r \cdot x \leq (1+x)^r .$$

Lemma 4.2.4 (Exponential inequality)**.** *For $y \geq 1$ and $|x| \leq 1$*

$$\left(1 + \frac{x}{y}\right)^y \leq e^x.$$

4.2.2 Parties, Weight, Adversary and Communication Network

We let $\mathcal{P}$ denote the static set of parties for which our protocols will work. For convenience, we let $n := |\mathcal{P}|$ and let $\mathcal{H} \subseteq \mathcal{P}$ be the set of honest parties.

We assume that a public weight is assigned to each party. 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.

We allow an adversary to corrupt any subset of the parties such that the remaining set of honest parties together constitutes more than a $\gamma \in (0,1]$ fraction of the total weight. Formally, we assume that

$$\sum_{p \in \mathcal{H}} \alpha_p \geq \gamma, \tag{4.2}$$

and that all parties have a non-zero positive weight, i.e., $\forall p \in \mathcal{P}, W_p > 0$.[3] We will refer to this assumption as the honest weight assumption. For simplicity, we consider a static adversary, although our results also hold against a so-called delayed-adaptive adversary [MNT22b], where the corruptions can be adaptively chosen but only happen after a certain amount of time.

Parties $\mathcal{P}$ have access to a complete network of point-to-point authenticated channels that guarantee delivery within a bounded delay. Concretely, we assume that all channels ensure delivery within δ_{Channel} time.

[3]For a discussion of the necessity of the zero-weight requirement see Section 4.4 and for methods to anyway achieve delivery to such zero-weight parties see Section 4.5. .

4.3 Weighted Flooding

In this section, we present a practical and provably secure flooding protocol WFF (weighted fan-out flooding) that only relies on the honest weight assumption. Before doing so, we first present our definition of a flooding protocol in Section 4.3.1. Then, in Section 4.3.2, we present a generic skeleton for flooding protocols that is parameterized by the way parties select their neighbors, instantiate this skeleton to obtain our practical protocol (WFF), and prove that it is sufficient to consider the way neighbors are selected to derive security of a protocol. We use this skeleton to define our theoretical flooding protocol that is secure based upon each party emulating a number of nodes proportional to their weight in an Erdős–Rényi graph (Section 4.3.3). Finally, in Section 4.3.4, we use two intermediary protocols to derive the security of WFF from our theoretical protocol.

4.3.1 Properties of Flooding Protocols

Below we give our property-based definition of a flooding protocol.[4]

Definition 4.3.1. 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 Δ-flooding protocol if the two properties hold with a probability overwhelming in the security parameter κ for each message m:

1) If m is input by an *honest* party for the first time at time τ, then by time $\tau + \Delta$ it is ensured that all other honest parties output m.

2) If m is output by an *honest* party at time τ, then by time $\tau + \Delta$ it is ensured that all honest parties output m.

Note that this definition subsumes the assumptions that many blockchain protocols rely on [CM19; Dav+18; Din+20; GKL15; PS17a; PS18b]. To the best of our knowledge only [Din+20] relies on both Properties 1) and 2), whereas the other works only rely solely on Property 1). However, as Property 2) essentially comes for free for the type of protocols we consider (each party will forward everything they receive and thereby act as if they themselves send the message), we have chosen to include it in our definition. Furthermore, because of this structure of our protocols, it is sufficient to bound the probability of Property 1) to show that our protocols are, in fact, flooding protocols according to the definition. For our proofs and lemma statements, it is, therefore, useful to define notation for the predicate that a message input to an honest party for the first time is delivered respecting the delivery bound for a flooding protocol, which is what we encapsulate in the predicate below.

Definition 4.3.2 (Timely delivery)**.** For a message m that is input for the first time at an honest party at time τ, we say that m is Δ-*timely-delivered* if all honest parties have output m no later than time $\tau + \Delta$. We let $\texttt{Timely}_m(\Delta)$ denote the induced predicate.

[4]Note that for protocols with no secrecy (each event is leaked to the adversary) and for functionalities that give the adversary complete control while respecting these properties, a simulation-based security notion is directly implied by the property-based definition. For flooding networks, this technique is used in the proofs in [MNT22b, proof of Lemma 3.6.1].

Similarly, for a message m that is input for the first time at an honest party, we define the *message complexity* as the number of messages sent by honest parties until all honest parties output m. Looking ahead, since our protocols only consist of forwarding the initial message m, the total message complexity is $|m|$ times the message complexity.

Mitigating denial-of-service attacks. It is immediate that any protocol that lives up to the definition of a flooding protocol, as given above, is open to denial-of-service attacks. An adversary can simply flood arbitrary messages until the bandwidth is exceeded. This is possible because the definition requires *all* messages to be forwarded. It is natural to consider a notion of validity to prevent such attacks and only require the delivery guarantees to apply for "valid" messages. Concretely, one could let each party $p \in \mathcal{P}$ have an updatable local predicate $\texttt{Valid}_p$ and only require that messages that are considered valid by all parties for Δ after being input/output for the first time should be propagated.

We have left this out of our definition and protocols for clarity of presentation. However, we note that it is easy to accommodate our protocols to such a notion by letting each party check if a message is valid before propagating it. We note that with such modification, all our proofs and lemmas still hold for messages considered valid by all parties for at least Δ after input/output.

4.3.2 A Skeleton For Flooding Protocols

We now present a skeleton for our flooding algorithm. The structure of the protocol is very similar to the protocols proposed in [MNT22b, Section 3.6.2], but contrary to their protocols, our protocol takes an additional parameter $\mathcal{N}$, which is an algorithm that allows each party to sample a set of neighbors. We refer to this parameter as the *neighborhood selection algorithm.*

The protocol accepts two commands: One for sending and one for checking which messages have been received. Once a send command is issued to a party, the party will forward the message to a set of neighbors determined using the neighborhood selection algorithm. Furthermore, once a message is received on a point-to-point channel, the receiver checks if it has already been relayed, and if not, it forwards the message to a set of neighbors that is again selected using the neighborhood selection algorithm.

Protocol $\Pi_{\text{Flood}}(\mathcal{N})$

We use $\mathcal{N}_p$ to denote the neighborhood distribution of party p. 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 $\texttt{Relayed}_i := \varnothing$.

Send: When p_i receives (Send, m), they sample a set of neighbors $N \overset{\$}{\leftarrow} \mathcal{N}_p$ and forwards the message to all parties in N. 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 $m \notin \texttt{Relayed}_i$, p_i continues as if they had received $(\mathcal{Send}, m)$. Otherwise, m is ignored.

Looking ahead and as an example of a neighborhood selection algorithm, we present our practical and provably secure algorithm next.

4.3.2.1 A Practical Neighborhood Selection Algorithm

Our algorithm $\mathsf{WFS}(\mathtt{E}, k)$ (abbreviation for "Weighted Fan-out Selection") takes two parameters: a function $\mathtt{E} : \mathcal{P} \to \mathbb{N}$ that allows taking stake into account when deciding how many neighbors each party should select and a parameter k that scales this number.

The idea of the algorithm is that each party p chooses $K := k \cdot \mathtt{E}(p)$ number of neighbors (excluding themselves). The neighbors are chosen according to weighted sampling without replacement [BPS18] where each party is again weighted with $\mathtt{E}$. More precisely, party p chooses K neighbors from $\mathcal{P} \setminus \{p\}$, and the probability to choose the tuple of neighbors $(q_1, \dots, q_K)$ is defined as:

$$\Pr[(q_1, \dots, q_K)] = \prod_{i=1}^{K} \frac{\mathtt{E}(q_i)}{\sum_{q \in \mathcal{P} \setminus \{p, q_1, \dots, q_{i-1}\}} \mathtt{E}(q)}. \tag{4.3}$$

The probability of choosing a particular neighborhood set $\{q_1, \dots, q_K\}$ is the sum of the probabilities over all the permuted tuples. We denote by $\mathcal{W}(K, \mathtt{E}, p)$ the resulting distribution.

Algorithm $\mathsf{WFS}_p(\mathtt{E}, k)$

```
1: Let $N := \varnothing$.
2: Set $K := k \cdot \mathtt{E}(p)$.
3: Sample $N \overset{\$}{\leftarrow} \mathcal{W}(K, \mathtt{E}, p)$.
4: return $N$.
```

Our final protocol is the protocol obtained by instantiating the flooding skeleton Π_{Flood} with the neighborhood selection algorithm WFS that again is to be instantiated with the function $\mathtt{E}(p) := \lceil \alpha_p \cdot n \rceil$. We name this protocol the *weighted fan-out flooding* protocol and use the abbreviation $\mathsf{WFF}(k) := \Pi_{\mathrm{Flood}}(\mathsf{WFS}(\mathtt{E}, k))$ for $\mathtt{E}(p) := \lceil \alpha_p \cdot n \rceil$. In Sections 4.3.3 and 4.3.4, it will become apparent why this exact choice of function is advantageous and ensures a secure protocol, but for now, we only state our final theorem, which states that WFF is, in fact, a flooding protocol with a logarithmic round complexity and a low message complexity.

Theorem 4.3.3. *Let* $\Delta := \left(7 \cdot \log\left(\frac{6n}{\log(n)+\kappa}\right) + 2\right) \cdot \delta_{Channel}$. *Then* $\mathsf{WFF}\left(\frac{\log(n)+\kappa}{\gamma}\right)$ *is a* Δ*-flooding protocol with message complexity less than* $2n \cdot \frac{\log(n)+\kappa}{\gamma}$.

4.3.2.2 The Honest Sending Process

To prove the security of WFF, we will relate the security of WFF to a series of other protocols, which will all take the structure of Π_{Flood} but use different neighborhood selection algorithms. Hence, we would like to be able to relate the security of the overall flooding protocol to just the neighborhood selection algorithm used. To do so, we first define a random process for creating a graph where each honest party is a node, given a family of neighborhood selection algorithms $\mathcal{N}$, a starting party p, and a distance λ. The intuition is that this process mimics the worst-case behavior of the adversary during a sending process starting from party p. However, separating this into a process without adversarial influence allows us to relate probabilistic experiments without considering the choices of an adversary, which could have a strategy that depends on parts of the outcome of the experiments.

Definition 4.3.4. Let $\mathcal{N}$ be a family of neighborhood selection algorithms, let $p \in \mathcal{H}$, and let $\lambda \in \mathbb{N}$ be a distance. We let the *honest sending process*, $\mathsf{HSP}(p, \mathcal{N}, \lambda)$, be a random process that returns a directed graph $G = (\mathcal{V}, E)$ defined by the following random procedure:

1. Initially, $E \coloneqq \varnothing$. Furthermore, we keep track of set $\texttt{Flipped} \coloneqq \varnothing$ that consists of nodes that have already had their outgoing edges decided, and a first-in-first-out queue $\texttt{ToBeFlipped} \coloneqq \{(p, 0)\}$ of nodes and their distance from p that are to have their edges decided.
2. The process proceeds with the following until $\texttt{ToBeFlipped} == \varnothing$.
   a) Take out the first element of $\texttt{ToBeFlipped}$ and let it be denoted by (p', i).
   b) Let $N \overset{\$}{\leftarrow} \mathcal{N}_{p'}$ and set $N \coloneqq N \cap \mathcal{H}$.
   c) Update the set of edges $E \coloneqq E \cup \{(p', p'') \mid p'' \in N\}$ and let $\texttt{Flipped} \coloneqq \texttt{Flipped} \cup \{p'\}$.
   d) If $i + 1 < \lambda$, for all $p'' \in N \setminus \texttt{Flipped}$ add $(p'', i + 1)$ to $\texttt{ToBeFlipped}$.
3. Finally, return $G = (\mathcal{H}, E)$.

Next, we are interested in bounding the probability that a message is delivered within the time guaranteed by the flooding algorithm in terms of the probability that there is a low distance for all parties from the sender. We show that the probability that Π_{Flood} ensures timely delivery for a message is lower-bounded by the probability that the honest sending process results in a graph where the sender can reach all other honest nodes within a certain number of steps. The basic idea of the proof is to construct a random graph by letting each party be a node and include a directed edge from one party to another if a message is sent and delivered before time $\lambda \cdot \delta_{\text{Channel}}$. We then show how the randomness from this experiment can be used to define HSP and relate the two graphs.

Lemma 4.3.5. *Let $\mathcal{N}$ be a family of neighborhood selection algorithms, let $p \in \mathcal{H}$, and let $\lambda \in \mathbb{N}$ be a distance. Further, let m be a message input to p for the first time during the execution of $\Pi_{Flood}(\mathcal{N})$ and let $G \overset{\$}{\leftarrow} \mathsf{HSP}(p, \mathcal{N}, \lambda)$. Then,*

$$\Pr[\texttt{MaxDist}(G, p) \leq \lambda] \leq \Pr[\texttt{Timely}_m(\lambda \cdot \delta_{Channel})]. \tag{4.4}$$

Proof. To see this we consider a another graph $G' = (\mathcal{V}', E')$ where each honest party is a node (i.e., $\mathcal{V}' = \mathcal{H}$) and there is an edge $(p_i, p_j) \in E'$ from party p_i to p_j if and only if p_j received the message m from p_i before time $\tau + \lambda \cdot \delta_{\text{Channel}}$. In this graph, it is clear that the delivery guarantee is satisfied for any node with an incoming edge. In particular if $\texttt{MaxDist}(G', p) \leq \lambda$ then $\texttt{Timely}_m(\lambda \cdot \delta_{\text{Channel}})$. It is hence sufficient to argue that $\texttt{MaxDist}(G', p) \leq \lambda$ stochastically dominates $\texttt{MaxDist}(G, p) \leq \lambda$. We show this via a direct coupling between the two graphs via a graph $\widetilde{G} = (\mathcal{H}, \widetilde{E})$, namely by first constructing G' and then constructing $\widetilde{G}$ by duplicating the edges from E' for all parties within distance $\lambda - 1$ of p to also appear in $\widetilde{E}$. It is clear that the $\widetilde{E} \subseteq E'$ and hence that $\texttt{MaxDist}(\widetilde{G}, p) \leq \lambda \implies \texttt{MaxDist}(G', p) \leq \lambda$. Therefore, what is left is only to argue that $G \sim \widetilde{G}$. Observe that any node that receives the message at the latest at $\tau + (\lambda - 1) \cdot \delta_{\text{Channel}}$ is ensured to have edges added corresponding to $\mathcal{N}$ in G' and hence also to $\widetilde{G}$. Furthermore, for any $d \in \mathbb{N}$ at time $\tau + d \cdot \delta_{\text{Channel}}$, any node at a distance d from the sender will get the message delivered and thereby select their neighbors. Therefore all nodes at a distance $\lambda - 1$ will have all their edges added to the graph $\widetilde{G}$ and hence $G \sim \widetilde{G}$. □

Lemma 4.3.5 ensures that it is sufficient to consider neighborhood selection algorithms and prove that graphs constructed via the honest sending process have a low distance from the sender to all other parties.

4.3.3 A Theoretical Protocol: Emulating Nodes in Erdős–Rényi Graphs

Our central idea for achieving a flooding network that relies on the honest weight assumption is to let each party emulate a number of nodes proportional to their weight in a hypothetical Erdős–Rényi graph. We will refer to this hypothetical graph as the *emulated graph*. Now, our idea is that if there is an edge between an emulated node v and another emulated node z corresponds to that the party emulating node v should forward the message to the party emulating z. Our goal is now to ensure each honest party emulates at least one node and that the emulated graph has a low diameter, as this will result in all parties receiving the message quickly.

Concretely, we introduce a function $\texttt{E} : \mathcal{P} \to \mathbb{N} \setminus \{0\}$, which for each party determines how many nodes this party should act as in the emulated graph. We refer to this function as the *emulation function*.[5] For such emulation function, we define notation for the number of emulated nodes $n_{\texttt{E}}$ and the number of honest nodes that are emulated $h_{\texttt{E}}$:

$$n_{\texttt{E}} \triangleq \sum_{p \in \mathcal{P}} \texttt{E}(p) \quad \text{and} \quad h_{\texttt{E}} \triangleq \sum_{p \in \mathcal{H}} \texttt{E}(p).$$

Before looking at how to choose an emulation function, let us present how the idea leads to a straightforward algorithm for selecting neighbors by letting the emulated graph take the form of an Erdős–Rényi graph. We let ρ denote the probability that there should be an edge between any two nodes in the emulated graph. The probability

[5]For a function to be an emulation function, we require that all parties should emulate at least 1 node, which is why the codomain of the function is defined to be $\mathbb{N} \setminus \{0\}$.

that party p_i should forward a message to party p_j is:

$$\begin{aligned}
&\Pr[p_i \text{ should forward a message to } p_j] \\
&:= \Pr[\text{exists edge from any of } p_i\text{'s emulated nodes to any of } p_j\text{'s}] \\
&= 1 - \Pr[\text{there are no edges between any of } p_i \text{ and } p_j\text{'s emulated nodes}] \\
&= 1 - (1 - \Pr[\text{there is an edge between any two emulated nodes})^{\mathtt{E}(p_i)\cdot\mathtt{E}(p_j)} \\
&= 1 - (1-\rho)^{\mathtt{E}(p_i)\cdot\mathtt{E}(p_j)}.
\end{aligned} \tag{4.5}$$

This gives rise to the following family of neighbor selection algorithms indexed by a party $p \in \mathcal{P}$ and parameterized by an emulation function $\mathtt{E}$ and an edge probability ρ.

Algorithm $\textsf{ER-Emulation}_p(\mathtt{E}, \rho)$

```
1: Let $N := \varnothing$.
2: Let $P := \mathcal{P} \setminus p$.
3: while $P \neq \varnothing$ do
4:     Pick $r \in P$.
5:     Sample $c \xleftarrow{\$} \mathcal{U}([0,1])$.
6:     if $c \leq 1 - (1-\rho)^{\mathtt{E}(p)\cdot\mathtt{E}(r)}$ then
7:         Update $N := N \cup \{r\}$.
8:     Update $P := P \setminus \{r\}$.
9: return $N$.
```

Relating Erdős–Rényi graphs and the honest sending process. We now formalize the intuition that given that an emulated graph is "well connected", then the graph from the honest sending process is also "well connected". In particular, we relate the probability that the distance in a directed Erdős–Rényi graph is large to the probability that the distance from the sender is large in the honest sending process. The basic idea of the proof is to use Equation (4.5) and a mapping between the nodes of Erdős–Rényi graph and the honest parties to define both graph distributions in terms of the same random experiment. We then observe that the edges relevant for the distance in Erdős–Rényi graphs are also included in the graph arising from the honest sending process.

Lemma 4.3.6. *Let $\rho \in [0,1]$, let $\lambda \in \mathbb{N}$, let $p \in \mathcal{H}$, and let $\mathtt{E} : \mathcal{P} \to \mathbb{N} \setminus \{0\}$ be an emulation function. Further, let $G_1 \xleftarrow{\$} \textsf{HSP}(p, \textsf{ER-Emulation}(\mathtt{E}, \rho), \lambda)$ and let $G_2 \xleftarrow{\$} \mathcal{G}_{\overrightarrow{\mathrm{ER}}}(h_{\mathtt{E}}, \rho)$. Then for any node $v \in \mathcal{V}$ we have,*

$$\Pr[\mathtt{MaxDist}(G_2, v) \leq \lambda] \leq \Pr[\mathtt{MaxDist}(G_1, p) \leq \lambda]. \tag{4.6}$$

Proof. Let $v \in \mathcal{V}$. We define a simple coupling between the two graphs, G_1 and $G_2 = (\mathcal{V}, E_2)$, via a new graph $\widetilde{G_1} = (\mathcal{H}, \widetilde{E_1})$.

Before defining the coupling itself, we define some additional notation. We define a mapping $\mathtt{m}\colon \mathcal{H} \to 2^{\mathcal{V}}$ that maps each honest party to a set of nodes via the emulation function. That is, for each party $p' \in \mathcal{H}$, we define $\mathtt{m}(p')$ to be a set of parties $\left\{z_{p'_1}, \ldots, z_{p'_{\mathtt{E}(p')}}\right\}$ such that for any two parties $p_i, p_j \in \mathcal{H}$ we have that their sets of emulated nodes are non-overlapping, i.e., $\mathtt{m}(p_i) \cap \mathtt{m}(p_j) = \varnothing$. Further, we define it such that $v \in \mathtt{m}(p)$. Note that $\bigcup_{p' \in \mathcal{H}} \mathtt{m}(p') = \mathcal{V}$. Similarly, we will use $\mathtt{m}^{-1} \colon \mathcal{V} \to \mathcal{H}$ to find the party that "emulates" a particular node. We now define our coupling via the following random process that closely mimics the honest sending process.

1. Initially, sample $G_2 = (\mathcal{V}, E_2) \xleftarrow{\$} \mathcal{G}_{\overrightarrow{\mathrm{ER}}}(h_{\mathtt{E}}, \rho)$ and let $\widetilde{E_1} := \varnothing$. Furthermore, we keep track of set $\mathtt{Flipped} := \varnothing$ that consists of nodes that have already had their outgoing edges decided, and a first-in-first-out queue $\mathtt{ToBeFlipped} := \{(p, 0)\}$ of nodes and their distance from p that are to have their edges decided.
2. The process proceeds with the following until $\mathtt{ToBeFlipped} == \varnothing$.
   a) Take out the first element of $\mathtt{ToBeFlipped}$ and let it be denoted by (p', i).
   b) We now update the neighborhood of p' by looking at the edges from the emulated nodes of party p', i.e., we assign a fresh variable N to be $N := \{(p', \mathtt{m}^{-1}(z)) \mid w \in \mathtt{m}(p') \wedge (w, z) \in E_2\} \setminus \{(p', p')\}$.
   c) Now, update the set of edges $E := E \cup \{(p', p'') \mid p'' \in N\}$ and let $\mathtt{Flipped} := \mathtt{Flipped} \cup \{p'\}$.
   d) Furthermore, if $i + 1 < \lambda$ then for all $p'' \in N \setminus \mathtt{Flipped}$ add $(p'', i + 1)$ to $\mathtt{ToBeFlipped}$.
3. Finally, return $\widetilde{G_1} = (\mathcal{H}, \widetilde{E_1})$.

That $G_1 \sim \widetilde{G_1}$ follows from Equation (4.5). It is thus left to show that $\mathtt{MaxDist}(G_2, v) \leq \lambda \implies \mathtt{MaxDist}(\widetilde{G_1}, p) \leq \lambda$. We prove this by proving the contrapositive statement, $\mathtt{MaxDist}(\widetilde{G_1}, p) > \lambda \implies \mathtt{MaxDist}(G_2, v) > \lambda$. Assuming the LHS of the implication, we get that some party $p' \in \mathcal{H}$ is not within distance λ of p. Now let $z \in \mathtt{m}(p')$ be a node that is emulated by p' (we know that such exists by the definition of the emulation function), and let us show that z is not within distance of v. For the sake of contradiction, assume that z is within distance λ of v. However, any path in G_2 of length at most λ from v induces a path in the $\widetilde{G_1}$ by applying $\mathtt{m}^{-1}$ to the path. When pruned for duplicate nodes, this path in $\widetilde{G_1}$ is at most as long as the path in G_2, and hence we have a contradiction. □

Next, we show that the probability that a particular node can reach all other nodes within a certain distance in a directed Erdős–Rényi graph is lower-bounded by the probability that an undirected Erdős–Rényi graph has a high diameter. The idea of the proof is to define a coupling between the two graphs such that the edges that are relevant for the distance from the particular node in the directed are ensured to have undirected counterparts included in the undirected graph. It follows that any path starting from this node in the directed graph translates to a similar path in the undirected graph.

Lemma 4.3.7. *Let $\rho \in [0, 1]$, let $\lambda \in \mathbb{N}$ and let $\eta \in \mathbb{N}$. Further, let $G_1 = (\mathcal{V}_1, E_1) \xleftarrow{\$} \mathcal{G}_{\overrightarrow{\mathrm{ER}}}(\eta, \rho)$ and let $G_2 = (\mathcal{V}_2, E_2) \xleftarrow{\$} \mathcal{G}_{\mathrm{ER}}(\eta, \rho)$. Then for any node $v \in \mathcal{V}_1$ we have,*

$$\Pr[\mathtt{Diam}(G_2) \leq \lambda] \leq \Pr[\mathtt{MaxDist}(G_1, v) \leq \lambda]. \tag{4.7}$$

Proof. We observe that $\mathcal{V}_1 = \mathcal{V}_2$ and will from now on just use $\mathcal{V}$. Let $v \in \mathcal{V}$. It is now sufficient to show that $\texttt{MaxDist}(G_1, v) \leq \lambda$ stochastically dominates $\texttt{Diam}(G_2) \leq \lambda$. We show this by defining a coupling of the two graphs $\widetilde{G_1} = (\mathcal{V}, \widetilde{E_1})$ and $\widetilde{G_2} = (\mathcal{V}, \widetilde{E_2})$ and show that if $\texttt{Diam}(\widetilde{G_2}) \leq \lambda$ holds then also $\texttt{MaxDist}(\widetilde{G_1}, v) \leq \lambda$ holds. The idea behind the coupling is to start an edge-selection process by selecting the edges of v and now select nodes from the next node to pick edges by taking the one that is "closest" to v. In more detail, we define the coupling via the following random process.

1. Initially, $\widetilde{E_1} := \widetilde{E_2} := \varnothing$. Furthermore, we keep track of set $\texttt{Flipped} := \varnothing$ that consists of nodes that have already had their outgoing edges decided, and a first-in-first-out queue $\texttt{ToBeFlipped} := \{v\}$ of nodes that are to have their edges decided.

2. The process proceeds with the following loop until $\texttt{Flipped} == \mathcal{V}$.

   a) Take out the first of $\texttt{ToBeFlipped}$ and call it z. If no such node exists, let z be an arbitrary element in $\mathcal{V} \setminus \texttt{Flipped}$.

   b) Now for all nodes $w \in \mathcal{V} \setminus \{z\}$ flip a coin that comes out head with probability ρ, and if heads let $\widetilde{E_1} := \widetilde{E_1} \cup \{(z, w)\}$. Further, if the coin comes out heads and $w \notin \texttt{Flipped}$ then let $\widetilde{E_2} := \widetilde{E_2} \cup \{\{z, w\}\}$.

   c) Finally, let $\texttt{Flipped} := \texttt{Flipped} \cup \{z\}$.

The above process ensures that $E_1 \sim \widetilde{E_1}$ as for any potential edge (z, w) an independent coin is flipped for whether or not to add an edge exactly once. Similarly, it also holds that $E_2 \sim \widetilde{E_2}$ as there is exactly one independent coin flip for each potential edge $\{z, w\}$.

What is left is thus to show that $\texttt{Diam}(\widetilde{G_2}) \leq \lambda \implies \texttt{MaxDist}(\widetilde{G_1}, v) \leq \lambda$. To see this let $z \in \mathcal{V}$, and let $\texttt{dist}_{\widetilde{G_1}}$ and $\texttt{dist}_{\widetilde{G_2}}$ denote the distance functions for the respective graphs. We now assume the LHS of the implication and prove the RHS. The LHS ensures that

$$\texttt{dist}_{\widetilde{G_2}}(v, z) \leq \lambda. \tag{4.8}$$

This implies that there is a path of nodes with edges between them $w_1, \ldots, w_{\lambda'-1}, w_{\lambda'}$ for some $\lambda' \leq \lambda$ that connects v to z in $\widetilde{G_2}$ and where $z = w_{\lambda'}$. We now observe that any node in this path w_i is also at distance i from v in graph $\widetilde{G_2}$. To see this, observe that edges are added to $\widetilde{E_1}$ in an ordered fashion such that they do not change the distance to any nodes that already had their edges selected. This implies that any such node in this path, w_i, selected its edges before node w_{i+1}. Now, as edges being added to $\widetilde{E_2}$ are only added to nodes that have not yet had their edges selected, this implies that this path also exists in $\widetilde{G_1}$ which concludes the proof. □

Choosing a good emulation function. Let us now consider how to select a good emulation function. Before considering a concrete function, let us consider what properties constitute a good emulation function. The only property of the emulation function we have used so far is that all parties should emulate at least 1 node.[6] However, there are additional things that we want from a useful emulation function:

1. It should ensure a low distance from any sender in the graph resulting from the honest sending process.

[6]This property was used in the proof of Lemma 4.3.6.

2. The message complexity of the protocol should be as small as possible.

Lemmas 4.3.6 and 4.3.7 bounds the probability that the honest sending process results in a graph with some nodes not reachable within the sender in terms of the probability that an Erdős–Rényi graph (of size identical to the number of honest emulated nodes) has a large diameter. Furthermore, looking ahead, we will instantiate $\rho \approx \frac{\log(h_{\mathtt{E}})+\kappa}{h_{\mathtt{E}}}$ to obtain an Erdős–Rényi graph that has a diameter logarithmic in $h_{\mathtt{E}}$ unless with a probability that is negligible in κ. Unfortunately, $h_{\mathtt{E}}$ will not be known at the time of instantiation, so we will have to instantiate ρ with a lower bound on $h_{\mathtt{E}}$ in the denominator and, similarly, an upper bound in the denominator. For this discussion, let us use $n_{\mathtt{E}}$ as an upper bound.

The expected number of neighbors for a party is linear in ρ. To see this let $N \overset{\$}{\leftarrow} \mathsf{ER\text{-}Emulation}_p(\mathtt{E}, \rho)$ and let us estimate the expected size of N using Bernoulli's inequality (Lemma 4.2.3):

$$\begin{aligned}\mathbb{E}[|N|] &= \sum_{r \in \mathcal{P}\setminus\{p\}} 1-(1-\rho)^{\mathtt{E}(p)\cdot\mathtt{E}(r)} \\ &\leq \sum_{r \in \mathcal{P}\setminus\{p\}} \rho \cdot \mathtt{E}(p)\cdot\mathtt{E}(r) \\ &\leq \rho\cdot\mathtt{E}(p)\cdot n_{\mathtt{E}}.\end{aligned} \tag{4.9}$$

Hence, for ρ chosen according to the above, a bound on the expected message complexity will be

$$O\left((\log(n_{\mathtt{E}})+\kappa)\cdot\frac{n_{\mathtt{E}}^2}{h_{\mathtt{E}}}\right). \tag{4.10}$$

Our approach for finding a good emulation function has thus been to search for an emulation function that makes this value as small as possible. As a result of this approach, we choose the emulation function to be

$$\mathtt{E}(p) := \lceil \alpha_p \cdot n \rceil. \tag{4.11}$$

For this emulation function above we can derive the following bounds using only the honest weight assumption:

$$h_{\mathtt{E}} = \sum_{p\in\mathcal{H}} \mathtt{E}(p) \geq \sum_{p\in\mathcal{H}} \alpha_p \cdot n \geq \gamma\cdot n, \tag{4.12}$$

and

$$n_{\mathtt{E}} = \sum_{p\in\mathcal{P}} \mathtt{E}(p) \leq \sum_{p\in\mathcal{P}} (\alpha_p\cdot n+1) = 2\cdot n. \tag{4.13}$$

By plugging the bounds from Equations (4.12) and (4.13) into Equation (4.10), we acquire an expected message complexity that is upper bounded by $O\left((\log(n)+\kappa)\cdot\frac{n}{\gamma}\right)$ when parameters are instantiated to obtain a logarithmic diameter of the graph. If we instead of assuming a constant fraction of honest weight, assumed a constant fraction of honest parties, we could let $\mathtt{E}(p) := 1$, which would result in $n_{\mathtt{E}} := 1$ and thereby a protocol identical to the one proposed in [MNT22b, Section 3.6.2]. By using the same analysis as above, we would then be able to bound the expected message complexity by $O\Big((\log(n) + \kappa) \cdot \frac{n}{\gamma}\Big)$. Interestingly, the bound on the message complexity for the weighted section would only be a factor of ≈ 4 larger than the corresponding bound for the non-weighted setting.

Proving security of our theoretical flooding protocol. Before stating and proving that the probability that $\Pi_{\mathrm{Flood}}(\mathsf{ER\text{-}Emulation}(\mathtt{E}, \rho))$ protocol does not ensure timely delivery is negligible for certain choices of $\mathtt{E}$ and ρ, we specialize a bound on the diameter of undirected Erdős–Rényi graphs from [MNT22b, Section 3.4.1].

Corollary 4.3.8. *Let $\eta \in \mathbb{N}$, $d \in [7, \infty]$, let $\rho := \frac{d}{\eta}$, and let $G \xleftarrow{\$} \mathcal{G}_{\mathrm{ER}}(\eta, \rho)$. Then*

$$\begin{aligned}&\Pr\left[\mathtt{Diam}(G) > 7 \cdot \log\left(\frac{\eta}{2 \cdot d}\right) + 2\right] \\ &\leq \eta \cdot \left(e^{-\frac{d}{18}} + \left(6 \cdot \log\left(\frac{\eta}{2 \cdot d}\right) + 1\right) \cdot e^{-\frac{7 \cdot d}{108}}\right) + e^{-\eta \cdot \left(\frac{d}{9} - 2\right)}.\end{aligned} \tag{4.14}$$

Proof. We set

$$\delta_1 := \delta_2 := \mu := \frac{1}{3} \quad \text{and} \quad \nu := \frac{7}{4}.$$

The bound now follows from Lemma 3.4.1 (Chapter 3) as the precondition of this lemma is fulfilled. □

We now bound the probability that the distance of the honest sending process using the neighborhood selection algorithm $\mathsf{ER\text{-}Emulation}(\mathtt{E}, \rho)$ has a large distance from the sender. The idea of the proof is to use Equations (4.12) and (4.13) to bound the size of the emulated graph in the honest sending process and apply Lemmas 4.3.6 and 4.3.7 to reduce the probability to the probability that an Erdős–Rényi graph has a low diameter. The bound then follows from Corollary 4.3.8.

Lemma 4.3.9. *Let $\mathtt{E}(p) := \lceil \alpha_p \cdot n \rceil$, let $d \in [7, \infty]$, and let $\rho := \frac{d}{\gamma \cdot n}$. Further, let $p \in \mathcal{H}$ and $G \xleftarrow{\$} \mathsf{HSP}(p, \mathsf{ER\text{-}Emulation}(\mathtt{E}, \rho), ((7 \cdot \log\left(\frac{n}{d}\right) + 2))$. Then*

$$\begin{aligned}&\Pr\left[\mathtt{MaxDist}(G, p) \leq \left(7 \cdot \log\left(\frac{n}{d}\right) + 2\right)\right] \\ &\geq 1 - \left(2 \cdot n \cdot \left(e^{-\frac{d}{18}} + \left(6 \cdot \log\left(\frac{n}{d}\right) + 1\right) \cdot e^{-\frac{7 \cdot d}{108}}\right) + e^{-\gamma \cdot n \cdot \left(\frac{d}{9} - 2\right)}\right).\end{aligned} \tag{4.15}$$

Proof. As $h_{\mathtt{E}} \leq n_{\mathtt{E}}$, Equations (4.12) and (4.13) ensures that $h_{\mathtt{E}} \in [\gamma \cdot n, 2 \cdot n]$. Corollary 4.3.8 ensures that for $G' \xleftarrow{\$} \mathcal{G}_{\mathrm{ER}}\left(h_{\mathtt{E}}, \frac{d}{h_{\mathtt{E}}}\right)$ we have

$$\begin{aligned}&\Pr\left[\mathtt{Diam}(G') \leq 7 \cdot \log\left(\frac{h_{\mathtt{E}}}{2 \cdot d}\right) + 2\right] \\ &\geq 1 - \left(h_{\mathtt{E}} \cdot \left(e^{-\frac{d}{18}} + \left(6 \cdot \log\left(\frac{h_{\mathtt{E}}}{2 \cdot d}\right) + 1\right) \cdot e^{-\frac{7 \cdot d}{108}}\right) + e^{-h_{\mathtt{E}} \cdot \left(\frac{d}{9} - 2\right)}\right).\end{aligned} \tag{4.16}$$

The probability that $\mathtt{Diam}(G') \leq 7 \cdot \log\left(\frac{h_{\mathtt{E}}}{2 \cdot d}\right) + 2$ holds monotonously increases when the edge probability increases and hence this probability also holds for $G' \xleftarrow{\$} \mathcal{G}_{\mathrm{ER}}\left(h_{\mathtt{E}}, \frac{d}{\gamma \cdot n}\right)$.

Further for any graph G and natural numbers $a, b \in \mathbb{N}$ such that $a \leq b$ we have that $\texttt{Diam}(G') \leq a \Rightarrow \texttt{Diam}(G') \leq b$. Hence, we have that $G' \xleftarrow{\$} \mathcal{G}_{\text{ER}}\left(h_{\texttt{E}}, \frac{d}{\gamma \cdot n}\right)$ satisfies

$$\begin{aligned} &\Pr\left[\max_{h \in [\gamma \cdot n, 2 \cdot n]} \texttt{Diam}(G) \leq 7 \cdot \log\left(\frac{h}{2 \cdot d}\right) + 2\right] \\ &\geq 1 - \max_{h \in [\gamma \cdot n, 2 \cdot n]} \left(h \cdot \left(e^{-\frac{d}{18}} + \left(6 \cdot \log\left(\frac{h}{2 \cdot d}\right) + 1\right) \cdot e^{-\frac{7 \cdot d}{108}}\right) + e^{-h \cdot \left(\frac{d}{9} - 2\right)}\right). \end{aligned} \tag{4.17}$$

Now, applying Lemmas 4.3.6 and 4.3.7 as well as inserting bounds for the maximum value for the expressions we obtain Equation (4.15). □

A direct corollary of Lemmas 4.3.5 and 4.3.9 is that the probability that the protocol $\Pi_{\text{Flood}}(\mathsf{ER\text{-}Emulation}(\texttt{E}, \rho))$ ensures timely delivery is lower bounded by Equation (4.15) when choosing $\texttt{E}$ and ρ as discussed above.

Corollary 4.3.10. *Let* $\texttt{E}(p) := \lceil \alpha_p \cdot n \rceil$*, let* $d \in [7, \infty]$*, and let* $\rho := \frac{d}{\gamma \cdot n}$*. If* m *is a message that is input to some honest party in either* $\Pi_{Flood}(\mathsf{ER\text{-}Emulation}(\texttt{E}, \rho))$*, then*

$$\begin{aligned} &\Pr\left[\texttt{Timely}_m\left(\left(7 \cdot \log\left(\frac{n}{d}\right) + 2\right) \cdot \delta_{Channel}\right)\right] \\ &\geq 1 - \left(2 \cdot n \cdot \left(e^{-\frac{d}{18}} + \left(6 \cdot \log\left(\frac{n}{d}\right) + 1\right) \cdot e^{-\frac{7 \cdot d}{108}}\right) + e^{-\gamma \cdot n \cdot \left(\frac{d}{9} - 2\right)}\right). \end{aligned} \tag{4.18}$$

4.3.4 Security of WFF

In the previous section, we proved that ER-Emulation induces a secure protocol. Unfortunately, it is not a practical neighborhood selection algorithm, as it requires each party to do n coin-flips per message sent and forwarded. In this section, we introduce two intermediate algorithms to prove WFF secure (Fast-ER-Emulation and Practical-ER-Emulation) by doing gradual changes to ER-Emulation, until we finally arrive at the algorithm WFS which is both practical, simple, and similar to algorithms deployed in practice (except that this algorithm maintains its complexity even for weighted corruptions).

4.3.4.1 Intermediary Neighborhood Selection Algorithms

We first introduce the algorithm Fast-ER-Emulation, which is distributed identically to ER-Emulation but is more practical. The algorithm exploits that another way of creating an Erdős–Rényi graph is first to decide how many edges each node should have using the binomial distribution and then select these neighbors at random.

Below we will abuse notation slightly and write $\texttt{E}(P)$ to denote the set of emulated nodes for a set of parties $P \subseteq \mathcal{P}$ and an emulation function $\texttt{E}$,

$$\texttt{E}(P) \triangleq \{p_i \mid p \in P \land i \in \{1, 2, \ldots, \texttt{E}(p)\}\}.^{7} \tag{4.19}$$

[7]This set may be different from the actual set of nodes that will be emulated in an execution of the protocol, as dishonest parties might choose to deviate from the protocol. However, it is still useful to define the set to define honest behavior.

Algorithm Fast-ER-Emulation$_p(\mathtt{E}, \rho)$

```
1: Let $N := \varnothing$.
2: for $i := 0;\ i < \mathtt{E}(p);\ i{+}{+}$ do
3:     Sample $k \overset{\$}{\leftarrow} \mathcal{B}(|\mathtt{E}(\mathcal{P} \setminus \{p\})|, \rho)$.
4:     Let $A$ be $k$ nodes sampled from $\mathtt{E}(\mathcal{P} \setminus \{p\})$ without replacement.
5:     Set $N := N \cup \left\{p' \mid p'_j \in A \wedge j \in \mathbb{N}\right\}$.
6: return $N$.
```

We now show Fast-ER-Emulation and ER-Emulation are identically distributed. The proof idea is to show that the respective neighborhood selection algorithms are distributed identically. This is shown by showing that for both distributions, each edge between emulated nodes appears with independent probability ρ.

Lemma 4.3.11. *Let $\rho \in [0,1]$, let $\lambda \in \mathbb{N}$, let $p \in \mathcal{H}$, and let $\mathtt{E} : \mathcal{P} \to \mathbb{N} \setminus \{0\}$ be an emulation function. If $G_1 \overset{\$}{\leftarrow}$ HSP$(p,$ ER-Emulation$(\mathtt{E}, \rho), \lambda)$ and $G_2 \overset{\$}{\leftarrow}$ HSP$(p,$ Fast-ER-Emulation$(\mathtt{E}, \rho), \lambda)$, then $G_1 \sim G_2$.*

Proof. It is sufficient to prove that for any p' we have that for $N_1 \overset{\$}{\leftarrow}$ ER-Emulation$_{p'}(\mathtt{E}, \rho)$ and $N_2 \overset{\$}{\leftarrow}$ Fast-ER-Emulation$_{p'}(\mathtt{E}, \rho)$ then $N_1 \sim N_2$. Both neighborhood selection algorithms work by letting a party be included in the neighborhood with the same probability as if there would have been an edge between any of the emulated nodes of the two parties. By Equation (4.5) edges appear with ρ for N_1. It is hence sufficient to look at the probability that any of the emulated nodes of p' select a node not belonging to p' with mutually independent probability ρ.

Let us look at a node $v \in \mathtt{E}(\{p'\})$ and calculate the probability that there are edges to a specific set of other nodes $U \subseteq \mathtt{E}(\mathcal{P} \setminus \{p'\})$ from v. Let A be the set of nodes v chooses. We now want to show that:

$$\Pr[U \subseteq A] = \rho^{|U|}. \tag{4.20}$$

We let $\eta := |\mathtt{E}(\mathcal{P} \setminus \{p'\})|$ and now apply the law of total probabilities for conditional events to obtain

$$\begin{aligned} \Pr[U \subseteq A] &= \sum_{i=0}^{\eta} \Pr[U \subseteq A \mid |A| = i] \cdot \Pr[|A| = i] \\ &= \sum_{i=|U|}^{\eta} \Pr[U \subseteq A \mid |A| = i] \cdot \Pr[|A| = i] \end{aligned} \tag{4.21}$$

For any $i \geq |U|$, we have that A is chosen uniformly among sets of size i. Of those sets exactly $\binom{\eta-|U|}{i-|U|}$ includes U. Hence,

$$\Pr[U \subseteq A \mid |A| = i] = \frac{\binom{\eta-|U|}{i-|U|}}{\binom{\eta}{i}}. \tag{4.22}$$

Inserting this, we obtain

$$\begin{aligned}\Pr[U \subseteq A] &= \sum_{i=|U|}^{\eta} \frac{\binom{\eta-|U|}{i-|U|}}{\binom{\eta}{i}} \binom{\eta}{i} \cdot \rho^i \cdot (1-\rho)^{\eta-i} \\ &= \sum_{i=|U|}^{\eta} \binom{\eta-|U|}{i-|U|} \cdot \rho^i \cdot (1-\rho)^{\eta-i}.\end{aligned} \tag{4.23}$$

We now change the variable letting $r := i + |U|$, factor out $\rho^{|U|}$, and use the binomial formula (Lemma 4.2.2) to obtain

$$\begin{aligned}\Pr[U \subseteq A] &= \sum_{r=0}^{\eta-|U|} \binom{\eta-|U|}{r} \cdot \rho^{r+|U|} \cdot (1-\rho)^{\eta-(r+|U|)} \\ &= \rho^{|U|} \cdot \sum_{r=0}^{\eta-|U|} \binom{\eta-|U|}{r} \cdot \rho^{r+|U|} \cdot (1-\rho)^{\eta-|U|-r} \\ &= \rho^{|U|} \cdot (\rho + (1-\rho))^{\eta-|U|} \\ &= \rho^{|U|}.\end{aligned} \tag{4.24}$$

Therefore we can conclude that all edges between emulated nodes appear with a mutually independent probability ρ and hence $G_1 \sim G_2$. $\square$

A problem of Fast-ER-Emulation is that each party p needs to make $\mathtt{E}(p)$ number of draws from the binomial distribution. One way to avoid this is to make a single random draw for the number of nodes all emulated nodes should send to and then afterward choose this number of nodes uniformly without replacement. Below we present the algorithm Practical-ER-Emulation, which does precisely that.

Algorithm Practical-ER-Emulation$_p(\mathtt{E}, \rho)$

```
1: Let $N := \varnothing$.
2: Sample $k \xleftarrow{\$} \mathcal{B}(\mathtt{E}(p) \cdot |\mathtt{E}(\mathcal{P} \setminus \{p\})|, \rho)$.
3: Let $A$ be $k$ nodes sampled from $\mathtt{E}(\mathcal{P} \setminus \{p\})$ without replacement.
4: Set $N := \{p \mid p_i \in A \wedge i \in \mathbb{N}\}$.
5: **return** $N$.
```

Practical-ER-Emulation is not distributed identically to Fast-ER-Emulation, as there is a smaller expected overlap between the selected emulated nodes. However, it still holds that the graph resulting from the honest sending process based upon Practical-ER-Emulation has a higher chance of having a low distance from the sender than the graph resulting from the honest sending process based upon Fast-ER-Emulation. We make this intuition formal in the lemma below. The basic idea of the proof is to define a coupling between the two graphs by defining a coupling between the respective neighborhood selection algorithms while ensuring that the set of neighbors sampled by Practical-ER-Emulation is a superset of the neighbors of those sampled by Fast-ER-Emulation. We define the coupling using rejection sampling and ensure that any neighbor that is rejected when sampling neighbors for Fast-ER-Emulation will also be rejected when sampling neighbors for Practical-ER-Emulation.

Lemma 4.3.12. *Let $\rho \in [0,1]$, let $\lambda \in \mathbb{N}$, let $p \in \mathcal{H}$, and let $\mathtt{E} : \mathcal{P} \to \mathbb{N} \setminus \{0\}$ be an emulation function. If $G_1 \xleftarrow{\$} \mathsf{HSP}(p, \mathsf{Fast\text{-}ER\text{-}Emulation}(\mathtt{E}, \rho), \lambda)$ and $G_2 \xleftarrow{\$} \mathsf{HSP}(p, \mathsf{Practical\text{-}ER\text{-}Emulation}(\mathtt{E}, \rho), \lambda)$ then*

$$\Pr[\mathtt{MaxDist}(G_1, p) \le \lambda] \le \Pr[\mathtt{MaxDist}(G_2, p) \le \lambda]. \tag{4.25}$$

Proof. Let $G_1 = (\mathcal{H}, E_1)$ and $G_2 = (\mathcal{H}, E_2)$. To show the lemma we define a coupling $\widetilde{G_1} = (\mathcal{H}, \widetilde{E_1})$ and $\widetilde{G_2} = (\mathcal{H}, \widetilde{E_2})$ such that $G_1 \sim \widetilde{G_1}$ and $G_2 \sim \widetilde{G_2}$ and $\mathtt{MaxDist}(\widetilde{G_1}, p) \le \lambda \implies \mathtt{MaxDist}(\widetilde{G_2}, p) \le \lambda$. To show the implication it is sufficient to show that $\widetilde{E_1} \subseteq \widetilde{E_2}$. We define the coupling by sampling $\widetilde{G_1}$ and $\widetilde{G_2}$ in parallel with a defined coupling between the neighborhood selection algorithms for any party $p' \in \mathcal{H}$. We let $N_1 \xleftarrow{\$} \mathsf{Fast\text{-}ER\text{-}Emulation}_{p'}(\mathtt{E}, \rho)$ and $N_2 \xleftarrow{\$} \mathsf{Practical\text{-}ER\text{-}Emulation}_{p'}(\mathtt{E}, \rho)$. Again we define a coupling $\widetilde{N_1} \sim N_1$ and $\widetilde{N_2} \sim N_2$ and show that $\widetilde{N_1} \subseteq \widetilde{N_2}$. This is sufficient to ensure that $\widetilde{E_1} \subseteq \widetilde{E_2}$ because this ensures that all parties that select their neighbors when constructing $\widetilde{G_1}$ also gets to choose their neighbors in the construction of $\widetilde{G_2}$.

We define $\widetilde{N_1}$ and $\widetilde{N_2}$ via the following process.

1. Initially, let $\widetilde{N_1} = \widetilde{N_2} := \varnothing$.

2. Initialize variables similar to the variables in Line 3 of $\mathsf{Fast\text{-}ER\text{-}Emulation}_{p'}(\mathtt{E}, \rho)$ by sampling $\widetilde{k_1^1} \xleftarrow{\$} \mathcal{B}\left(|\mathtt{E}(\mathcal{P} \setminus \{p\})|, \rho\right), \ldots, \widetilde{k_1^{\mathtt{E}(p')}} \xleftarrow{\$} \mathcal{B}\left(|\mathtt{E}(\mathcal{P} \setminus \{p\})|, \rho\right)$, and a variable $\widetilde{k_2} := \sum_{i=1}^{\mathtt{E}(p')} \widetilde{k_1^i}$ similar to the k in Line 2 of $\mathsf{Practical\text{-}ER\text{-}Emulation}_{p'}(\mathtt{E}, \rho)$

3. Additionally, we initialize $\widetilde{A_1^1} := \varnothing, \ldots, \widetilde{A_1^{\mathtt{E}(p')}} := \varnothing$ that corresponds to the variable A in Line 4 of $\mathsf{Fast\text{-}ER\text{-}Emulation}_{p'}(\mathtt{E}, \rho)$ and $\widetilde{A_2} := \varnothing$ that corresponds to the variable A in Line 3 of $\mathsf{Practical\text{-}ER\text{-}Emulation}_{p'}(\mathtt{E}, \rho)$.

4. Now for $i \in \{1, \ldots, \mathtt{E}(p')\}$ and for $j \in \left\{1, \ldots, \widetilde{k_1^i}\right\}$ do:
   a) Set $\mathtt{Accepted}_1 := \bot$ and $\mathtt{Accepted}_2 := \bot$.
   b) While $\neg\mathtt{Accepted}_1 \vee \neg\mathtt{Accepted}_2$ do:
      i. Sample v uniformly in $\mathtt{E}(\mathcal{P} \setminus \{p'\})$.
      ii. If $v \notin \widetilde{A_2} \wedge \neg\mathtt{Accepted}_2$ set $\widetilde{A_2} := \widetilde{A_2} \cup \{v\}$ and $\mathtt{Accepted}_2 := \top$.
      iii. If $v \notin \widetilde{A_1^i} \wedge \neg\mathtt{Accepted}_1$ set $\widetilde{A_1^i} := \widetilde{A_1^i} \cup \{v\}$ and $\mathtt{Accepted}_1 := \top$.

5. Finally, set $\widetilde{A_1} := \bigcup_{i=1}^{\mathtt{E}(p')} \widetilde{A_1^i}$, $\widetilde{N_1} := \left\{p \mid p_i \in \widetilde{A_1} \wedge i \in \mathbb{N}\right\}$, and $\widetilde{N_2} := \left\{p \mid p_i \in \widetilde{A_2} \wedge i \in \mathbb{N}\right\}$.

Note at first that $\widetilde{N_1} \sim N_1$ as the procedure simply uses rejection sampling to sample $\widetilde{k_1^i}$ elements from $\mathtt{E}(\mathcal{P} \setminus \{p'\})$ without repetition and adds these to $\widetilde{A_1}$, and hence have the same distribution as $\mathsf{Fast\text{-}ER\text{-}Emulation}_{p'}(\mathtt{E}, \rho)$. Furthermore, note that the procedure simply samples $\widetilde{k_2} = \sum_{i=1}^{\mathtt{E}(p')} \widetilde{k_1^i}$ elements from $\mathtt{E}(\mathcal{P} \setminus \{p'\})$ without repetition via rejection sampling and add these to $\widetilde{A_2}$. Because $\widetilde{k_2}$ is the sum of $\mathtt{E}(p')$ independent random variables that are each distributed as $\mathcal{B}(|\mathtt{E}(\mathcal{P} \setminus \{p'\})|, \rho)$ we get that $\widetilde{k_2} \sim \mathcal{B}(\mathtt{E}(p') \cdot |\mathtt{E}(\mathcal{P} \setminus \{p'\})|, \rho)$ and hence that $\widetilde{N_2} \sim N_2$.

Furthermore, the algorithm ensures that $\widetilde{N_1} \subseteq \widetilde{N_2}$ because at any point in the algorithm we have that $\widetilde{A_2}$ is a superset of any $\widetilde{A_1^i}$. □

Note that Lemmas 4.3.5, 4.3.9, 4.3.11 and 4.3.12 together imply that the probability that $\Pi_{\text{Flood}}(\mathsf{Fast\text{-}ER\text{-}Emulation}(\mathtt{E}, \rho))$ and $\Pi_{\text{Flood}}(\mathsf{Practical\text{-}ER\text{-}Emulation}(\mathtt{E}, \rho))$ do not ensure timely delivery is negligible for a certain choice of $\mathtt{E}$ and ρ.

Corollary 4.3.13. *Let $\mathtt{E}(p) := \lceil \alpha_p \cdot n \rceil$, let $d \in [7, \infty]$, and let $\rho := \frac{d}{\gamma \cdot n}$. If m is a message that is input to some honest party in either*

- $\Pi_{\mathit{Flood}}(\mathsf{Fast\text{-}ER\text{-}Emulation}(\mathtt{E}, \rho))$,
- *or* $\Pi_{\mathit{Flood}}(\mathsf{Practical\text{-}ER\text{-}Emulation}(\mathtt{E}, \rho))$

then

$$\begin{aligned}&\Pr\left[\mathtt{Timely}_m\left(\left(7 \cdot \log\left(\frac{n}{d}\right) + 2\right) \cdot \delta_{\mathit{Channel}}\right)\right] \\ &\geq 1 - \left(2 \cdot n \cdot \left(e^{-\frac{d}{18}} + \left(6 \cdot \log\left(\frac{n}{d}\right) + 1\right) \cdot e^{-\frac{7 \cdot d}{108}}\right) + e^{-\gamma \cdot n \cdot \left(\frac{d}{9} - 2\right)}\right).\end{aligned} \tag{4.26}$$

Note that Practical-ER-Emulation is very similar to WFS. The main difference is that in Practical-ER-Emulation, the number of neighbors is sampled according to the binomial distribution, whereas WFS chooses a fixed number of neighbors. We use this observation to relate the probability that the graph constructed by the honest sending process of Practical-ER-Emulation has a low distance from the sender to the probability that the honest sending process of WFS has a low distance from the sender. Similarly to the proof of Lemma 4.3.12, the proofs rely on a coupling between the two graphs defined by a coupling between their neighborhood selection algorithms using rejection sampling. However, in this coupling, the invariant that the edges sampled by WFS are a superset of those of Practical-ER-Emulation is only maintained when no party samples more than k neighbors in Practical-ER-Emulation. Hence, we bound this probability using the Chernoff bound (Lemma 1.3.1).

Lemma 4.3.14. *Let $\rho \in [0, 1]$, let $\epsilon \in [0, 1]$ let $\lambda \in \mathbb{N}$, let $p \in \mathcal{H}$, let $k \geq \lceil (1 + \epsilon) \cdot n_{\mathtt{E}} \cdot \rho \rceil$, and let $\mathtt{E} : \mathcal{P} \to \mathbb{N} \setminus \{0\}$ be an emulation function. If $G_1 \xleftarrow{\$} \mathsf{HSP}(p, \mathsf{Practical\text{-}ER\text{-}Emulation}(\mathtt{E}, \rho), \lambda)$ and $G_2 \xleftarrow{\$} \mathsf{HSP}(p, \mathsf{WFS}(\mathtt{E}, k), \lambda)$ then*

$$\Pr[\mathtt{MaxDist}(G_1, p) \leq \lambda] - |\mathcal{H}| \cdot e^{-\frac{\epsilon^2 \cdot (n-1) \cdot \rho}{3}} \leq \Pr[\mathtt{MaxDist}(G_2, p) \leq \lambda]. \tag{4.27}$$

Proof. Let $G_1 = (\mathcal{H}, E_1)$ and $G_2 = (\mathcal{H}, E_2)$. To show the lemma we define again a coupling $\widetilde{G_1} = (\mathcal{H}, \widetilde{E_1})$ and $\widetilde{G_2} = (\mathcal{H}, \widetilde{E_2})$ such that $G_1 \sim \widetilde{G_1}$ and $G_2 \sim \widetilde{G_2}$. For each party $p' \in \mathcal{H}$, we introduce a random variable $X_{p'}$ which denotes the number of outgoing edges this party has in $\widetilde{G_1}$. We further define C to be the event $\bigcap_{p' \in \mathcal{H}}(X_{p'} \leq$

$(1+\epsilon)\cdot \mathtt{E}(p')\cdot |\mathtt{E}(\mathcal{P}\setminus\{p'\})|\cdot\rho)$ (this event can be thought of as that no party picks "outrageously many neighbors"). We now wish to show two things:

$$\mathtt{MaxDist}(\widetilde{G_1},p)\leq\lambda\wedge C\implies\mathtt{MaxDist}(\widetilde{G_2},p)\leq\lambda, \tag{4.28}$$

and

$$\Pr[\mathtt{MaxDist}(\widetilde{G_1},p)\leq\lambda\wedge C]\geq\Pr[\mathtt{MaxDist}(G_1,p)\leq\lambda]-|\mathcal{H}|\cdot e^{-\frac{\epsilon^2\cdot(n-1)\cdot\rho}{3}}. \tag{4.29}$$

To show Equation (4.28) it is sufficient to show that $C\implies\widetilde{E_1}\subseteq\widetilde{E_2}$. As in the proof of Lemma 4.3.12, we define the coupling by sampling $\widetilde{G_1}$ and $\widetilde{G_2}$ in parallel with a defined a coupling between the neighborhood selection algorithms for any party $p'\in\mathcal{H}$. We let $N_1\xleftarrow{\$}\mathsf{Practical\text{-}ER\text{-}Emulation}_{p'}(\mathtt{E},\rho)$ and $N_2\xleftarrow{\$}\mathsf{WFS}_{p'}(\mathtt{E},k)$. Again we define a coupling $\widetilde{N_1}\sim N_1$ and $\widetilde{N_2}\sim N_2$ and show that $C\implies\widetilde{N_1}\subseteq\widetilde{N_2}$ which again will imply that $C\implies\widetilde{E_1}\subseteq\widetilde{E_2}$. Again we define a coupling $\widetilde{N_1}\sim N_1$ and $\widetilde{N_2}\sim N_2$ and show that $\widetilde{N_1}\subseteq\widetilde{N_2}$. This is sufficient to ensure that $\widetilde{E_1}\subseteq\widetilde{E_2}$ because this ensures that all parties that select their neighbors when constructing $\widetilde{G_1}$ also gets to choose their neighbors in the construction of $\widetilde{G_2}$.

We define $\widetilde{N_1}$ and $\widetilde{N_2}$ via the following process.

1. Initially, let $\widetilde{N_1}=\widetilde{N_2}:=\varnothing$.
2. We further initialize variables corresponding to the variables k and A used in Lines 2 and 3 of $\mathsf{Practical\text{-}ER\text{-}Emulation}_{p'}(\mathtt{E},\rho)$ by sampling $\widetilde{k_1}\xleftarrow{\$}\mathcal{B}\left(\mathtt{E}(p')\cdot|\mathtt{E}(\mathcal{P}\setminus\{p'\})|,\rho\right)$ and set $\widetilde{A_1}:=\varnothing$. We also initialize a variable $\widetilde{K_2}:=\mathtt{E}(p')\cdot k$ identical to the variable K in Line 2 of $\mathsf{WFS}_{p'}(\mathtt{E},k)$.
3. Now for $i\in\left\{1,\ldots,\max(\widetilde{k_1},\widetilde{K_2})\right\}$ do:
   a) Set $\mathtt{Accepted}_1:=i>\widetilde{k_1}$ and $\mathtt{Accepted}_2:=i>\widetilde{K_2}$.
   b) While $\neg\mathtt{Accepted}_1\vee\neg\mathtt{Accepted}_2$ do:
      i. Sample p_j uniformly in $\mathtt{E}(\mathcal{P}\setminus\{p'\})$.
      ii. If $p\notin\widetilde{N_2}\wedge\neg\mathtt{Accepted}_2$ set $\widetilde{N_2}:=\widetilde{N_2}\cup\{p\}$ and $\mathtt{Accepted}_2:=\top$. [8]
      iii. If $v\notin\widetilde{A_1}\wedge\neg\mathtt{Accepted}_1$ set $\widetilde{A_1}:=\widetilde{A_1}\cup\{v\}$ and $\mathtt{Accepted}_1:=\top$.
4. Finally, set $\widetilde{N_1}:=\left\{p\mid p_i\in\widetilde{A_1}\wedge i\in\mathbb{N}\right\}$.

Note at first that $\widetilde{N_1}\sim N_1$ as the procedure simply uses rejection sampling to sample $\widetilde{k_1}$ elements from $\mathtt{E}(\mathcal{P}\setminus\{p'\})$ without repetition and adds these to $\widetilde{A_1}$, and hence have the same distribution as $\mathsf{Practical\text{-}ER\text{-}Emulation}_{p'}(\mathtt{E},\rho)$. Similarly, it is clear that $\widetilde{N_2}\sim N_2$ as once again rejection sampling is used to pick $\mathtt{E}(p')\cdot k$ from $\mathcal{P}\setminus\{p'\}$ weighted according to $\mathtt{E}$. It is also immediate to see that $C\implies\widetilde{N_1}\subseteq\widetilde{N_2}$ as if C happens then surely the max of $\widetilde{k_1}$ and $\widetilde{K_2}$ is $\widetilde{K_2}$, as $n_{\mathtt{E}}>|\mathtt{E}(\mathcal{P}\setminus\{p'\})|$. Hence all parties that will be added to $\widetilde{N_1}$ will also be added to $\widetilde{N_2}$.

[8] Note that p in this step does not refer to the sender in the honest sending process but rather the party that is supposed to emulate node p_j.

It is thus left to show Equation (4.29). We have that

$$\Pr[\texttt{MaxDist}(\widetilde{G_1}, p) \leq \lambda \wedge C] = \Pr[\texttt{MaxDist}(\widetilde{G_1}, p) \leq \lambda] - \Pr[\neg C]. \tag{4.30}$$

As we have already argued that $G_1 \sim \widetilde{G_1}$ it is sufficient to show that

$$\Pr[\neg C] \leq |\mathcal{H}| \cdot e^{-\frac{\epsilon^2 \cdot (n-1) \cdot \rho}{3}}. \tag{4.31}$$

Further, using a union bound, we have that.

$$\begin{aligned} \Pr[\neg C] &= \Pr\left[\bigcup_{p' \in \mathcal{H}} (X_{p'} > (1+\epsilon) \cdot \texttt{E}(p') \cdot |\texttt{E}(\mathcal{P} \setminus \{p'\})| \cdot \rho)\right] \\ &\leq \sum_{p' \in \mathcal{H}} \Pr\left[X_{p'} > (1+\epsilon) \cdot \texttt{E}(p') \cdot |\texttt{E}(\mathcal{P} \setminus \{p'\})| \cdot \rho\right]. \end{aligned} \tag{4.32}$$

Furthermore, for each party, it is clear that the number of neighbors selected is less than the selected emulated neighbors. We let $Y_{p'}$ denote this number of emulated neighbors. Hence, it is sufficient to show that for any $p' \in \mathcal{H}$ we have,

$$\Pr\left[Y_{p'} > (1+\epsilon) \cdot \texttt{E}(p') \cdot |\texttt{E}(\mathcal{P} \setminus \{p'\})| \cdot \rho\right] \leq e^{-\frac{\epsilon^2 \cdot (n-1) \cdot \rho}{3}}. \tag{4.33}$$

Now, as for any $p'' \in \mathcal{P}$ we have that $\texttt{E}(p'') \geq 1$ it follows that $|\texttt{E}(\mathcal{P} \setminus \{p'\})| \geq n-1$. This makes desired equation follow from the Chernoff bound (Lemma 1.3.1)

$$\begin{aligned} \Pr\left[Y_{p'} > (1+\epsilon) \cdot \texttt{E}(p') \cdot |\texttt{E}(\mathcal{P} \setminus \{p'\})| \cdot \rho\right] &\leq e^{-\frac{\epsilon^2 \cdot |\texttt{E}(\mathcal{P} \setminus \{p'\})| \cdot \texttt{E}(p') \cdot \rho}{3}} \\ &\leq e^{-\frac{\epsilon^2 \cdot (n-1) \cdot \rho}{3}}. \end{aligned} \tag{4.34}$$

□

We now provide a corollary that bounds the concrete probability that a message input via WFF is delivered timely.

Corollary 4.3.15. *Let $k \in \mathbb{N}$ such that $k \geq \frac{42}{\gamma}$. If m is a message that is input to some honest party in* $\mathsf{WFF}(k)$ *then*

$$\begin{aligned} &\Pr\left[\texttt{Timely}_m\left(\left(7 \cdot \log\left(\frac{n \cdot 6}{k \cdot \gamma}\right) + 2\right) \cdot \delta_{Channel}\right)\right] \\ &\geq 1 - n \cdot e^{-\frac{(n-1) \cdot k}{n \cdot 24}} - e^{-\gamma \cdot n \cdot \left(\frac{k \cdot \gamma}{54} - 2\right)} - \left(2 \cdot n \cdot \left(e^{-\frac{k \cdot \gamma}{108}} + \left(6 \cdot \log\left(\frac{n \cdot 6}{k \cdot \gamma}\right) + 1\right) \cdot e^{-\frac{7 \cdot k \cdot \gamma}{648}}\right)\right). \end{aligned} \tag{4.35}$$

Proof. Let $\texttt{E}(p) := \lceil \alpha_p \cdot n \rceil$. Lemma 4.3.5 ensures that is enough to reason about the honest sending process with the neighborhood selection algorithm $\mathsf{WFS}(\texttt{E}, k)$. We observe that

$$n_{\texttt{E}} = \sum_{p \in \mathcal{P}} \lceil \alpha_p \cdot n \rceil \leq \sum_{p \in \mathcal{P}} \alpha_p \cdot n + 1 = 2 \cdot n. \tag{4.36}$$

We let $d := \frac{k \cdot \gamma}{6}$, and note that the precondition for Lemma 4.3.14 is satisfied for $\rho := \frac{d}{\gamma \cdot n}$ so we instantiate this letting $\epsilon := \frac{1}{2}$, and using the bound above on $n_{\mathtt{E}}$. Furthermore, note that $|\mathcal{H}| \leq n$ and hence it is sufficient to bound the probability that the honest sending process of $\mathsf{Practical\text{-}ER\text{-}Emulation}(\mathtt{E}, \rho)$ has a large distance from the sender. Equation (4.35) now follows by Lemmas 4.3.9, 4.3.11 and 4.3.12. □

As observed earlier, it is sufficient to bound the probability that a message is timely delivered to bound the probability that any of the two properties of a flooding protocol is achieved. Further, note that a party p sends at most $k \cdot \mathtt{E}(p)$ messages when a message is forwarded. Hence, the message complexity is bounded by $\sum_{p \in \mathcal{H}} k \cdot \mathtt{E}(p) \leq 2 \cdot k \cdot n$.

$$\sum_{p \in \mathcal{H}} k \cdot \mathtt{E}(p) \leq 2 \cdot k \cdot n. \tag{4.37}$$

Therefore, the security of WFF (and thereby Theorem 4.3.3) follows directly from this corollary.

4.4 Asymptotic Optimality and Practical Considerations

Our results from Section 4.3.2.1 show that the protocol $\mathsf{WFF}(k)$ provides provably secure flooding. With respect to efficiency, the results show that there are two possible drawbacks: First, the emulation function $\mathtt{E}(p) = \lceil \alpha_p \cdot n \rceil$ forces parties with very high weight to send to many parties, which lead to bandwidth issues. Secondly, Theorem 4.3.3 shows that in our protocol, the number of parties each node has to send to increases logarithmically in the total number of nodes. In this section, we show that both properties are inherent to "flooding protocols".

4.4.1 Workload of Heavy Parties

It is easy to see that in at least in extreme cases, very heavy parties have to send to a lot of other parties: If there is a single party that has the majority of the total weight, it could be that only this party and an additional one are honest. Since the heavy party is the only one that can be relied upon for message delivery, it needs to send to all other parties. The following lemma generalizes this idea to less extreme settings.

Lemma 4.4.1. *For any protocol Π that guarantees delivery to* all honest *parties, and for any subset $S \subseteq \mathcal{P}$ such that $\sum_{p \in S} \alpha_p \geq \gamma$, we have with overwhelming probability that*

$$\sum_{p \in S} \mathtt{degree}_{\Pi}(p) \geq |\mathcal{P} \setminus S|. \tag{4.38}$$

Proof. Let S be any such set. By the honesty assumption, it could be that there is precisely one honest party in $\mathcal{P} \setminus S$. To guarantee delivery to this party, some party in S must send to it. Since it cannot be distinguished which party in $\mathcal{P} \setminus S$ is honest, the parties in S must send to all parties in $\mathcal{P} \setminus S$. □

Another consequence of Lemma 4.4.1 is that having a huge number of nodes with very little weight also increases the workload for all other nodes, as shown below.

Corollary 4.4.2. *Assume there is a large set $T \subseteq \mathcal{P}$ of parties with combined relative weight $\leq 1-\gamma$ and $|T| \geq n-\epsilon$ for some constant $\epsilon > 0$, and define $S := \mathcal{P} \setminus T$. Then, the average degree of the parties in S must be at least $\frac{n-\epsilon}{\epsilon} \in \Omega(n)$ with overwhelming probability.*

Proof. Since $\sum_{p \in S} \alpha_p = 1 - \sum_{p \in T} \alpha_p \geq \gamma$, Lemma 4.4.1 implies that the average degree of the parties in S is at least $\frac{|\mathcal{P} \setminus S|}{|S|}$ with overwhelming probability. By assumption, we have $\frac{|\mathcal{P} \setminus S|}{|S|} = \frac{|T|}{n-|T|} \geq \frac{n-\epsilon}{\epsilon} \in \Omega(n)$. □

Limiting the workload. As we have seen above, having very heavy or many very light parties necessarily yields a large number of outgoing connections for some of the nodes. This is not only undesirable but may also become prohibitive in practice due to limited network bandwidth. If the flooding is deployed, say for a proof-of-stake blockchain, this can be mitigated by putting a lower and an upper limit on the amount of stake for actively participating nodes. This implies that people holding a lot of stake need to split their stake over several nodes (which is anyway beneficial for decentralization if they are run in different locations), and people with too little stake need to, e.g., delegate their stake to another node if supported by the blockchain. The latter can still passively participate by fetching data from other nodes as discussed for zero-weight parties in Section 4.5.2.

4.4.2 Logarithmic Growth of Message Complexity

It is well known that Erdős–Rényi graphs are connected with high probability if and only if edges are included with probability larger than $\frac{\log n}{n}$ [Bol01, Theorem 7.3]. This means the expected degree of a node must be larger than $\log n$ to obtain a connected graph, even without considering corruptions. Since our proofs in Section 4.3 depart from Erdős–Rényi graphs, one cannot hope to prove a better message complexity with our proof techniques.

On the other hand, our final protocol $\mathsf{WFF}(k)$ does not choose neighbors in the way Erdős–Rényi graphs are constructed but more closely corresponds to so-called directed k-out graphs, which have also been considered in the literature. Those are directed graphs where for each node v independently, k uniformly random other nodes are sampled, and directed edges from v to the k sampled nodes are added. It is known that such graphs are connected with probability approaching 1 for $n \to \infty$ already for constant $k = 2$ [FF82]. Hence, at least without corruptions, $O(n)$ overall message complexity should be enough for our protocol. When considering corruptions, however, a result by Yagan and Makowski [YM13] implies that $\log n$ connections for each node are necessary, as we show below. This shows that $\mathsf{WFF}(k)$ and Theorem 4.3.3 are asymptotically optimal, at least for the particular case in which all parties have the same weight.

Lemma 4.4.3. *For any flooding protocol in which all honest parties send to k uniformly chosen nodes and delivery to all honest nodes is guaranteed with probability $\geq 1/2$ where up to a $(1-\gamma)$ fraction of nodes can be corrupted, we have for sufficiently large n that*

$$k \geq \frac{\log n}{\gamma + 1/n - \log(1 - \gamma - 1/n)}.$$

Proof. Yagan and Makowski [YM13] have considered the setting in which for each of the n nodes p_i, k distinct random other nodes are sampled and undirected edges between p_i and all k sampled nodes are added to a graph. They then consider the subgraph H consisting of the first $\lfloor \gamma' n \rfloor$ nodes for some constant $\gamma' \in (0,1)$ and show in [YM13, Theorem 3.2] that

$$k < \frac{\log n}{\gamma' - \log(1-\gamma')} \implies \lim_{n\to\infty} \Pr[H \text{ contains isolated node}] = 1. \tag{4.39}$$

To translate this to our setting, first note that corrupting at most $\lfloor (1-\gamma)n \rfloor$ nodes from the end to leave the first $\lfloor \gamma n + 1 \rfloor$ parties honest is a valid adversarial strategy. To be compatible with the result above, we can set $\gamma' := \gamma + 1/n$. Further note that a node p being isolated in H has the same probability as an honest node not sending to any other honest node and no honest node sending to that one in a flooding protocol. In that case, if p is the sender in the flooding protocol, no honest node will receive the message, and if some other node is the sender, p will not receive the message. Hence, the flooding protocol will fail to deliver the message to all honest nodes in both cases. This implies that, for sufficiently large n, flooding protocols with $k < \frac{\log n}{\gamma + 1/n - \log(1-\gamma-1/n)}$ fail to deliver messages with high probability. □

4.5 Delivery to Parties With Zero Weight

So far, we have excluded parties with zero weight from participating in our protocol. While they are not relevant for the security of consensus protocols running on top of the network, it is still important in practice to allow such passive nodes to obtain the data from the blockchain, e.g., for connecting wallets. We discuss some options that allow zero-weight parties to obtain data, each with their advantages and disadvantages.

4.5.1 Adjusting the Emulation Function

Recall from Section 4.3.3 that we use the emulation function

$$\mathtt{E}(p) = \lceil \alpha_p \cdot n \rceil. \tag{4.40}$$

This implies that parties with weight 0, i.e., $\alpha_p = 0$ emulate 0 parties and consequently do not send anything and also do not receive anything. If we want to guarantee delivery to parties with zero weight, we can instead use the emulation function

$$\mathtt{E}'(p) := \lfloor \alpha_p \cdot n \rfloor + 1. \tag{4.41}$$

This ensures that all parties emulate at least one party. Furthermore, inequalities (4.12) and (4.13) from Section 4.3.3 also hold for $\mathtt{E}'$ and our results from that section follow similarly as for $\mathtt{E}$. Hence, using the emulation function $\mathtt{E}'$ guarantees delivery to all parties, including those with weight 0.

A downside of this approach is that considering parties with weight 0 opens up the system for Sybil attacks: An attacker can easily add additional zero-weight nodes to the system and thereby increase n arbitrarily without changing any α_p. According to $\mathtt{E}'$, the work required from honest parties with non-zero weight thus increases linearly in n, allowing the attacker to increase the workload of honest parties arbitrarily. Therefore, such an approach is only practical if there is some mechanism for preventing Sybil attacks.

4.5.2 Fetching Data

Since guaranteeing that all zero-weight parties receive all data in the flooding process can substantially increase the workload for honest parties, we here provide an alternative. The idea is to exclude zero-weight parties from the regular protocol as we do in our main results and to allow those parties to obtain the state by querying other nodes. To prevent Sybil attacks, parties with non-zero weight can refuse to answer if they receive too many requests. This ensures that the flooding among parties with non-zero weight, critical for consensus of the blockchain, cannot be negatively affected by zero-weight parties; the worst outcome of Sybil attacks is that honest zero-weight parties cannot obtain data from the blockchain.

We formalize this idea in the algorithm Fetch below.

```
Algorithm Fetch($k$)
1: Let $N := \varnothing$.
2: Sample $k$ parties $p_1, \dots, p_k \in \mathcal{P}$ weighted w.r.t. $\alpha_{p_i}$ and add these to $N$.
3: Request data from all $N$ parties and return the union.
```

The probability that a party does not fetch some data that is already sufficiently spread drops exponentially fast in k. We formalize this in the lemma below.

Lemma 4.5.1. *[Fetching from Constant Number of Parties] Let $k \in \mathbb{N}$ and let β be the fraction of weight assigned to honest parties and hold some piece of data. The probability that the state returned by* Fetch(k) *does not include that data is at most*

$$(1-\beta)^k. \tag{4.42}$$

Proof. Let D be the set of parties that are honest and hold the data and let X_i for $i \in \{1, \dots, k\}$ be the random variable denoting the ith party that is picked by Fetch(k). Using the definition of conditional events, we have

$$\begin{aligned}\Pr[\text{no picked party is honest and has data}] &= \Pr\left[\bigcap_{i=1}^{k} X_i \notin D\right] \\ &= \prod_{i=1}^{k} \Pr\left[X_i \notin D \;\middle|\; \bigcap_{j<i} X_j \notin D\right].\end{aligned} \tag{4.43}$$

Furthermore, we have for $i \in \{1, \dots k\}$,

$$\begin{aligned}\Pr\left[X_i \notin D \;\middle|\; \bigcap_{j<i} X_j \notin D\right] &= 1 - \Pr\left[X_i \in D \;\middle|\; \bigcap_{j<i} X_j \notin D\right] \\ &= 1 - \sum_{p_z \in D} \Pr\left[X_i = p_z \;\middle|\; \bigcap_{j<i} X_j \notin D\right] \\ &\leq 1 - \sum_{p_z \in D} \frac{\alpha_z}{\sum_{p_v \in \mathcal{P}} \alpha_v} \\ &= 1 - \beta.\end{aligned} \tag{4.44}$$

Hence, we can conclude that

$$\Pr[\text{no picked party is honest and has data}] \leq (1-\beta)^k \,. \tag{4.45}$$

Since $\mathsf{Fetch}(k)$ returns the union of all obtained data, it is sufficient to pick a single honest party that holds the data, which concludes the proof. □

4.6 Performance Evaluation via Simulations

To show that our protocol WFF performs well in practice, we perform various benchmarks with varying weight distributions and adversarial strategies. The source code, and a description of how to run the benchmarks, can be found at `https://github.com/guilhermemtr/Weighted-Flooding-Simulator`.[9]

4.6.1 Scope of Simulations

Weight distributions. We consider weight distributions covering scenarios where parties have similar weights and scenarios with different weights. More concretely, we consider:

- The constant distribution (Const), characterized by the number of parties n. In this distribution, all parties have equal weights and are, therefore, equivalent to the non-weighted setting. This serves as a baseline for our simulations.
- The exponential distribution (Exp), characterized by the number of parties n and the weight ratio r between the heaviest party and the lightest party. It corresponds to the (perhaps more realistic) exponential weight distribution—wherein the weights of parties form an exponential curve. More concretely, for $i \in \{1, \ldots, n-1\}$, the weight of p_{i+1} is $r^{-(n-1)}$ times the weight of p_i.
- The few heavy distribution (FH), characterized by the number of parties n, the weight ratio r between the heaviest and lightest party, and the number of heavy parties c. It corresponds to the distribution where $n-c$ parties have constant weight, and the other c parties have r times more weight. This weight distribution is meant to capture extreme scenarios.

Sender. To ensure that our protocol performs well *independently* of the weight of the sender, for the exponential distribution, we consider three choices for the sender: heaviest, lightest, and median-weight party, and for the few heavy weight distribution, we consider both a heavy and a light party as the sender.

Corruption strategies. Given that parties in our protocol forward a message to their neighbors, we consider the worst behavior that prevents message propagation, i.e., corrupted parties simply do not send. We consider adversaries that can corrupt up to 50% of the total weight. To ensure that our protocol performs well independently of how adversaries spend their corruption budget, we consider adversaries that greedily corrupt as many parties as possible, following one of the strategies below:

[9]All simulations presented in this section were performed on the ETH Zurich Euler cluster, but there are no hindrances to running them on less powerful computers.

- Random corruption (Rand) where the adversary corrupts parties uniformly at random.
- Light-First corruption (Light) where the adversary corrupts parties by their weight in increasing order, starting with the lighter ones.
- Heavy-First corruption (Heavy) where the adversary corrupts parties by their weight in decreasing order, starting from the heavier ones.

One might note that the corruption strategy is irrelevant to the constant weight distribution. For this reason, we only consider the random corruption strategy for the constant weight distribution.

4.6.2 Methodology

To obtain statistical confidence, we make 10 000 runs for each parameter configuration (e.g., weight distribution, adversary strategy, choice of the sender, number of parties, etc.). All runs are executed independently.

In the evaluations, a run is considered successful if the sender's message is delivered to *all* (honest and dishonest) parties. As one might note, this contrasts with the timely predicate (see Definition 4.3.2), which only requires a message to be delivered to all honest parties. Thus, the success rate metric we consider for the evaluations is a lower bound on the actual success rate of our protocol. The rationale behind this definition is as follows: consider an adversary that corrupts a set C of parties; if the adversary would alternatively pick some party $p \in C$, and corrupt $C \setminus \{p\}$, then the protocol would have to guarantee that every honest party, including p, still gets the message. Since p is now honest, it seems a harder requirement to make p now also receive the message. This justifies our choice of making adversaries corrupt as many parties as possible.

We define the maximum latency as the highest number of hops (among successful runs only) that a message took to be propagated; if none of the 10 000 runs was successful, we do not plot the maximum latency. The latency unit in our plots is δ_{Channel}. Note we do not require messages to be delivered within a fixed time-bound. As we have observed from our simulations, the maximum latency for any configuration which succeeds reasonably often is within $9 \cdot \delta_{\text{Channel}}$, and hence we consider this practical (details of the maximum latency can be found in Figures 4.2 to 4.4).

To ensure our protocol performs well *independently* of the sender's weight, we take the worst result among the sender choices (for each weight distribution).

Methodology details. For the exponential and few heavy weight distributions, where there can be senders with different weights, we make 10 000 runs for each case. More concretely, for the exponential weight distribution, we make 10 000 runs for the lightest sender, 10 000 runs for the median weighted sender, and 10 000 runs for the heaviest sender, whereas for the few heavy weight distribution, we make 10 000 runs for the lightest sender, and 10 000 runs for the heaviest sender. The average success rate shown in the plots is the least (10 000 run) average success rate among the different senders. For latency, we take the maximum among all runs for all possible senders.

For a fixed set of parameters, in each of the 10 000 runs, a new corruption set is chosen (independent of the ones picked in other runs). This means that for the random corruption strategy, a fresh set of corrupted parties is selected for each run. On the other hand, since the light first and heavy first corruption strategies are deterministic, the set of corrupted parties is always the same for each run. The sender, which is determined by the set of parameters, cannot be corrupted.

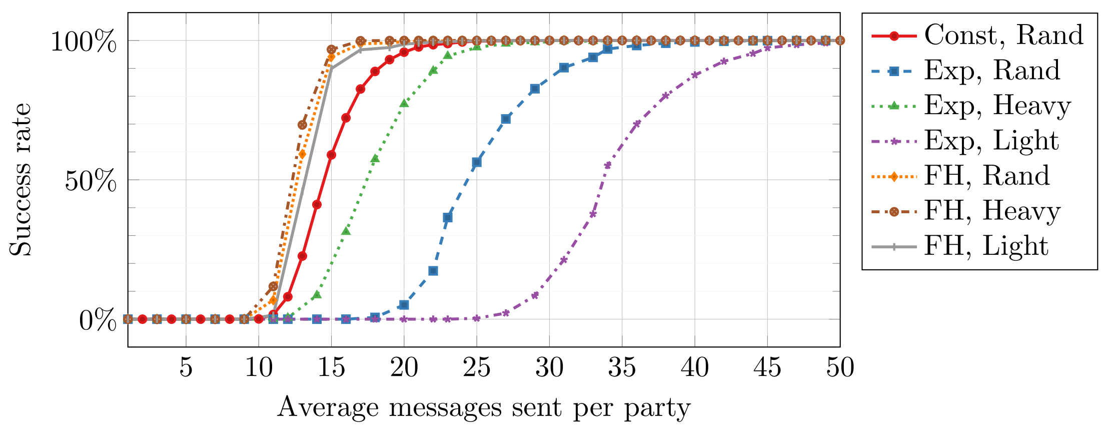
4.6.3 Simulations and Results

Comparison against weight-oblivious protocols. To compare the performance of WFF and a weight oblivious protocol, we measured the success rate for $\mathsf{WFF}(k)$ and a weight oblivious protocol $\mathsf{WOF} := \Pi_{\mathrm{Flood}}(\mathsf{WFS}(\mathtt{E}, k))$ with $\mathtt{E}(p) := 1$ for different exponential weight distribution (with changing ratios between the heaviest and lightest party).[10] The results can be found in Figure 4.1. The plot shows that our protocol (WFF) achieves a 100% success rate at a much lower number of transmitted messages than the weight-oblivious one (WOF). The only exception is when the weight ratio between the heaviest and the lightest parties is 1, the exponential weight distribution is the same as the constant weight distribution, and hence the protocols become identical. Note that while the WFF protocol achieves practical security with low message complexity regardless of the ratio between the heaviest and lightest party, the message complexity of WOF to achieve a 100% success rate increases drastically as the ratio increases.

Performance for changing weight distributions. In Section 4.3.2.1, we bounded the message complexity of WFF by $\frac{2 \cdot n \cdot (\log(n) + \kappa)}{\gamma}$ (see Theorem 4.3.3), and in Section 4.4.2 we showed that this number of messages is inherent for the constant weight distribution (see Lemma 4.4.3), implying that WFF is optimal up to a constant factor for this distribution. Although the obtained upper bound is independent of the weight, since it is tight only for the constant weight distribution, it could be that WFF performs poorly for other distributions. To show this is not the case, we measured the success rate (and maximum latency) for sending a single message in $\mathsf{WFF}(k)$ as a function of the message complexity (induced by adjusting k) for different weight distributions and corruption strategies. See Figure 4.2.

Unsurprisingly, the adversarial strategy inducing the highest cost corrupts as many light nodes as possible. This fits the intuition from Section 4.3.3: By corrupting as many light nodes as possible, an adversary can get a slight advantage in terms of the number of emulated nodes they control because the ceiling embedded in the emulation function has a proportionally larger effect on such nodes. Furthermore, note that for the constant weight distribution $\mathsf{WFF}(k)$, selects k neighbors uniformly at random and at least $\lceil \gamma \cdot n \rceil$ of the parties remains honest. Hence, this corresponds to the performance that can be expected by additionally assuming that a certain fraction of the parties remains honest and use flooding protocols tailored to this setting. We emphasize that our protocol only induces marginally larger (within a small constant factor) message complexity for all the considered weight distributions and corruption strategies. This aligns with Section 4.3.3, where our bound on the message complexity for the weighted setting was only worse by a factor of 4 compared to the bound that relied on a constant fraction of honest parties. Therefore, security for our protocol in the weighted setting comes comparatively at a much lower cost.

[10]The protocol $\mathsf{WOF} := \Pi_{\mathrm{Flood}}(\mathsf{WFS}(\mathtt{E}, k))$ for $\mathtt{E}(p) := 1$ corresponds to the protocol where each party selects k parties uniformly at random as their neighbors without taking weight into account.



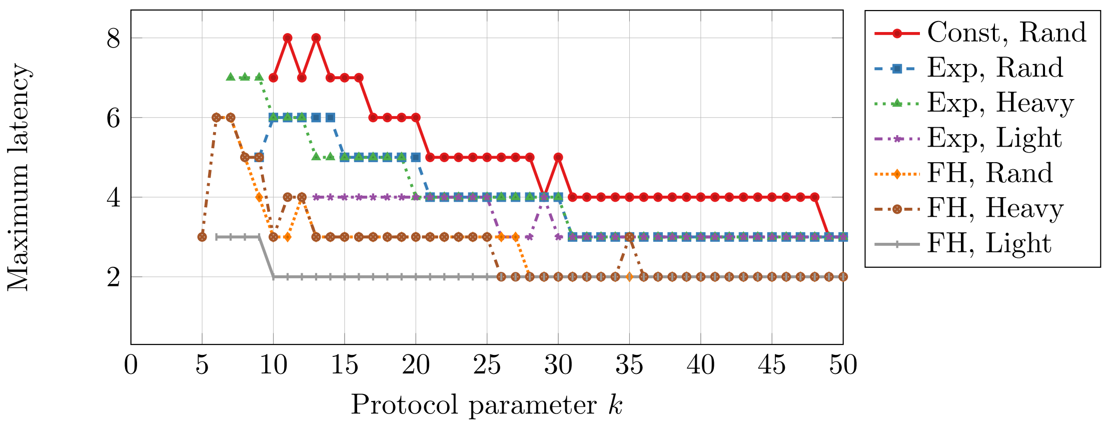
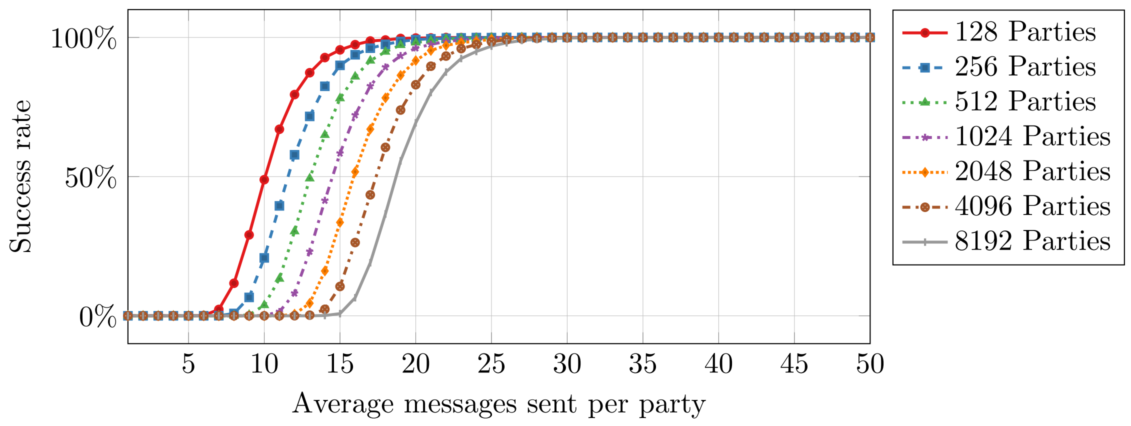


Figure 4.2: Success rate and maximum latency of WFF protocol for different weight distributions and corruption strategies, depending on the average number of messages sent per party, for $n = 1024$ parties, a 50% corruption threshold, a ratio of 10^6 between heaviest and lightest parties and $c = 10$ heavy parties for FH.

Note that for all successful runs, the latency is at most $8 \cdot \delta_{\text{Channel}}$. Furthermore, our protocol actually induces a lower latency when deployed with unevenly distributed weights. This is because connectivity is concentrated around the heavy parties, which have larger neighborhoods.

Scalability. A key feature of flooding protocols is their scalability. To benchmark the scalability of our proposed protocol WFF, we measured, for different numbers of parties, the success rate of the protocol depending on the average number of messages each party sends (again induced by varying k). For simplicity, we chose only to include the constant weight distribution (the performance for varying weight distributions is plotted in Figure 4.2). The results can be found in Figure 4.3. As one can observe, both the average message complexity per party and the maximum latency only increase logarithmically with the number of parties, confirming our theoretical expectations (see Section 4.3.4).

By the time of writing, there are around 12k running Bitcoin nodes [22a] and roughly 8k nodes in the Ethereum network [22b]. Extrapolating from Figures 4.2



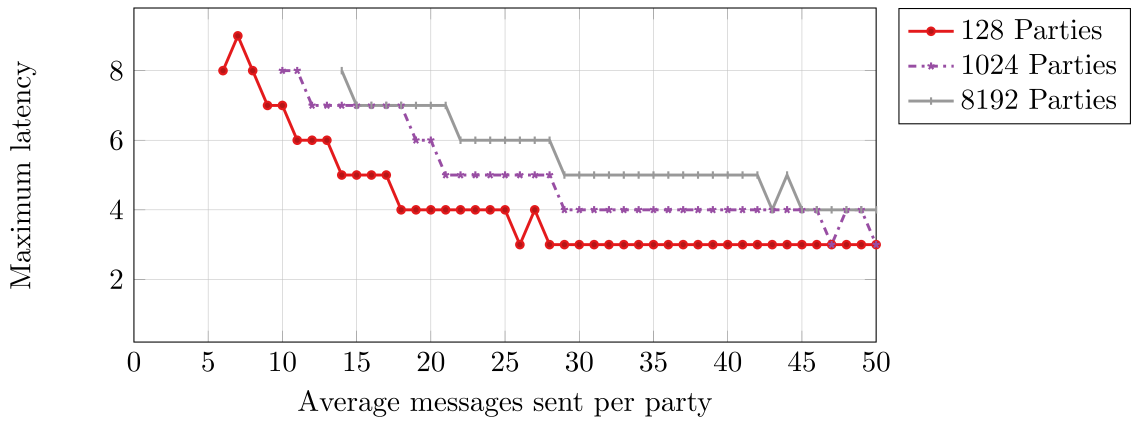
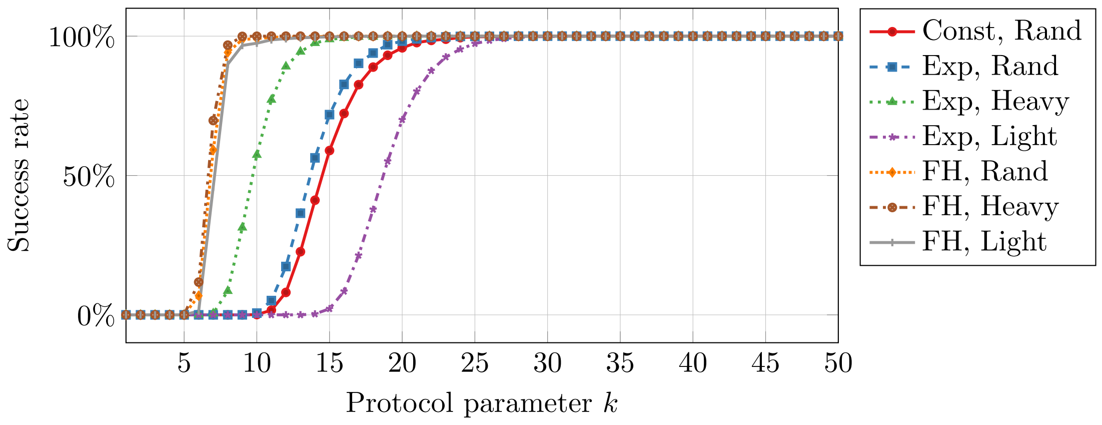


Figure 4.3: Scalability of WFF protocol. We consider the constant weight distribution, and the random corruption strategy, with a 50% corruption threshold.

and 4.3, it seems that independently of the stake distribution, WFF can realize a secure flooding network with an average number of connections per message of just ∼55 for such number of nodes. We conclude that this is within the realm of the number of connections existing widely used implementations maintain by default. Note, however, that the workload is not distributed evenly among nodes in WFF, as heavier nodes need to maintain more connections. In Section 4.4, we showed that this is inherent for this type of protocol in the weighted setting.

Estimating protocol parameters for practical security. As already mentioned, the number of messages a party sends is given by its emulation function $\mathtt{E}(p) := \lceil \alpha_p \cdot n \rceil$ multiplied by the protocol parameter k. We now analyze what values one can set k to in order to achieve security in practice. Figure 4.4 shows, for the different weight distributions and corruption strategies how the success rate and the maximum latency vary depending on the protocol parameter k.

It is worth mentioning that although, at first sight, our protocols may seem to perform better for the exponential and few heavy weight distributions, this is actually not the case: while it is true that the protocol achieves a higher success rate and a lower diameter for smaller values of k, for both these weight distributions (but not for the constant weight distribution) the average number of messages sent per party grows by a factor larger than 1 (see Figure 4.5).



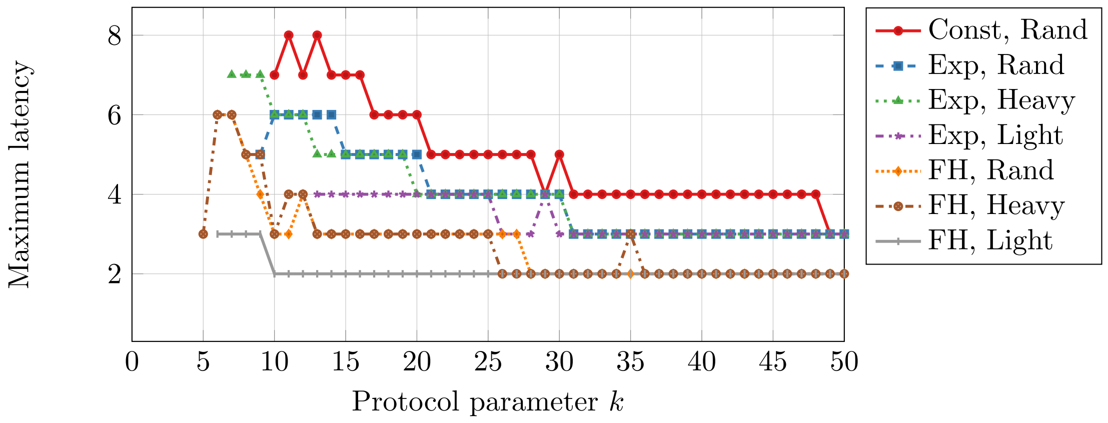


Figure 4.4: Success rate and maximum latency of WFF protocol for different weight distributions and corruption strategies, depending on the protocol parameter k, for $n = 1024$ parties, a 50% corruption threshold, a ratio of 10^6 between richest and poorest parties and $c = 10$ heavy parties for FH.

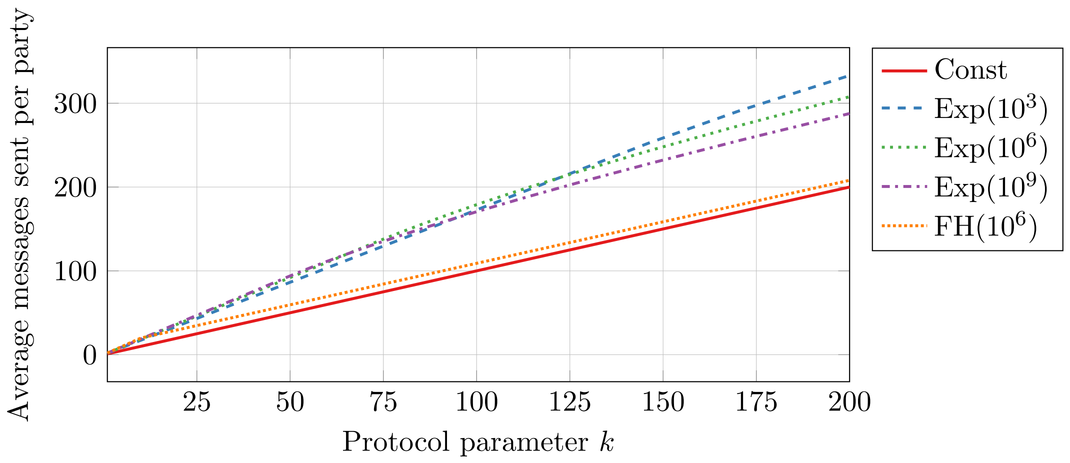


Figure 4.5: Average number of messages sent per party for WFF protocol, depending on the protocol parameter k, for different weight distributions. In the plot, the exponential and few heavy weight distributions are parameterized by the weight ratio between the lightest and heaviest party. We assume no corruptions.

5 Asymptotically Optimal Message Dissemination with Applications to Blockchains

5.1 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 [Woo+14], such multicast networks are efficiently implemented via a flooding protocol [KMG03; Liu+22b; MNT22b], 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. It was shown in [KMG03] that one needs an expected neighborhood size of $\Omega(\log(n) + \kappa)$, where n is the number of parties and κ is a security parameter, for the message to reach all parties with overwhelming probability in κ. As a consequence, current flooding protocols incur $\Omega(l \cdot (\log(n) + \kappa))$ bits of per-party communication (in total $\Omega(l \cdot n \cdot (\log(n) + \kappa))$ bits), for an l-bit message. A trivial lower bound on the total communication complexity is $\Omega(l \cdot n)$ since all n parties need to receive the message. This, in turn, implies that (the maximal) per-party communication must be $\Omega(l)$ bits. This leaves a gap between the lower bounds and what current flooding protocols achieve. For practical blockchain systems where messages contain large blocks (e.g., around 1MB), the incurred communication constitutes one of the main bottlenecks. We, therefore, ask the following question:

> *Is there a flooding protocol that incurs the optimal total communication of $O(l \cdot n)$ bits, where each party communicates $O(l)$ bits?*

We answer this question in the affirmative by providing two highly robust flooding protocols for n parties with a success rate overwhelming in the security parameter κ. 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 accumulating a constant fraction of the total resource. More details follow below.

5.1.1 Contributions

Warm up: Optimal flooding with a linear neighborhood and constant diameter. We first 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. 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 will forward it to *all* other parties. Once a party receives sufficiently many shares, they will be able to reconstruct the original message.

Theorem 5.1.1 (ECCast (informal))**.** *For n parties,* ECCast *ensures asymptotically optimal flooding with a diameter of* 2 *and an overwhelming success probability in κ, for message of length at least $\Omega(n \cdot (\log(n) + \kappa))$ and at least a constant fraction of the parties remaining honest.*

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 [Yin+19].

Optimal flooding with a logarithmic neighborhood and diameter. We then present the protocol ECFlood,[2] which requires each party to connect to only $O(\log(n) + \kappa)$ other parties and use only $O(l)$ 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 each party with an independent probability ρ. When a party receives such a share, they will then again forward it to all other parties with the same independent probability ρ. Once a party receives sufficiently many shares, they will be able to reconstruct the original message.

Theorem 5.1.2 (ECFlood (informal))**.** *For n parties and a security parameter κ, there are $\mu = O(\log(n)+\kappa)$ and $\rho = O(n^{-1})$ such that* ECFlood *ensures asymptotically optimal flooding with a logarithmic diameter and an overwhelming success probability in κ, for messages of length at least $\Omega((\log(n) + \kappa) \cdot (\log(\log(n)) + \kappa))$ and at least a constant fraction of the parties remaining honest.*

In particular, it is worth noting that ECFlood shaves a factor of $\log(n) + \kappa$ off both the communication complexity and 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 that ECFlood requires no trusted setup but merely relies on a weak cryptographic accumulation scheme, which can be realized efficiently from standard cryptographic assumptions.

[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 5.1: Comparison of flooding for messages of length l among n parties where a constant fraction of parties is honest.

Property	Naive	[Liu+22b; MNT22b]	ECFlood (this work)	ECCast (this work)
Min. message length	1	1	$\Omega((\log n + \kappa)(\log\log n + \kappa))$	$\Omega(n \cdot (\log(n) + \kappa))$
Max. neighbors	$n-1$	$O(\log(n) + \kappa)$	$O(\log(n) + \kappa)$	$n-1$
Max. per-party comm.	$l \cdot (n-1)$	$O(l \cdot (\log(n) + \kappa))$	$O(l)$	$O(l)$
Total comm.	$l \cdot (n-1)$	$O(l \cdot n \cdot (\log(n) + \kappa))$	$O(l \cdot n)$	$O(l \cdot n)$
Diameter	1	$O(\log(n))$	$O(\log(n))$	2

We summarize the properties of ECFlood and ECCast and compare them to other flooding protocols with similar robustness in Table 5.1, 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 (and achieve optimal communication for sufficiently long messages).

Probabilistic simulations for ECFlood. While the theoretical analysis of ECFlood shows that our protocol is asymptotically optimal, we use probabilistic simulations to evaluate its practical efficiency. The main results of our simulations are shown in Figures 5.1 and 5.2. The parameter d corresponds to the expected number of parties each share is sent to by every party, i.e., $\rho = d/n$. Figure 5.1 shows that increasing d also increases the redundancy (i.e., the per-party communication complexity divided by the message length l) and thus the communication complexity. On the other hand, Figure 5.2 shows that increasing d decreases the latency. Since there are μ shares to be sent to d parties, the total number of neighbors per party is $\mu \cdot d$. Figure 5.1 further shows that increasing the number of shares μ decreases the redundancy.

With these tradeoffs in mind, we can compare our results to the state-of-the-art provably secure flooding protocol [Liu+22b]. The simulations in [Liu+22b, Figure 4.3]

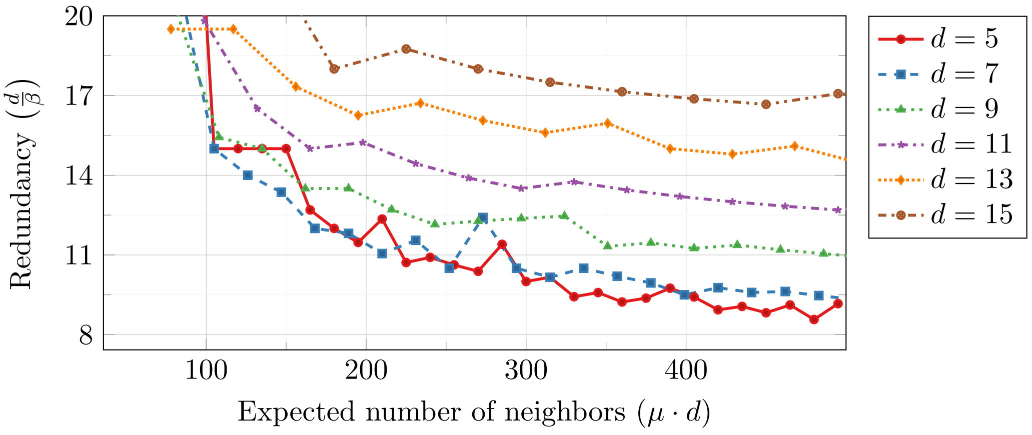


Figure 5.1: Results for simulations of ECFlood($\frac{d}{n}$) for different values of d, a fixed number of parties $n = 8192$, and a variable number of shares μ. The graphs show the redundancy of the protocol as a function of the expected number neighbors. The number of shares is incremented in steps of 3.

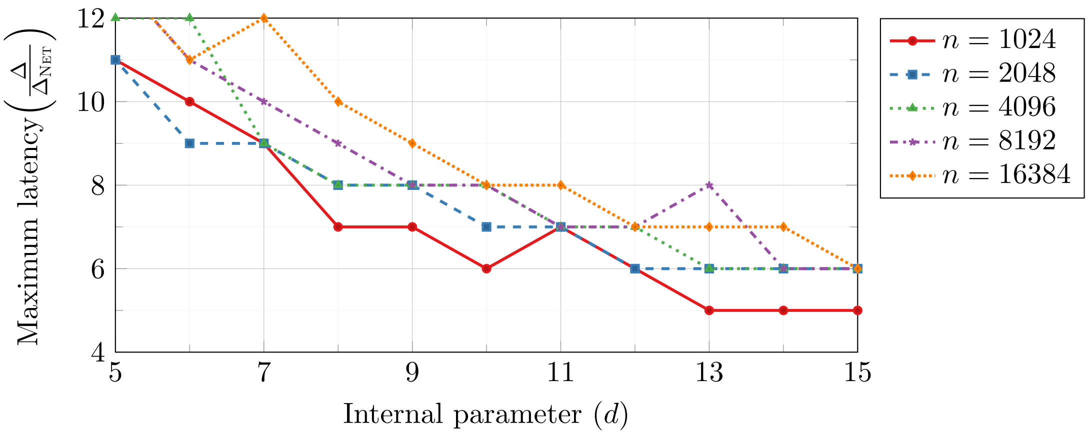


Figure 5.2: Results for simulations of $\mathsf{ECFlood}\left(\frac{d}{n}\right)$ for different values of d and n, but with a fixed number of shares $\mu = 30$. The graphs show the maximal latency before any party has received enough shares to be able to reconstruct for different values of d.

show that to succeed 99% of the time for 8192 parties, one has to, in their protocol, let each party forward the entire message to around 28 neighbors, resulting in a latency of 6 network delays with redundancy 28. ECFlood for $d = 15$ achieves the same latency with a redundancy below 20 for 200 neighbors per party. If a slightly larger diameter is acceptable, ECFlood with $d = 9$ has a worst-case latency of 8 network delays and redundancy of less than 14 for 200 neighbors per party. This means that even though parties have to connect to more neighbors in ECFlood, the total amount of data sent is less than half compared to the state-of-the-art for these parameters. We note that the number of connections is also a limiting factor, but 200 is still practical; e.g., Bitcoin clients, by default, have up to 125 connections [Ger+15].

Flooding in the weighted setting. We leverage the idea of emulation from [Liu+22b] to make a general transformation from a flooding protocol that is secure, assuming a constant fraction of the parties behaves honestly, to a secure flooding protocol, assuming that a constant 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 $\widetilde{\gamma} \cdot 2^{-1}$. Hence, if we assume a constant fraction of honest weight, there will also be a constant fraction of honest emulated parties. This implies that we can translate our protocols ECFlood and ECCast (as they will work for any constant fraction of parties), using Flood2WeightedFlood, also to work assuming only a constant fraction of honest weight. 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 [Liu+22b, Corollary 4.4.2], it was shown that it is inherent for the weighted setting that parties with a large fraction of weight must send more messages.

Outline of the chapter. In this section, we proceed with a technical overview, discuss our model, and review related work. In Section 5.2, we define our model and the primitives our protocols build upon. Next, in Section 5.3, as a warm-up, we present the ECCast protocol and prove it secure. In Section 5.4, we present the ECFlood protocol and prove it secure. In Section 5.5, we present our protocol for transforming any flooding protocol to one that works in the weighted setting. Finally, in Section 5.6, we estimate practical parameters for our protocol using probabilistic simulations.

5.1.2 Technical Overview

As discussed above, prior works that let each party forward a message to a random subset of neighbors (see, e.g., [KMG03; Liu+22b; MNT22b]) need each party to connect to $\Omega(\log(n) + \kappa)$ 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 no party is unlucky is negligible, the additional $\log(n)$ neighbors are needed. It is, therefore, impossible to decrease the number of needed connections of such flooding protocols.

To further improve the communication complexity, we focus instead on reducing the number of bits sent to each of these neighbors. For that, we deviate from the above approach and design our flooding protocol in two steps. First, we consider a *weak* flooding protocol that ensures that with a constant probability, a constant fraction of the parties receives the message. Secondly, we introduce a compiler that lifts a weak flooding protocol to a complete flooding protocol that guarantees delivery to all parties with overwhelming probability.

Flooding amplification. The protocol compiler WeakFlood2Flood splits a message into a number of shares μ using erasure-correcting codes and makes use of a weak flooding protocol to distribute each of these shares. Since the shares are created using erasure-correcting codes, each party doesn't need to receive all shares to reconstruct the original message.

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. A such accumulator can be implemented efficiently, e.g., using Merkle trees or signature schemes.

More concretely, let the reconstruction threshold be $\tau = \xi \cdot \mu$, for some constant ξ. Using standard erasure-correcting codes (e.g., Reed-Solomon codes), this can be obtained with a share size of $O(l \cdot \tau^{-1})$. To achieve a flooding protocol with optimal communication, we need to ensure that 1) each party receives τ shares and 2) each instance of weak flooding only incurs constant overhead with respect to the size of each share (i.e., $O(l \cdot \tau^{-1})$ bits) of per-party communication.

Using the Chernoff bound, one can show that if there is a constant independent probability for each party to receive each sent share, then all parties receive a constant fraction of the shares with overwhelming probability when at least $\log(n) + \kappa$ shares are sent.

Hence, the task of finding an asymptotically optimal flooding protocol is reduced to finding a weak flooding protocol that ensures delivery with a constant independent probability for each input message and sends each input message to only a constant number of parties.

A weak flooding protocol. Our candidate for a weak flooding protocol is the protocol $\mathsf{ERFlood}(\rho)$ that lets each party forward each message to each party with an independent probability ρ.

Previous works [KMG03] showed that the probability that there is an isolated party for $n \to \infty$ when $\rho = \frac{\log(n)+c+o(1)}{n}$ for some constant c is given by $1 - e^{-e^c}$. This means that one needs to set $\rho = \Omega(\log(n) \cdot n^{-1})$ to have a constant success probability for all parties to receive the message. Consequently, the expected size of each neighborhood would be $\Omega(\log(n))$, which is too much communication.

To overcome this, we observe that we only need that the probability that any fixed party receives the message is constant. Using a novel analysis of the protocol, we prove that by sending the message to only a constant number of neighbors, there is a constant probability that the message reaches a constant fraction of all parties.

5.1.3 Model

Our results are proven for a static set of parties connected by unauthenticated point-to-point channels that immediately leak any message sent to the adversary. Our analysis additionally uses an upper bound on the time it takes to send a message through each channel, but this is not required to be known before our protocols are deployed.

On composable security. We prove that our protocols implement a property-based definition of flooding. It is, therefore, not automatically guaranteed that our results compose with other constructions as it would have been if they were proven within a composable security framework such as UC [Can20]. However, note that none of our protocols has any secrecy, i.e., all inputs given to any honest party are immediately leaked to the adversary. As used in [MNT22b, proof of Lemma 3.6.1], it is therefore straightforward to simulate such protocols because the simulator can run the original flooding protocol with full knowledge of all inputs. Based upon this, we believe it is straightforward to show that our flooding protocols UC-realize a flooding functionality as the one provided in Chapter 3.

On security against adaptive adversaries. Our results are proven for all adversaries that can statically corrupt all but a constant fraction of the parties (except for the protocol $\mathsf{Flood2WeightedFlood}$, where we naturally require a constant fraction of the weight remains honest). However, even stronger results hold. The protocol ECCast is secure against an adaptive adversary if we assume that the adversary cannot retract messages that are already sent (often referred to as the *atomic message send model*). Unfortunately, as observed by Matt et al. [MNT22b], this is not sufficient for any protocol with a sublinear neighborhood because an adversary can then simply corrupt the entire neighborhood of the sender and thereby prevent delivery of the message. This rules out that ECFlood can be proven secure using only this assumption.

To circumvent this apparent impossibility of security against adaptive adversaries for flooding protocols, Matt et al. [MNT22b] introduced the model of δ-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. All of our proofs for ECFlood go through if it is assumed that an adversary is delayed for the entire delivery time of the protocol, as then the set of parties controlled by an adversary will be independent of the neighborhoods chosen for the delivery of a particular message. The setting will, thereby, essentially be static for each specific message.

We note that it may not be practically feasible to require each peer to resample their neighbors for each message, as suggested by our protocol ECFlood. If one is willing to assume an adversary delayed for an extended period of time, then our protocol can be run securely without resampling neighbors for each message but instead only resampling neighbors at specific time intervals of length at least the corruption delay of the adversary.

On the weighted model. Our protocol Flood2WeightedFlood compiles any protocol secure assuming a constant fraction of parties to one that is secure assuming only a constant fraction of publicly assigned weights remains under honest control. This assumption can be realized for *Proof-of-Stake* based protocols [CM19; DPS19; Dav+18] where the protocol relies on a constant fraction of the stake behaving honestly. As the stake is publicly available in the maintained ledger of such protocols, it is immediate that it can be used directly as public weights.

As noted by [Liu+22b], committee selection techniques [PS17b; PS18b] can be used to instantiate weights from other resource assumptions such as computational power.

5.1.4 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 [Woo+14], etc.) are typically implemented via a forwarding mechanism, where for a party to distribute a message, the party 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 $\rho > \frac{\log(n)+\kappa}{\gamma\cdot n}$ to ensure that messages are delivered to all honest parties with overwhelming probability in κ.

Matt, Nielsen, and Thomsen [MNT22b] formally proved the 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 follow-up work [Liu+22b], 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 [KMG03; Liu+22b; MNT22b] incur a total communication of $O(l \cdot n \cdot (\log(n) + \kappa))$ bits, for a message of size l. In contrast, our protocols incur the (asymptotically) optimal total communication of $O(l \cdot n)$.

Coretti, Kiayias, Moore, and Russell [Cor+22] considered the problem of designing a message diffusion mechanism based on the majority of honest stake assumption explicitly tailored for the Ouroboros Praos consensus protocol [Dav+18]. However, their flooding protocol achieves a weaker guarantee because 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 the efficiency of flooding protocols tailored for blockchains by applying structured approaches and heuristics [RT19; SOA16; 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]. Many subsequent works achieve communication complexity $O(l \cdot n + \text{poly}(n, \kappa))$ (see, e.g., [Bha+22; FH06; GP16; Nay+20]). We note that techniques similar to those we use for our ECCast protocol were used within agreement protocols in [Nay+20].

In all these works, parties communicate to all other parties (so the neighborhood size is $n - 1$). In contrast, we provide ECFlood where each party communicates to only $O(\log(n) + \kappa)$ neighbors.

5.2 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 fundamental bounds that we will use for our proofs.

5.2.1 Parties, Adversary and Communication Network

We consider a 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.[3] We will use h to denote a bound on the size of $\mathcal{H}$. For the remainder of the chapter, 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 and assume this to be a constant.

[3] One can extend our protocols to handle so-called delayed adaptive adversaries, using techniques presented in [MNT22b].

We assume that all parties are connected pairwise through unauthenticated point-to-point channels and let Δ_{NET} denote an upper bound on the delivery time for the underlying point-to-point channels.

5.2.2 Primitives

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

Definition 5.2.1 (Erasure Correcting Code Scheme)**.** Let $\mu \in \mathbb{N}$ be the number of shares, and let $e \in \mathbb{N}$ be the number of erasures to be tolerated. A pair of algorithms ζ is a (μ, e)-erasure-correcting-code-scheme (abbreviated (μ, e)-ECCS) if it consists of two algorithms:

- ζ.Enc: An encoding algorithm that takes a message $m \in \{0,1\}^*$ and produces a sequence of shares $s_1, \ldots, s_\mu$.
- ζ.Dec: A decoding algorithm that if a sequence of shares $s'_1, \ldots, s'_\mu$ s.t. it holds for at least $\mu - e$ of them that $s'_i = s_i$ and for the remaining $s'_i = \perp$ 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, e)-ECCS in a straightforward manner. That is, for a message m with length l (where $l \geq \mu - e$), let each share be an element in a Galois Field of order 2^a for $a = \lceil \max(\log(\mu - 1), \log(\frac{l}{\mu - e})) \rceil$. If ζ is such a scheme, we will have for a message where $a = \lceil \log(\frac{l}{\mu-e}) \rceil$ that

$$\zeta.\texttt{ShareSize}(l) = O\Big(\frac{l}{\mu - e}\Big). \tag{5.1}$$

This implies that the total bitlength of the shares will be $O(l \cdot \frac{\mu}{\mu - e})$. We note that it has been shown that the encoding and decoding of such codes be done in $O(2^a \cdot a^2)$ time [Did09].

Weak cryptographic accumulators. We will, in this work, 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). We define this below.

Definition 5.2.2 (Weak Static Cryptographic Accumulation Scheme)**.** A pair of algorithms α is a *weak static cryptographic accumulation scheme* (abbreviated WSCAS) if it consists of two algorithms:

- α.Accumulate($\{m_1, \ldots, m_\eta\}$) : An algorithm for accumulating a set of input values $\{m_1, \ldots, m_\eta\}$. It returns an accumulated value z and a sequence of proofs

$\pi_1, \ldots, \pi_\eta$ 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 polynomial-time adversary can find a set of values $M := \{m_1, \ldots, m_\eta\}$, a value $m' \notin M$, and a proof π such that the accumulator verifies $\alpha.\mathsf{Verify}(m', \pi, z) = \top$ for $z \leftarrow \alpha.\mathsf{Accumulate}(M)$.

See [BP97] for the original formal definition of collision-freeness, and [Özç+21] for an overview of accumulator constructions.

We use the notation $\alpha.\texttt{AccSize}$ for a bound on the size of the accumulated value and $\alpha.\texttt{ProofSize}(\eta)$ 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 this yields

$$\alpha.\texttt{AccSize} = O(\kappa) \text{ and } \alpha.\texttt{ProofSize}(\eta) = O(\kappa). \tag{5.2}$$

The same complexity can be achieved by basing α on RSA accumulators [BM93] 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 $\alpha.\texttt{ProofSize}(\eta) = O(\log(\eta) \cdot \kappa)$.

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 [Liu+22b, Section 4.3.1] with only minor differences. Informally, we want each message input delivered to all parties with a probability that is overwhelming in the security parameter.

Definition 5.2.3 (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 τ, then by time $\tau + \Delta$ there is a probability overwhelming in the security parameter κ that all other honest parties output m.

In contrast to previous definitions, we do not require that flooding protocols guarantee message relay (it is allowed that a message sent by the adversary is only received by a subset of honest parties), as this definition suffices for most blockchain protocols [CM19; Dav+18; GKL15; PS17a; PS18b; Yin+19], since they do not depend on relaying of received messages at the flooding layer. To achieve message relay, one can let the honest parties re-distribute the received messages.

5.2.3 Additional Notation

We use the notation $\Gamma^{\lambda}_{p_s}(G)$ for the set of neighbors of a party (usually the sender) p_s at a distance at most λ in a graph G. When clear from the context, we omit both p_s and G for this set and merely write Γ^{λ}. 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.

5.2.4 Bounds

Lemma 5.2.4 (Convergence of Geometric Sum)**.** *Let $c, a \in \mathbb{R}$ and $|a| < 1$ then*

$$\sum_{n=0}^{\infty} c \cdot a^n = \frac{c}{1-a}.$$

Lemma 5.2.5. *Let $c, a \in \mathbb{R}$ s.t. $|a| < 1$ and $c \geq 2$ then*

$$\sum_{n=1}^{\infty} a^{(c^n)} \leq \frac{a}{1-a}. \tag{5.3}$$

Proof. First, note that

$$\sum_{n=1}^{\infty} a^{(c^n)} \leq \sum_{n=0}^{\infty} a^{(c^{n+1})} = \sum_{n=0}^{\infty} a^{c \cdot (c^n)}. \tag{5.4}$$

Now, note that for all $n \in \mathbb{N}$ we have that

$$1 + c^n \leq c \cdot c^n \implies a^{c \cdot c^n} \leq a^{1+c^n}. \tag{5.5}$$

Hence, by combining Equations (5.4) and (5.5) we get that

$$\sum_{n=1}^{\infty} a^{(c^n)} \leq \sum_{n=0}^{\infty} a \cdot a^{(c^n)}. \tag{5.6}$$

Equation (5.3) follows by the convergence of the geometric sum (Lemma 5.2.4) and the fact that for all $n \in \mathbb{N}$ we have that $a^{(c^n)} \leq a^n$. □

5.3 Optimal Flooding With a Constant Diameter and Linear Neighborhood

In this section, we warm up by presenting our protocol ECCast and show that it is a flooding protocol with a maximum per-party communication of $O(l)$, a total communication complexity of $O(l \cdot n)$, 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 the original sender did not send. To prevent this, we use the cryptographic accumulator.

Protocol ECCast(ζ, α)

The protocol is parameterized by, a (n, e)-ECCS ζ for some $e \in \mathbb{N}$, 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 $\texttt{Received}_i \coloneqq \varnothing$.

Send: When p_i receives (Send, m) they share the message m into shares $\zeta.\mathsf{Enc}(m) = s_1, \dots, s_n$. Furthermore, they obtain an accumulated value and proofs for each share and its share number $z, \pi_1, \dots, \pi_n = \alpha.\mathsf{Accumulate}(\{(s_j, j) \mid 1 \leq j \leq n\})$. 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 $|\texttt{ReceivedShares}_i[z]| \geq n - e$, then they
  1. Obtain a sequence of shares $s_1, \dots, s_n$ by letting $s_j = s$ if $(s, j) \in \texttt{ReceivedShares}_i[z]$ and otherwise sets $s_j = \perp$ (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, $\texttt{Received}_i \coloneqq \texttt{Received}_i \cup \{\zeta.\mathsf{Dec}(s_1, \dots, s_n)\}$.
- 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.

We now prove the following theorem.

Theorem 5.3.1. *Let $e \geq n \cdot (1 - \gamma)$, let ζ be a (n, e)-ECCS, and let α be a WSCAS, then the protocol* ECCast(ζ, α) *is a strong* $(2 \cdot \Delta_{\text{NET}})$*-flooding protocol.*

Proof. Let m be a message input to some honest party s at time τ, and let the accumulated value sent out be z. The delivery guarantees for the underlying point-to-point channels ensures that at latest at time $\tau + \Delta_{\text{NET}}$ 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 $\tau + 2 \cdot \Delta_{\text{NET}}$. Hence, if p_i is an honest party, then the size of $\texttt{ReceivedShares}_i[z]$ will be at least $n \cdot \gamma = n - n \cdot (1 - \gamma) \geq n - e$. Therefore any honest party p_i will be able to reconstruct the original message at the latest at time $\tau + 2 \cdot \Delta_{\text{NET}}$ 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. However, this does not happen unless the adversary can break the collision-freeness of the WSCAS. $\square$

Communication complexity of ECCast. Let us now analyze the complexity of $\mathsf{ECCast}(\zeta, \alpha)$ (for ζ and α instantiated as suggested by Theorem 5.3.1) 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

$$n \cdot (1 + \zeta.\mathtt{ShareSize} + \log(n) + \alpha.\mathtt{ProofSize} + \alpha.\mathtt{AccSize}). \tag{5.7}$$

If we instantiate the ECCS with Reed-Solomon codes, we get that the size of each share is bounded by $O(l \cdot (\gamma \cdot n)^{-1})$. For a constant fraction of honest parties, we, therefore, have that $\zeta.\mathtt{ShareSize} = O(l \cdot n^{-1})$. And by using an efficient WSCAS α with the size of accumulated value and proof $O(\kappa)$ (see Equation (5.2)), we get that the communication of each party is bounded by

$$O(l + n \cdot (\log(n) + \kappa)). \tag{5.8}$$

As all parties use this communication, the total communication complexity is bounded by

$$O(n \cdot (l + n \cdot (\log(n) + \kappa))). \tag{5.9}$$

This communication is optimal when $l = \Omega(n \cdot (\log(n) + \kappa))$. We remark that the constant multiplied to l is only γ^{-1} times the constant of the Reed-Solomon codes.

5.4 Optimal Flooding With Logarithmic Neighborhood and Diameter

In this section, we show how to obtain a flooding protocol with (asymptotically) optimal communication complexity. We achieve this 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.

5.4.1 Weak Flooding

Informally, 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. For the sake of simplicity, we first give an informal definition of a weak flooding protocol below.

Definition 5.4.1 (Weak Flooding (informal)). 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 τ when a message m is input to some honest party, then it must be that for any $p_i \in \mathcal{H}$

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

The reason that this is only an informal definition is that the definition fails to encapsulate what happens if multiple messages are sent in a protocol. For example, if a sequence of messages is sent simultaneously, then a protocol satisfying the definition above could let all messages be delivered with probability ξ or deliver none with probability $(1-\xi)$. Such a definition is, therefore, insufficient if one wishes to amplify the delivery guarantees for multiple messages. Therefore, to get a definition where it is reasonable to boost the delivery probability, it is necessary for us to describe how the delivery of several messages relates.

To formally define this property, we first introduce random variables for each delivery event to be able to precisely describe what kind of dependency is allowed between delivery events.

Definition 5.4.2 (Timely delivery)**.** We say that a message m input at time τ is Δ-*timely-delivered* for a party p_i if p_i has output m at the latest at time $\tau+\Delta$. We let $\texttt{Timely}_{m,i}(\Delta)$ denote the induced indicator variable equal to 1 if the predicate holds and 0 otherwise.

Using this, we next define a weak flooding algorithm.

Definition 5.4.3 (Weak Flooding)**.** Let Π be a protocol executed by parties $\mathcal{P}$, where each party $p \in \mathcal{P}$ can input a *set* of messages at any time, and as a consequence, parties may get a set of messages as output. We say that Π is a *weak* (Δ, ξ)-flooding protocol if for any such set of messages $\{m_1, m_2, \dots, m_\eta\}$ input to some honest party in the execution of Π where each message has never been sent over any channel before, then for any party $p_i \in \mathcal{H}$ there exist independent binary random variables $X_1, \dots, X_\eta$ where for all $z \in \{1, 2, \dots, \eta\}$, $\Pr[X_z = 1] \geq \xi$, and

$$\sum_{j=1}^{\eta} X_j \preceq \sum_{j=1}^{\eta} \texttt{Timely}_{m_j,i}(\Delta).$$

Remark 5.4.1. Directly requiring the $\texttt{Timely}$ random variables to be independent would be a too strong requirement: An adversary can always ensure the delivery of a message if some other message is delivered by simply "injecting" the message on all point-to-point channels, thereby making the delivery events dependent. Note, however, that this influence can only be exerted in a way that increases the probability for delivery events to appear, which is what is captured in the definition above.

Protocol description. We now show that the protocol ERFlood[4] from [MNT22b, Section 3.6.2], actually is a weak flooding protocol by providing a new analysis for the protocol. Concretely, we will make a different analysis and show that just a constant expected degree ERFlood ensures that the probability that a party receives a message that is input for the first time is also constant.

We first recap the protocol, which we phrase using the abstraction of *neighborhood selection algorithm* as in [Liu+22b, Section 4.3.2] . This abstraction allows our intermediate results to be used for different algorithms. However, unlike in their work, we

[4]The name ERFlood was given due to the relation to Erdős–Rényi-graphs.

consider only one neighborhood selection algorithm for the entire protocol instead of letting each party have a unique one.

When talking about a set of messages input to a party (as required by the definition of a weak flooding protocol), we will interpret this as that the party which receives the set of messages will send each of them immediately after the other using the send command. The particular neighborhood selection algorithm we will concentrate on in this section is the one where each party selects all other parties as their neighborhood with an independent probability ρ. We denote this algorithm with ERS, which is shorthand for *Erdős–Rényi selection.* Note that this neighborhood selection algorithm is equivalent to $\mathsf{ER\text{-}Emulation}_p(\mathtt{E}, \rho)$ from Section 4.3.3 when $\mathtt{E}(p) \coloneqq 1$ for all p.

Algorithm $\mathsf{ERS}(\rho)$

```
1: Let $N \coloneqq \varnothing$.
2: Let $P \coloneqq \mathcal{P}$.
3: while $P \neq \varnothing$ do
4:     Pick $r \in P$.
5:     Sample $c \xleftarrow{\$} \mathcal{U}([0,1])$.
6:     if $c \leq \rho$ then
7:         Update $N \coloneqq N \cup \{r\}$.
8:     Update $P \coloneqq P \setminus \{r\}$.
9: return $N$.
```

Remark 5.4.2. [Liu+22b, proof of Lemma 4.3.11] showed that this sampling can be done more efficiently by first sampling a number k from the binomial distribution with parameters n and ρ, and afterward, sample k parties from $\mathcal{P}$ without replacement.

Using this abstraction, we define the protocol $\mathsf{ERFlood}(\rho) \coloneqq \Pi_{\mathrm{Flood}}(\mathsf{ERS}(\rho))$ and next state that this is a weak flooding protocol. We will prove this in Section 5.4.1.1. Note that we, in theorem, explicitly quantify over the number of parties n after the existential quantification of the success probability bound, highlighting that the probability is independent of the number of parties.

Theorem 5.4.4. *There exists $\xi \in (0,1]$ s.t. for any $n \geq 11 \cdot \gamma^{-1}$ there is a $\rho = O(n^{-1})$ and $\Delta = O(\log(n) \cdot \Delta_{\mathrm{NET}})$ s.t. the protocol* $\mathsf{ERFlood}(\rho)$ *is a weak (Δ, ξ)-flooding protocol.*

Previous analysis [MNT22b, Corollary 3.6.3] of this protocol only showed the necessary condition that $\rho = \Omega(\log(n) \cdot n^{-1})$ so that the overall protocol delivers the message to all parties with constant probability. Our analysis instead proves that $\rho = O(n^{-1})$ is enough to guarantee that with a constant probability of success, any fixed party receives the message.

5.4.1.1 Security Analysis of ERFlood

To prove that $\mathsf{ERFlood}$ 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 idea of the honest sending process from [Liu+22b, Section 4.3.2.2] and the probability of a timely delivery to the probability that a party has an incoming edge in this graph. The idea of the honest sending process, which we recap below, is to let it mimic the sending of a particular 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. The definition of the honest sending process can be found in Section 4.3.2.2.

Similarly to [Liu+22b], we relate the probability of an event in the honest sending process (in our case that a party is in the neighborhood of the graph produced by the honest sending process) to the probability of a timely delivery and use this to show that $\mathsf{ERFlood}$ is a weak flooding protocol for specific parameters.

Lemma 5.4.5. *Let $\lambda \in \mathbb{N}$ be a distance, let $\rho \in [0,1]$ and let $\Delta := \lambda \cdot \Delta_{\mathrm{NET}}$. Further, let $s_{min} \in \mathcal{H}$ and $p_{min} \in \mathcal{H}$ s.t. when $G \xleftarrow{\$} \mathsf{HSP}(s_{min}, \mathsf{ERS}(\rho), \lambda)$ then $\Pr[p_{min} \in \Gamma^{\lambda}_{s_{min}}(G)]$ is minimized over all such $s, p \in \mathcal{H}$. If*

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

then $\mathsf{ERFlood}(\rho)$ is a weak (Δ, ξ)-flooding protocol.

Proof. Let $m_1, m_2, \ldots, m_\eta$ be the set of "fresh" (meaning that they have not been sent over any channel before) messages input to some honest party s in the execution of $\mathsf{ERFlood}(\rho)$ at time τ, and let p_i be an honest party. Now, we let $G_1 \xleftarrow{\$} \mathsf{HSP}(s, \mathsf{ERS}(\rho), \lambda), \ldots, G_\eta \xleftarrow{\$} \mathsf{HSP}(s, \mathsf{ERS}(\rho), \lambda)$ and let $T_1, \ldots, T_\eta$ be indicator variables such that T_j indicates if the event $p_i \in \Gamma^{\lambda}_{s}(G_j)$ happened. It is clear that $T_1, \ldots, T_\eta$ are independent. Furthermore, we have for any T_j that

$$\Pr[T_j = 1] = \Pr[p_i \in \Gamma^{\lambda}_{s}(G_j)] \ge \Pr[p_{\min} \in \Gamma^{\lambda}_{s_{\min}}(G)] \ge \xi. \tag{5.10}$$

To prove that $\mathsf{ERFlood}(\rho)$ is weak (Δ, ξ)-flooding protocol it is thus left to show that

$$\sum_{j=1}^{\eta} T_j \preceq \sum_{j=1}^{\eta} \mathtt{Timely}_{m_j,i}. \tag{5.11}$$

To do so, we follow the proof of [Liu+22b, Lemma 4.3.5] by coupling the execution of $\mathsf{ERFlood}(\rho)$ to the creation of a graph from the honest sending process. However, instead of creating just one graph, we observe the execution and create multiple graphs $G'_1, \ldots, G'_\eta$. For all G_j, we let $G'_j = (\mathcal{V}'_j, E'_j)$ and let the set of nodes be all honest parties (i.e., $\mathcal{V}'_j = \mathcal{H}$). We add an edge between to honest parties (p_k, p_z) to E'_j if and only if p_k sent the message m_j to p_z and p_z receives it before time $\tau + \lambda \cdot \Delta_{\mathrm{NET}}$. With this definition it is clear that if p_i has an incoming edge in G'_j then $\mathtt{Timely}_{m_j,i} = 1$. We now do a coupling between the graphs $G'_1, \ldots, G'_\eta$ and $G_1, \ldots G_\eta$ by first running the execution of $\mathsf{ERFlood}(\rho)$ and then define a new graphs $\widetilde{G_1}, \ldots, \widetilde{G_\eta}$ where each graph $\widetilde{G_j}$ will be defined in terms of G'_j. We define $\widetilde{G_j} = (\widetilde{\mathcal{V}_j}, \widetilde{E_j})$ by letting $\widetilde{\mathcal{V}_j} = \mathcal{H}$ and define $\widetilde{E_j}$ by duplicating the edges from E'_j to $\widetilde{E_j}$ for all parties within distance $\lambda - 1$ of s in G'_j.

Now, let $\widetilde{T_1}, \ldots, \widetilde{T_\eta}$ be indicator variables such that $\widetilde{T_j}$ indicates if the event $p_i \in \Gamma^{\lambda}_{s}(\widetilde{G_j})$ happened. Observe, that if a $\widetilde{T_j} = 1$ then party p_i has an incoming edge in

$\widetilde{G_j}$. This implies that party also has an edge in G'_j as $\widetilde{E_j} \subseteq E'_j$ which again (as argued before) implies that $\texttt{Timely}_{m_j,i} = 1$. Therefore, it is clear that

$$\sum_{j=1}^{\eta} \widetilde{T_j} \leq \sum_{j=1}^{\eta} \texttt{Timely}_{m_j,i}, \tag{5.12}$$

which again implies that for any $c \in \mathbb{R}$ we have that

$$\Pr\left[\sum_{j=1}^{\eta} \texttt{Timely}_{m_j,i} \leq c\right] \leq \Pr\left[\sum_{j=1}^{\eta} \widetilde{T_j} \leq c\right]. \tag{5.13}$$

It is thus left to show that $\sum_{j=1}^{\eta} T_j \sim \sum_{j=1}^{\eta} \widetilde{T_j}$. We will do so by arguing that for any G_j we have that $G_j \sim \widetilde{G_j}$ and that $\widetilde{G_j}$ independent. Note that each honest party selects their neighbors in $\mathsf{ERFlood}(\rho)$ using the same neighborhood selection used in the honest sending process. Furthermore, each honest party makes independent draws of neighbors for each different message. By assumption, each message is different, and this, therefore, ensures that the distributions of the edges in $\widetilde{G_j}$ are really independent. It is, therefore, only left to argue that there will not be any parties within distance $\lambda - 1$ in any graph G'_j that have not had their edges selected. This follows directly by the assumption that the delay on the point-to-point channels is at most Δ_{NET} and therefore, a party at a distance k in G'_j will at latest select their neighbors $k \cdot \Delta_{\mathrm{NET}}$ time after a message is input to the initial sender. □

Lemma 5.4.6. *Let $\delta_1, \delta_2, \alpha \in [0,1]$. Further, let $\phi \in \mathbb{R}$ be an expected expansion factor, $d \in \mathbb{R}$ be a constant, and let $\rho := \frac{d}{h}$, let $\lambda := \frac{\log\left(\frac{\alpha \cdot |\mathcal{H}|}{(1-\delta_1)\cdot d}\right)}{\log((1-\delta_2)\cdot\phi)}$. Finally, let $s \in \mathcal{H}$ and let $G \xleftarrow{\$} \mathsf{HSP}(s, \mathsf{ERS}(\rho), \lambda)$. If*

$$e^{-d\cdot\alpha} + \frac{\alpha \cdot \phi}{1-\alpha} \leq 1 \tag{5.14}$$

and

$$(1-\delta_2)\cdot\phi \geq 2, \tag{5.15}$$

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

$$\frac{\alpha \cdot |\mathcal{H}| - 1}{|\mathcal{H}|} \cdot \left(1 - e^{-\frac{\delta_1^2 \cdot d}{2}} - \frac{e^{-\frac{\delta_2^2\cdot(1-\delta_1)\cdot d}{2}}}{1 - e^{-\frac{\delta_2^2\cdot(1-\delta_1)\cdot d}{2}}}\right) \leq \Pr[p \in \Gamma_s^{\lambda}(G)].$$

Proof. We let Γ^{λ} be the set of neighbors of the sender s at a distance at most λ in a graph G. Using the law of total probability, we note that the probability that p is in the close neighborhood of the sender is lower bounded by the probability that the sender has a large honest neighborhood $\alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|$ for some $\alpha \in (0,1]$ *and* the party p is in fact in this neighborhood, i.e., $p \in \Gamma^{\lambda}$:

$$\begin{aligned}
&\Pr\left[p \in \Gamma^{\lambda}\right] = \Pr\left[p \in \Gamma^{\lambda} \cap \alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right] + \Pr\left[p \in \Gamma^{\lambda} \cap \alpha \cdot |\mathcal{H}| > |\Gamma^{\lambda}|\right] \\
&\geq \Pr\left[p \in \Gamma^{\lambda} \cap \alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right] \\
&= \Pr\left[p \in \Gamma^{\lambda} \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right] \cdot \Pr\left[\alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right].
\end{aligned} \tag{5.16}$$

We now bound these two probabilities individually.

Let us 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 the 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.

$$\begin{aligned}
&\Pr\left[p \in \Gamma^\lambda \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^\lambda|\right] \\
&= \sum_{a=1}^{|\mathcal{H}|} \Pr\left[p \in \Gamma^\lambda \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^\lambda| \cap |\Gamma^\lambda| = a\right] \cdot \Pr\left[|\Gamma^\lambda| = a \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^\lambda|\right] \\
&= \sum_{a=\alpha\cdot|\mathcal{H}|}^{|\mathcal{H}|} \Pr\left[p \in \Gamma^\lambda \mid |\Gamma^\lambda| = a\right] \cdot \Pr\left[|\Gamma^\lambda| = a \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^\lambda|\right] \\
&= \sum_{a=\alpha\cdot|\mathcal{H}|}^{|\mathcal{H}|} \frac{a-1}{|\mathcal{H}|} \cdot \Pr\left[|\Gamma^\lambda| = a \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^\lambda|\right] \\
&\geq \frac{\alpha \cdot |\mathcal{H}| - 1}{|\mathcal{H}|} \cdot \sum_{a=\alpha\cdot|\mathcal{H}|}^{|\mathcal{H}|} \Pr\left[|\Gamma^\lambda| = a \mid \alpha \cdot |\mathcal{H}| \leq |\Gamma^\lambda|\right] \\
&= \frac{\alpha \cdot |\mathcal{H}| - 1}{|\mathcal{H}|}.
\end{aligned} \tag{5.17}$$

Let us 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 reuse parts of a proof by Matt, Nielsen, and Thomsen [MNT22b, proof of Lemma 3.4.1] but look at a more specific setting where and directly insert the parameters for this setting. 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 minimum honest nodes within λ steps. We have that

$$\Pr\left[\alpha \cdot |\mathcal{H}| \leq |\Gamma^\lambda|\right] = 1 - \Pr[D], \tag{5.18}$$

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 event that the number of immediate neighbors of the sender deviates significantly from the mean

$$A_0 \coloneqq \big(|\theta^1| > (1 - \delta_1) \cdot d\big). \tag{5.19}$$

Next, we define the events that the number of neighbors in $\nu + 1$ is at least $(1 - \delta_2) \cdot \phi$ times the number of neighbors at step ν

$$B_\nu \coloneqq \left(|\theta^{\nu+1}| > (1 - \delta_2) \cdot \phi \cdot |\theta^\nu|\right). \tag{5.20}$$

Finally, we define the event that the neighbors within distance ν is at least $\alpha \cdot |\mathcal{H}|$

$$C_\nu \coloneqq (|\Gamma^\nu| \geq \alpha \cdot |\mathcal{H}|). \tag{5.21}$$

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

$$\lambda = \frac{\log\left(\frac{\alpha\cdot|\mathcal{H}|}{(1-\delta_1)\cdot d}\right)}{\log((1-\delta_2)\cdot\phi)} + 1 \iff (1-\delta_1)\cdot d\cdot((1-\delta_2)\cdot\phi)^{\lambda-1} = \alpha\cdot|\mathcal{H}|. \tag{5.22}$$

Therefore, if A_0 and $B_1, \ldots, B_\lambda$ holds, then

$$\begin{aligned}
|\Gamma^\lambda| &= \sum_{\nu=0}^{\lambda}|\theta^\nu| \\
&= 1 + \sum_{\nu=0}^{\lambda-1}|\theta^{\nu+1}| \\
&\geq 1 + \sum_{\nu=0}^{\lambda-1}((1-\delta_2)\cdot\phi)^\nu\cdot|\theta^1| \\
&\geq 1 + (1-\delta_1)\cdot d\cdot\sum_{\nu=0}^{\lambda-1}((1-\delta_2)\cdot\phi)^\nu \\
&> (1-\delta_1)\cdot d\cdot((1-\delta_2)\phi)^{\lambda-1} \\
&= \alpha\cdot|\mathcal{H}|.
\end{aligned} \tag{5.23}$$

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

$$\left(\bigwedge_{\nu=0}^{\lambda-1} A_\nu \implies \neg D\right) \iff \left(D \implies \bigvee_{\nu=0}^{\lambda-1}\neg A_\nu\right). \tag{5.24}$$

Hence, we get the following:

$$\begin{aligned}
\Pr[D] &\leq \Pr\left[\bigcup_{i=0}^{\lambda-1}\neg A_i\right] \\
&\leq \sum_{i=0}^{\lambda-1}\Pr\left[\neg A_i \mid \bigcap_{j<i} A_j\right] \\
&= \Pr[\neg A_0] + \sum_{i=1}^{\lambda-1}\Pr\left[\neg A_i \mid \bigcap_{j<i} A_j\right] \\
&= \Pr[\neg A_0] + \sum_{i=1}^{\lambda-1}\Pr\left[\neg B_i \cap \neg C_i \mid \bigcap_{j<i} A_j\right] \\
&\leq \Pr[\neg A_0] + \sum_{i=1}^{\lambda-1}\Pr\left[\neg B_i \mid \bigcap_{j<i} A_j \cap \neg C_i\right] \\
&= \Pr[\neg A_0] + \sum_{i=1}^{\lambda-1}\Pr\left[\neg B_i \mid \bigcap_{1\leq j<i} B_j \cap \neg C_i \cap A_0\right].
\end{aligned} \tag{5.25}$$

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

Claim 5.4.7 (Fast expansion to small fraction)**.** *For any $i \in \{1, \dots, \lambda - 1\}$ we have*

$$\Pr\left[\neg B_i \mid \bigcap_{1\le j<i} B_j \cap \neg C_i \cap A_0\right] \le e^{-\frac{\delta_2^2\cdot((1-\delta_2)\cdot\phi)^{\nu}\cdot(1-\delta_1)\cdot d}{2}}.$$

Proof. We look at the probability space where $\bigcap_{1\le j<i} B_j \cap \neg C_i \cap A_0$ holds. We define the number of 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 by $U := \mathcal{H} \setminus \Gamma^{\nu}$. For each $u \in U$ we introduce an indicator variable X_u which indicates if u is in $\theta^{\nu+1}$. As the probability that there is an edge between any two honest parties is independent of other edges, we have

$$\Pr[X_u = 1] = 1 - (1-\rho)^r \ge 1 - e^{-\rho r}. \tag{5.26}$$

The size of $\theta^{\nu+1}$ is the sum of these independent variables, i.e.,

$$|\theta^{\nu+1}| = \sum_{u\in U} X_u. \tag{5.27}$$

As we are looking at the case where $\neg C_i$, we have $|U| \ge (1-\alpha)\cdot|\mathcal{H}|$ which by linearity of expectations gives us that

$$\mathbb{E}[|\theta^{\nu+1}|] \ge |\mathcal{H}|\cdot(1-\alpha)\cdot\left(1-e^{-\rho r}\right). \tag{5.28}$$

We now subtract $\phi\cdot r$ on each side of the inequality above and insert $\rho = \frac{d}{h} \le \frac{d}{|\mathcal{H}|}$, to show that the expected value is larger than $\phi \cdot r$. We get

$$\begin{aligned} \mathbb{E}[|\theta^{\nu+1}|] - \phi\cdot r &\ge |\mathcal{H}|\cdot(1-\alpha)\cdot\left(1 - e^{-\rho r} - \frac{\phi\cdot r}{|\mathcal{H}|\cdot(1-\alpha)}\right) \\ &= |\mathcal{H}|\cdot(1-\alpha)\cdot\left(1 - e^{-d\frac{r}{|\mathcal{H}|}} - \frac{\phi\cdot r}{|\mathcal{H}|\cdot(1-\alpha)}\right). \end{aligned} \tag{5.29}$$

We let $x = \frac{r}{|\mathcal{H}|}$ and set $f(x) = 1 - e^{-dx} - x\frac{\phi}{(1-\alpha)}$. We differentiate this twice and find $f''(x) = -d^2e^{-dx} \le 0$, which implies that f is concave, which again means that the minimum values are at one of the endpoints of the function. As $x \in [0,\alpha]$ it is enough to check that $f(0) \ge 0$ and $f(\alpha) \ge 0$ which will imply that $\mathbb{E}[|\theta^{\nu+1}|] \ge \phi\cdot|\theta^{\nu}|$.

$$\begin{aligned} &f(0) = 1 - e^{-d\cdot 0} - 0 = 0. \\ &f(\alpha) = 1 - e^{-d\cdot\alpha} - \frac{\alpha\cdot\phi}{1-\alpha} \ge 0 \iff e^{-d\cdot\alpha} + \frac{\alpha\cdot\phi}{1-\alpha} \le 1. \end{aligned} \tag{5.30}$$

We now use Chernoff (Lemma 1.3.1) to bound the probability that this is not the case which means that for any $\delta_2 \in [0,1]$, we get that

$$\Pr[|\theta^{\nu+1}| \le (1-\delta_2)\cdot\phi\cdot|\theta^{\nu}|] \le e^{-\frac{\delta^2\cdot\phi\cdot|\theta^{\nu}|}{2}} \tag{5.31}$$

However, $\bigcap_{j<i} B_j \cap A_0$ and $(1-\delta_2)\cdot\phi \ge 1$ ensures that

$$|\theta^{\nu}| \ge ((1-\delta_2)\cdot\phi)^{\nu}\cdot(1-\delta_1)\cdot d. \tag{5.32}$$

Hence, within the probability space where $\bigcap_{1\le j<i} B_j \cap \neg C_i \cap A_0$ holds we have that

$$\Pr[|\theta^{\nu+1}| \le (1-\delta_2)\cdot\phi\cdot|\theta^{\nu}|] \le e^{-\frac{\delta_2^2\cdot((1-\delta_2)\cdot\phi)^{\nu}\cdot(1-\delta_1)\cdot d}{2}}. \tag{5.33}$$

□

We apply Chernoff for bounding the probability of the event A_0, the claim we have just proven, and Lemma 5.2.5 to the bound we established in Equation (5.25) to obtain a final bound for the probability of the event D

$$\begin{aligned}
\Pr[D] &\leq e^{-\frac{\delta_1^2 \cdot d}{2}} + \sum_{\nu=1}^{\lambda-1} e^{-\frac{\delta_2^2 \cdot ((1-\delta_2)\cdot\phi)^{\nu} \cdot (1-\delta_1)\cdot d}{2}} \\
&< e^{-\frac{\delta_1^2 \cdot d}{2}} + \sum_{\nu=1}^{\infty} e^{-\frac{\delta_2^2 \cdot ((1-\delta_2)\cdot\phi)^{\nu} \cdot (1-\delta_1)\cdot d}{2}} \\
&= e^{-\frac{\delta_1^2 \cdot d}{2}} + \sum_{\nu=1}^{\infty} \left(e^{-\frac{\delta_2^2 \cdot (1-\delta_1)\cdot d}{2}} \right)^{(((1-\delta_2)\cdot\phi)^{\nu})} \\
&\leq e^{-\frac{\delta_1^2 \cdot d}{2}} + \frac{e^{-\frac{\delta_2^2 \cdot (1-\delta_1)\cdot d}{2}}}{1 - e^{-\frac{\delta_2^2 \cdot (1-\delta_1)\cdot d}{2}}}.
\end{aligned} \tag{5.34}$$

Hence, it follows that

$$\Pr\left[\alpha \cdot |\mathcal{H}| \leq |\Gamma^{\lambda}|\right] \geq 1 - e^{-\frac{\delta_1^2 \cdot d}{2}} - \frac{e^{-\frac{\delta_2^2 \cdot (1-\delta_1)\cdot d}{2}}}{1 - e^{-\frac{\delta_2^2 \cdot (1-\delta_1)\cdot d}{2}}}. \tag{5.35}$$

The desired bound on the probability for late delivery now follows from Equations (5.16), (5.17) and (5.35).

□

Using Lemmas 5.4.5 and 5.4.6, we are now ready to prove our main theorem for ERFlood. For completeness, we restate it below.

Theorem 5.4.4. *There exists $\xi \in (0,1]$ s.t. for any $n \geq 11 \cdot \gamma^{-1}$ there is a $\rho = O(n^{-1})$ and $\Delta = O(\log(n) \cdot \Delta_{\text{NET}})$ s.t. the protocol* ERFlood(ρ) *is a weak (Δ, ξ)-flooding protocol.*

Proof. We let $\delta_1 \coloneqq \delta_2 \coloneqq \alpha \coloneqq \frac{1}{10}$ and let $\phi \coloneqq \frac{20}{9}$. This ensures that Equation (5.15) is always fulfilled and if we let

$$d \geq -10 \cdot \ln\left(\frac{61}{81}\right) \tag{5.36}$$

then also Equation (5.14) is fulfilled. Hence, if we let $\rho = \frac{d}{h}$ then Lemma 5.4.6 ensures that for $\lambda = O(\log(n))$ then for any sender s and receiver p (in particular also the sender and receiver that minimizes the probability for the event below) then if $G \overset{\$}{\leftarrow}$ HSP$(s, \text{ERS}(\rho), \lambda)$ then

$$\frac{h-10}{10 \cdot h} \cdot \left(1 - e^{-\frac{d}{200}} - \frac{e^{-\frac{d}{180}}}{1 - e^{-\frac{d}{180}}}\right) \leq \Pr[p \in \Gamma_s^{\lambda}(G)]. \tag{5.37}$$

We note that $n \geq 11 \cdot \gamma^{-1} \implies h \geq 11$ and therefore that

$$\frac{h-10}{10 \cdot h} \geq \frac{1}{110}. \tag{5.38}$$

Hence, for large enough (but still constant) d then $\xi := \frac{1}{110} \cdot \left(1 - e^{-\frac{d}{200}} - \frac{e^{-\frac{d}{180}}}{1-e^{-\frac{d}{180}}}\right) \in (0, 1]$ is a constant independent of n. Lemma 5.4.5 implies that $\mathsf{ERFlood}(\rho)$ is a weak (Δ, ξ)-flooding protocol with $\Delta = \Delta_{\text{NET}} \cdot \lambda = O(\log(n) \cdot \Delta_{\text{NET}})$. Further, note that this holds when $\rho = \frac{d}{h} = O(\gamma^{-1} \cdot n^{-1}) = O(n^{-1})$, where the last equality follows because we assume that a constant fraction of the parties are honest. □

5.4.2 Flooding Amplification

We present a compiler that amplifies delivery guarantees of a weak flooding protocol to full-fledge 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 it into multiple shares using the ECCS. The sender will then send 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, suppose everybody receives enough original shares within the given time. In that case, the only thing that can prevent an honest party from reconstructing the message sent by the sender is if an adversary manages 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, a proof, 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-freeness property of the accumulator scheme to inject such false shares.

Therefore, it is only left to ensure that all parties receive enough of the original shares. We will ensure this by instantiating WeakFlood2Flood with a weak flooding protocol ERFlood, which we will instantiate 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.

The formal description of the protocol can be found below.

Protocol WeakFlood2Flood(Π, ζ, α)

The protocol is parameterized by a weak flooding protocol Π, a (μ, e)-ECCS ζ for some $\mu, e \in \mathbb{N}$, 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 $(\mathcal{S}end, m)$ 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 $z, \pi_1, \ldots, \pi_\mu = \alpha.\mathsf{Accumulate}(\{(s_i, i) \mid 1 \le i \le \mu\})$. The party now draws a uniformly sampled random number $r \xleftarrow{\$} \mathcal{U}(\{0,1\}^\kappa)$.

Now, the party inputs the set of messages $\{(s_j, j, r, \pi_j, z) \mid 1 \leq j \leq \mu\}$ to Π. Finally, they add m to $\texttt{Received}_i$.

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

When party p_i receives a tuple (s, j, r, π, z) in Π where $\alpha.\mathsf{Verify}((s, j), \pi, z) = \top$ they add (s, j) to $\texttt{ReceivedShares}_i[z]$. Furthermore, p_i check if $|\texttt{ReceivedShares}_i[z]| \geq \mu - e$. If that is the case p_i does the following:

1. Obtains a sequence of shares $s_1, \dots, s_\mu$ 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, $\texttt{Received}_i \coloneqq \texttt{Received}_i \cup \{\zeta.\mathsf{Dec}(s_1, \dots, s_\mu)\}$.

At first, it might seem strange that a sender samples a random number and attaches this to each share. The reason for this is that the weak flooding protocol we will instantiate WeakFlood2Flood with will only ensure that sufficiently many shares are delivered if they are sent for the first time in the execution of the protocol. Hence, to ensure that the probability that this happens is negligible in κ, we attach the random number. We note that the birthday paradox bounds the probability that an adversary manages to do so throughout the execution. For clarity of presentation, we will not propagate this exact probability through our proofs but merely assume that an adversary does not guess such a random number that an honest party will later attach to their message. Similarly, we will also assume that the adversary that no collision happens for the cryptographic accumulator.

Security of WeakFlood2Flood. We now state and prove the security of WeakFlood2Flood, given that the protocol is instantiated with a weak flooding protocol.

Theorem 5.4.8. *Let $\xi \in (0, 1]$, let $\Delta \in \mathbb{N}$, and let Π be a weak (Δ, ξ)-flooding protocol. Further, let $\delta \in (0, 1]$, let $\mu \in \mathbb{N}$, let $e \geq \mu \cdot (1 - (1 - \delta) \cdot \xi)$, let ζ be a (μ, e)-ECCS, and let α be a WSCAS. The probability that a message sent in the protocol* $\mathsf{WeakFlood2Flood}(\Pi, \zeta, \alpha)$ *is not delivered within Δ to all parties is less than*

$$|\mathcal{H}| \cdot e^{-\frac{\delta^2 \cdot \xi \cdot \mu}{2}}.$$

Proof. Let us look at a particular message sent by an honest sender s at time τ by flooding the shares $s_1, \dots, s_\mu$. We note that if a party p receives more than $(1 - \delta) \cdot \xi \cdot \mu$ number of shares, then by properties of the (μ, e)-ECCS, then p can reconstruct the message unless p received some shares different from the original shares and tries to reconstruct from these. However, if this happens, this means that an honest party added a share and share number (s, i) to the set of shares for a particular accumulator, where (s, i) was not a part of the original shares used to construct the accumulated value. Hence, this only happens with the same probability as a collision in the accumulation scheme. It is, therefore, sufficient to bound the probability that there exists a party that does not receive $(1 - \delta) \cdot \xi \cdot \mu$ shares from the sender.

For each honest party $p_i \in \mathcal{H}$, we introduce notation for random indicator variables $S_{i,1}, S_{i,2}, \dots, S_{i,\mu}$ where $S_{i,j}$ indicates whether or not party p_i received share s_j before time $\tau + \lambda \cdot \Delta_{\text{NET}}$. 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 $\lambda \cdot \Delta_{\text{NET}}$. Because Π is a weak (ξ, Δ)-flooding protocol and the shares are now input to Π as messages for the first time, we know that there exists a sequence of independent variables $S^*_{i,1}, S^*_{i,2}, \dots, S^*_{i,\mu}$ where $\Pr[S^*_{i,j} = 1] \geq \xi$. Furthermore, if we let their sum be given by $S^*_i = \sum_{j=1}^{\mu} S^*_{i,j}$ then we know that that S_i stochastically dominates S^*_i. That is that for any $k \in \mathbb{R}$ we have

$$\Pr[S_i \leq k] \leq \Pr[S^*_i \leq k]. \tag{5.39}$$

Hence, it is sufficient to bound the probability for the event that $S^*_i \leq (1-\delta) \cdot \xi \cdot \mu$ to bound the probability that party p_i receives less than $(1-\delta) \cdot \xi \cdot \mu$ shares before $\tau + \Delta$. By linearity of expectation, we have that

$$\mathbb{E}[S^*_i] \geq \xi \cdot \mu. \tag{5.40}$$

Chernoff (Lemma 1.3.1) gives us that for any $\delta \in [0, 1)$ we have

$$\Pr\left[S^*_i \leq (1-\delta) \cdot \xi \cdot \mu\right] \leq e^{-\frac{\delta_3^2 \cdot \xi \cdot \mu}{2}}. \tag{5.41}$$

Finally, we calculate the probability that there *exists any honest party* that does not receive at least $(1-\delta) \cdot \xi \cdot \mu$ shares using the union bound:

$$\begin{aligned} &\Pr\left[\exists p_i \in \mathcal{H},\ \text{s.t. } p_i \text{ receives less than } (1-\delta) \cdot \xi \cdot \mu \text{ shares}\right] \\ &\leq \sum_{p_i \in \mathcal{H}} \Pr\left[S_i \leq (1-\delta) \cdot \xi \cdot \mu\right] \\ &\leq |\mathcal{H}| \cdot e^{-\frac{\delta_3^2 \cdot \xi \cdot \mu}{2}}. \end{aligned} \tag{5.42}$$

□

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

Corollary 5.4.9. *Let $\xi \in (0, 1]$, let $\Delta \in \mathbb{N}$, and let Π be a weak (Δ, ξ)-flooding protocol. Further, let $\mu \geq \frac{\log(n)+\kappa}{\xi}$, let $e \geq \mu \cdot \left(1 - \frac{\xi}{2}\right)$, let ζ be a (μ, e)-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 5.4.8 instantiated with $\delta := \frac{1}{2}$. □

5.4.3 Putting It All Together

We now consider the instantiation ECFlood (from "erasure coded flood"), defined as ECFlood(ρ, ζ, α) := WeakFlood2Flood(ERFlood$(\rho), \zeta, \alpha)$, where the protocol compiler WeakFlood2Flood is instantiated with the weak flooding ERFlood(ρ) for $\rho \in (0, 1]$, ζ is a (μ, e)-ECCS and α is WSCAS scheme.

Communication complexity analysis. Consider the case where a single message of length l is input to an honest party, and let S denote the total number of messages sent by honest parties. Then, the total communication complexity of $\mathsf{ECFlood}(\rho, \zeta, \alpha)$ is upper bounded by S times the size of each of the messages sent for each share.

Each message consists of a share, the sequence number of the share, nonce of length κ, an accumulator proof, and an accumulated value. Hence, the total communication complexity will be upper bounded by

$$S \cdot (\zeta.\mathtt{ShareSize}(l) + \log(\mu) + \kappa + \alpha.\mathtt{ProofSize}(\mu) + \alpha.\mathtt{AccSize}). \tag{5.43}$$

Let us now bound the number of messages sent by honest parties S. For each share, the expected size of the neighborhood for a party is given by the probability of selecting each party as a neighbor times the number of parties. If we let $N \overset{\$}{\leftarrow} \mathsf{ERS}(\rho)$, then the expected size is given by $\mathbb{E}[|N|] = \rho \cdot n$. The total number of messages sent is upper bounded by the scenario where all parties are honest and forward a message for each share exactly once. By linearity of expectation, we get that the expected number of messages is given by

$$\mathbb{E}[S] \leq \mu \cdot \rho \cdot n^2. \tag{5.44}$$

By Chernoff (Lemma 1.3.1), we have that for any $\delta \in [0, 1]$,

$$\Pr\Big[S \geq (1 + \delta) \cdot \mu \cdot \rho \cdot n^2\Big] \leq e^{-\frac{\delta^2 \cdot \mu \cdot \rho \cdot n^2}{3}}. \tag{5.45}$$

Hence, if $\mu \geq \kappa$ and $\rho = \Omega\big(n^{-2}\big)$, then the probability that $S = O(\mu \cdot \rho \cdot n^2)$ is overwhelming in κ. Consequently, we have that the communication complexity is upper bounded by

$$O(\mu \cdot \rho \cdot n^2 \cdot (\zeta.\mathtt{ShareSize}(l) + \log(\mu) + \kappa + \alpha.\mathtt{ProofSize}(\mu) + \alpha.\mathtt{AccSize})), \tag{5.46}$$

with a probability that is overwhelming in κ. We further note that if additionally $\rho = \Omega\big(n^{-1}\big)$, then the number of parties a single party has to connect with in the protocol is also bounded by

$$O(\rho \cdot n \cdot \mu), \tag{5.47}$$

by Chernoff (Lemma 1.3.1), with a probability overwhelming in κ. Thereby, the communication for each party will be upper bounded by:

$$O(\mu \cdot \rho \cdot n \cdot (\zeta.\mathtt{ShareSize}(l) + \log(\mu) + \kappa + \alpha.\mathtt{ProofSize}(\mu) + \alpha.\mathtt{AccSize})), \tag{5.48}$$

Below we state a simple corollary of Corollary 5.4.9 and Theorem 5.4.4 that bounds the necessary parameters to instantiate $\mathsf{WeakFlood2Flood}$ securely with $\mathsf{ERFlood}$.

Corollary 5.4.10. *There exists $\xi \in (0, 1]$ s.t. for any $n \geq 11 \cdot \gamma^{-1}$ there is a $\rho = O(n^{-1})$ and $\Delta = O(\log(n) \cdot \Delta_{\mathrm{NET}})$ s.t. if $\mu \geq \frac{\log(n)+\kappa}{\xi}$, $e \geq \mu \cdot \Big(1 - \frac{\xi}{2}\Big)$, ζ is a (μ, e)-ECCS, and α is a WSCAS, then the protocol $\mathsf{ECFlood}(\rho, \zeta, \alpha)$ is a strong Δ-flooding protocol.*

Using Reed-Solomon codes and since $e \geq \mu \cdot \Big(1 - \frac{\xi}{2}\Big)$, we have that Equation (5.1) implies that each share size is bounded by:

$$\zeta.\mathtt{ShareSize} = O\bigg(\frac{l}{\mu \cdot \xi}\bigg) = O\bigg(\frac{l}{\mu}\bigg). \tag{5.49}$$

Furthermore, using efficient accumulators (see Equation (5.2)) that have accumulator and proof sizes of $O(\kappa)$ bits, and setting $\mu = \frac{\log(n)+\kappa}{\xi}$, we obtain from Equation (5.46) that the total communication complexity is bounded by

$$O(n \cdot (l + (\log(n) + \kappa) \cdot (\log(\log(n)) + \kappa))). \tag{5.50}$$

Note that this is optimal if

$$l = \Omega\big((\log(n) + \kappa) \cdot (\log(\log(n)) + \kappa)\big). \tag{5.51}$$

Similarly, by Equation (5.47) each party has at most $O(\log(n) + \kappa)$ neighbors and by Equation (5.48) the per-party communication is bounded by $O(l + (\log(n) + \kappa) \cdot (\log(\log(n)) + \kappa))$.

5.5 Flooding in the Weighted Setting

So far, we considered the setting where a certain fraction of parties may behave adversarially. However, blockchain systems are often based on the assumption that the adversary controls a fraction of the total weight of a resource. Leveraging ideas from [Liu+22b], we provide a general transformation for a flooding protocol in the equal-weights setting to security in the weighted setting.

Model. Before describing the transformation, we briefly recap the model of public weighted parties. We assume that a public weight is assigned to each party. 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.

We allow an adversary to 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. Formally, we assume that,

$$\sum_{p \in \mathcal{H}} \alpha_p \geq \widetilde{\gamma}, \tag{5.52}$$

and that all parties have a non-zero positive weight i.e. $\forall p \in \mathcal{P}, W_p > 0$.

Transformation. 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 [Liu+22b] 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

$$\mathtt{E}(p) := \{p_i \mid i \in \mathbb{N} \wedge i \leq \lceil \alpha_p \cdot n \rceil\}. \tag{5.53}$$

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 $\mathtt{E}(p) \neq \varnothing$. For convenience, we introduce notation for the set of emulated parties, $\mathcal{P}_\mathtt{E} = \bigcup_{p \in \mathcal{P}} \mathtt{E}(p)$, the total number of emulated parties $n_\mathtt{E} = |\mathcal{P}_\mathtt{E}|$, the set of emulated parties that are emulated by honest players $\mathcal{H}_\mathtt{E} =$

$\bigcup_{p \in \mathcal{H}} \mathtt{E}(p)$ and the number of honestly emulated parties $h_{\mathtt{E}} = |\mathcal{H}_{\mathtt{E}}|$. Following [Liu+22b], we note that

$$n_{\mathtt{E}} = \sum_{p \in \mathcal{P}} \lceil \alpha_p \cdot n \rceil \leq \sum_{p \in \mathcal{P}} \alpha_p \cdot n + 1 = 2 \cdot n, \tag{5.54}$$

and

$$h_{\mathtt{E}} = \sum_{p \in \mathcal{H}} \lceil \alpha_p \cdot n \rceil \geq \widetilde{\gamma} \cdot n. \tag{5.55}$$

When defining a strong flooding protocol in Section 5.2.2, 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 worked for the same set of assumed parties 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 saying 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 $(\mathcal{Send}, m)$ they pick $p_i \in \mathtt{E}(p)$ and forward $(\mathcal{Send}, m)$ to p_i in Π.

Get Messages: When p receives $(\mathcal{GetMessages})$ they pick $p_i \in \mathtt{E}(p)$ forward $(\mathcal{GetMessages})$ 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 itself be a strong flooding protocol.

Theorem 5.5.1. *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 input to some honest party at time τ. 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 τ. Because Π is a strong Δ-flooding protocol for $\mathcal{P}_{\mathtt{E}}$ when $\widetilde{\gamma} \cdot n$ parties are honest (Equation (5.55) ensures that this is the case), then there is an overwhelming probability in κ that all emulated parties receive m before $\tau + \Delta$. As each honest party emulates at least one party, this implies that all honest parties will also receive the message with an overwhelming probability in Π. □

Realising a strong flooding protocol for $\mathcal{P}_{\mathtt{E}}$. At first glance, it may seem like that Theorem 5.5.1 allows us to translate the protocols presented in Section 5.4 to the weighted setting without further work. 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 for the emulated set of parties $\mathcal{P}_{\mathtt{E}}$. To use these protocols, blackbox in the weighted setting, 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 regular channel, they will take it as an input on the emulated channel between the emulated parties p_e and $p_{e'}$.

Communication complexity analysis. We note that the analysis of the communication complexity presented in Section 5.4.3 also applies when the protocol is transformed to work for the weighted setting because $n_{\mathtt{E}} = O(n)$ (Equation (5.54)) and the fraction of honest emulated parties $\frac{h_{\mathtt{E}}}{n_{\mathtt{E}}} = O(\widetilde{\gamma})$ (by Equations (5.54) and (5.55)). The only change 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 5.4.3, we see that for a suitable ρ, ζ and α, the communication complexity of the $\mathsf{Flood2WeightedFlood}(\mathsf{ECFlood}(\rho, \zeta, \alpha))$ is bounded by

$$O(n \cdot (l + (\log(n) + \kappa)^2)). \tag{5.56}$$

That is $\mathsf{Flood2WeightedFlood}(\mathsf{ECFlood}(\rho, \zeta, \alpha))$ has an optimal communication complexity when $l = \Omega\big((\log(n) + \kappa)^2\big)$. 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, $\mathsf{Flood2WeightedFlood}(\mathsf{ECFlood}(\rho, \zeta, \alpha))$ 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 delivered to *all* emulated parties of p_j. In that case, the communication complexity of $\mathsf{Flood2WeightedFlood}(\mathsf{ECFlood}(\rho, \zeta, \alpha))$ would be optimal under the same constraints as $\mathsf{ECFlood}(\rho, \zeta, \alpha)$.

5.6 Estimating Practical Parameters for ECFlood Using Probabilistic Simulations

In Section 5.4.3, it was shown that parameters could be set such that ECFlood theoretically constitutes an asymptotically optimal flooding algorithm. To show this, we instantiated many variables to constants required by our analysis, but these are most likely not instantiated anywhere near optimally, nor are our analyses themselves optimal. To provide a guideline for how to instantiate the parameters of our protocol for practical performance, we made probabilistic simulations of the protocol executions that explore how the message complexity of our protocols is affected by adjusting parameters.

5.6.1 Setup for Simulations

We implemented a probabilistic experiment among n nodes where $\frac{n}{2}$ of them behave honestly and will actually forward any received message according to our protocol if they receive a message. We consider not forwarding any messages as the worst-case behavior of an adversary. Therefore, 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

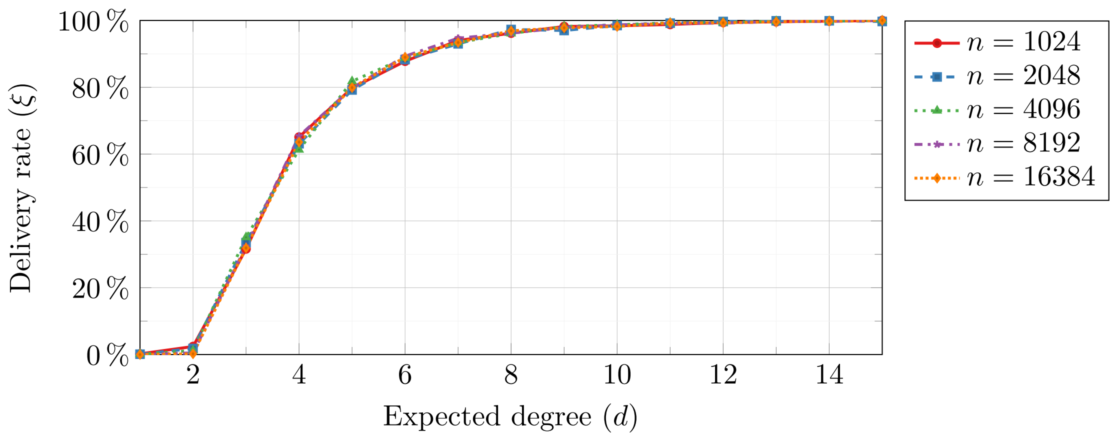


Figure 5.3: Results for simulations of ERFlood$\left(\frac{d}{n}\right)$ for different values of d and n. The delivery rate for a fixed party different from the sender is plotted as a function of changing the expected degree d.

distribute several messages by selecting a set of neighbors using ERS$\left(\frac{d}{n}\right)$ (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 ERFlood, 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. Below, we report on various statistics from these simulations.

5.6.2 Estimating Parameters for ERFlood

In Section 5.4.3 it was shown that the communication complexity of WeakFlood2Flood instantiated with ERFlood$\left(\frac{d}{n}\right)$ is directly proportional to $\frac{d}{\xi}$, if ERFlood constitutes a weak (Δ, ξ) flooding protocol for some Δ and ξ.

Let's first ignore how Δ is affected by changing the expected degree d and focus solely on finding estimates of $\frac{d}{\xi}$ for different expected degrees. A simple way to estimate the maximum ξ for which ERFlood$\left(\frac{d}{n}\right)$ 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 can be found in Figure 5.3.

It is noteworthy that the graphs for different values of n are incredibly close. When ERFlood$\left(\frac{d}{n}\right)$ is executed, it is clear that d will be the expected size of the neighborhood of each party. Hence, our simulations confirm that the delivery rate of ERFlood$\left(\frac{d}{n}\right)$ (and thereby how well it acts as a weak flooding algorithm) is genuinely independent of the number of parties in the protocol 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 seems to be about $\sim 30\%$. 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 5.4.

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 50% of the parties.

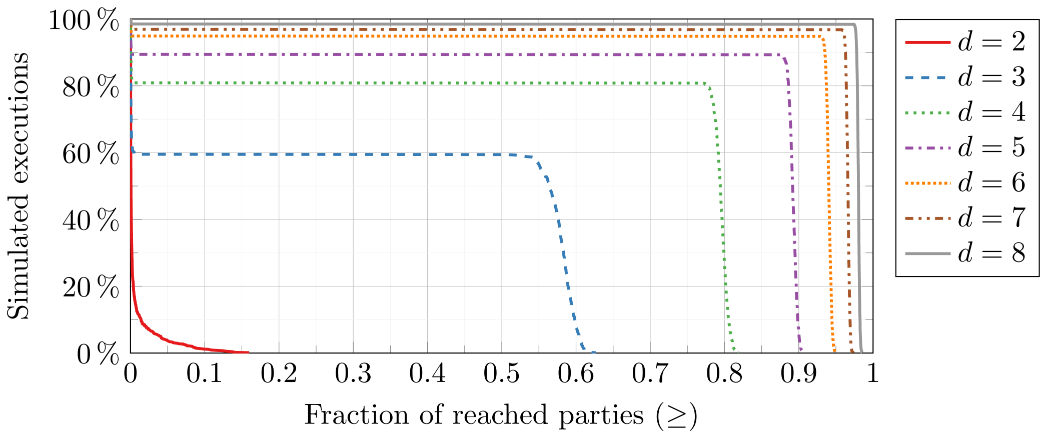


Figure 5.4: Results for simulations of ERFlood$\left(\frac{d}{n}\right)$ for different values of d and a fixed $n = 8192$. The graphs show the percentages of the simulated executions (on the y-axis) where at least a certain fraction of parties received the message (on the x-axis).

With estimates on ξ, we can also estimate the size of the factor $\frac{d}{\xi}$ multiplied by the message length in the communication complexity. In Figure 5.5, we plot this. Our simulations indicate that the best value for d to get the lowest overall communication complexity is ~ 5 where $\frac{d}{\xi}$ will be just above 6.

We now turn our attention to how the Δ parameter of the weak flooding algorithm ERFlood$\left(\frac{d}{\xi}\right)$ 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 5.6, 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 expected degrees d. We include only those degrees of interest w.r.t. an overall low communication complexity.

We note that during the execution, no message was delivered in more than 33 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.

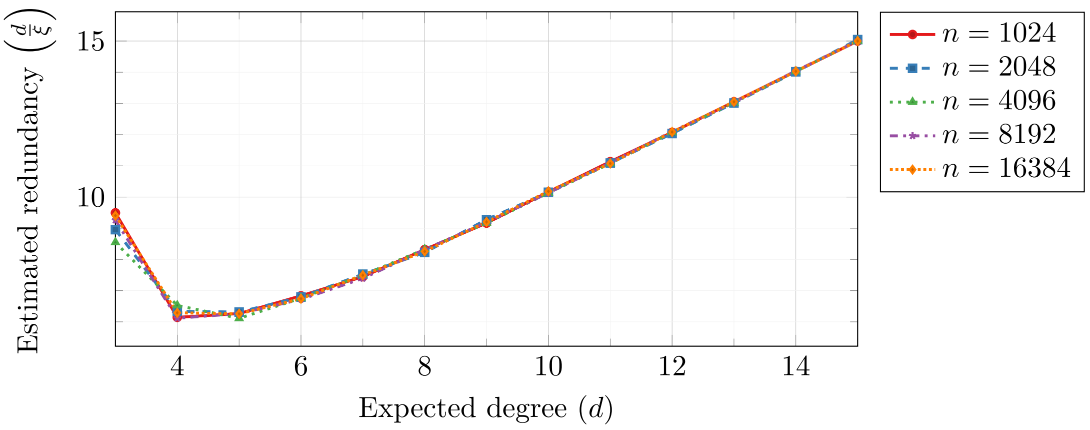


Figure 5.5: Results for simulations of ERFlood$\left(\frac{d}{n}\right)$ for different values of d and n. The graphs show the expected degree normalized by the observed delivery rate as a function of the expected degree.

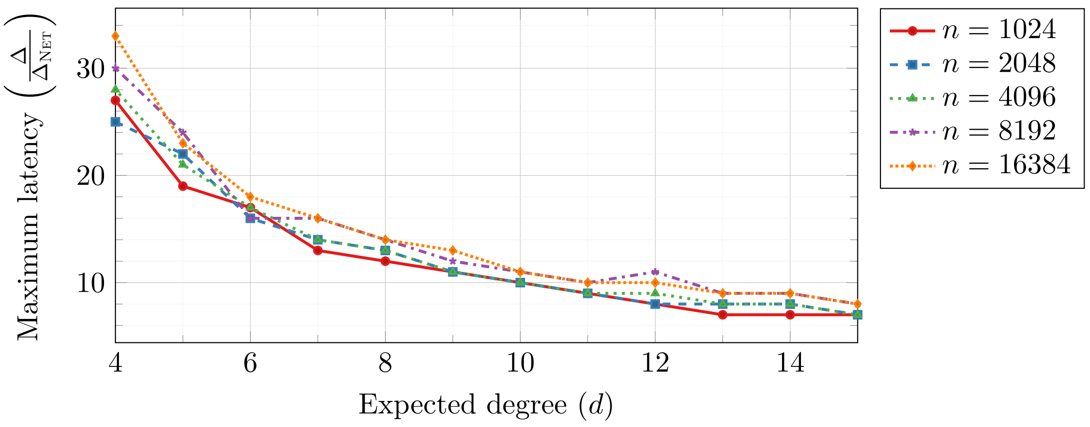


Figure 5.6: Results for simulations of ERFlood$\left(\frac{d}{n}\right)$ for different values of d and n. The graphs show the maximal latency from the sender to any party that receives the message for different values of d.

5.6.3 Estimating Parameters for ECFlood

Even though we in the previous section established that ERFlood$\left(\frac{d}{n}\right)$ delivers the "best" weak flooding network when d is approximately 5, this does not necessarily translate to the protocol WeakFlood2Flood instantiated with ERFlood$\left(\frac{d}{n}\right)$. The reason is that WeakFlood2Flood also needs to be instantiated with an (μ, e)-ECCS, and the redundancy of the ECCS scheme is proportional to $(\mu - e)^{-1}$. For WeakFlood2Flood to be secure, we need that all parties receive at least $\mu - e$ shares. In expectation, all parties will receive $\xi \cdot \mu$ when instantiated with a weak ξ-flooding protocol. However, 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 e for a ζ that is

(μ, e)-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, which is also a parameter of ECFlood, as we will not do any simulations w.r.t. the efficiency of such a scheme. Sometimes we will also leave out the ECCS of the notation and treat the number of shares explicitly.

To estimate how to set e 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 acquired shares across all simulations, β, is affected by a varying number of shares μ and degrees d. The results of this can be found in Figure 5.7.

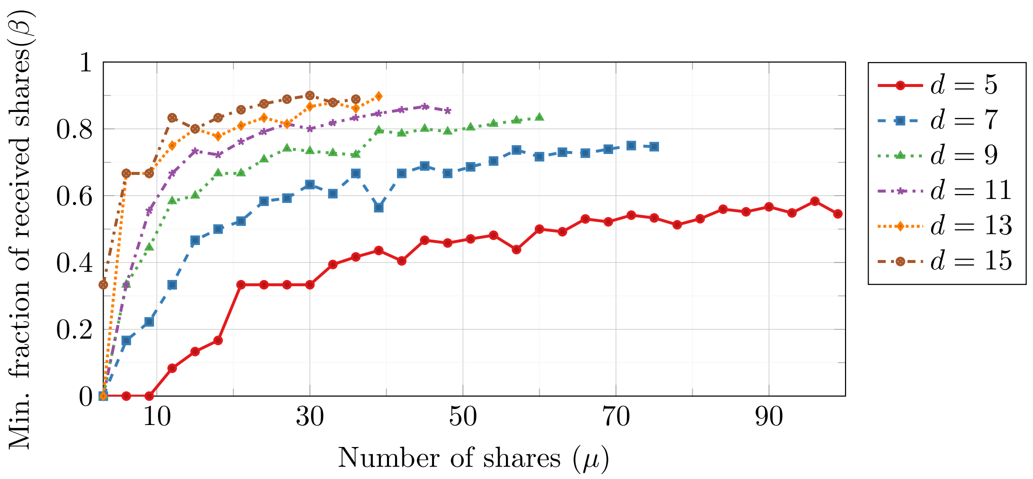


Figure 5.7: Results for simulations of $\mathsf{ECFlood}(\frac{d}{n})$ for different values of d, a fixed number of parties $n = 8192$, and a variable number of shares μ. The graphs show the minimum fraction of shares received by any party in the simulations for different number of expected neighbors $\mu \cdot d$. The number of shares is incremented in steps of 3.

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

In Figure 5.1, we plot the redundancy when letting $e = \mu \cdot (1 - \beta)$ 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 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 $d = 5$ (according to Figure 5.5), 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 $\mu = 70$).

We note that in terms of redundancy, this is a significant improvement over the protocol presented in [Liu+22b]. Simulations in [Liu+22b, Figure 4.3] showed that to get a flooding protocol that succeeds 99% of the time for 8192 parties, one would, in their protocol, have to let each party forward the entire message to around 28 neighbors. This will also be their worst-case redundancy; hence, the protocol presented in this work requires less than half the communication complexity for sufficiently long messages. We conclude that ECFlood is not only asymptotically optimal but seems to have advantages over existing byzantine fault-tolerant flooding protocols within the practical range of parameters.

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 5.8. The maximum fraction of shares that all parties in all simulations received in Figure 5.8 corresponds to β in Figure 5.7. 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 e slightly lower than $\mu \cdot (1 - \beta)$.

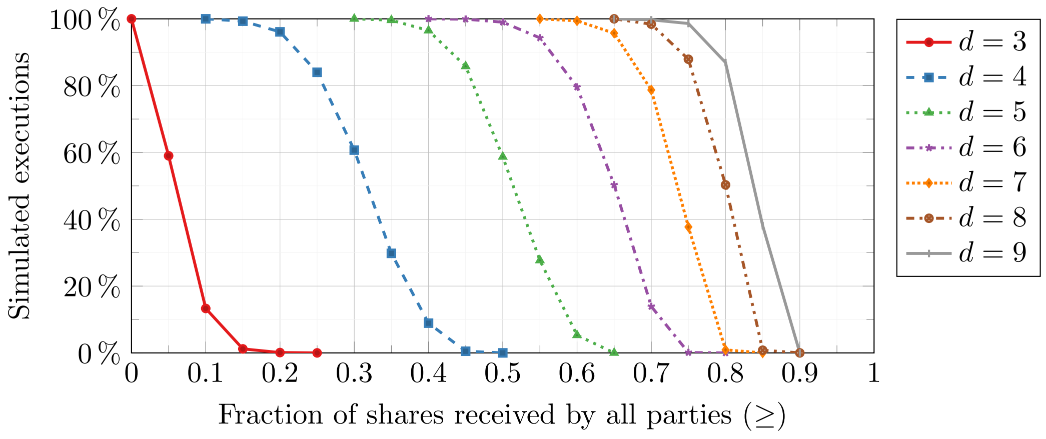


Figure 5.8: Results for simulations of $\mathsf{ECFlood}(\frac{d}{n})$ for different values of d, a fixed number of parties $n = 8192$, and a fixed number of shares $\mu = 20$. The graphs show the percentages of the simulated executions (on the y-axis) where all parties received at least a certain percentage of all the shares sent out by the sender.

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 $\mu = 30$ and a varying number of parties μ and internal parameter d in Figure 5.2. 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 the latency of ERFlood (Figure 5.6). We believe this is because it is a rare event that the maximum latency of ERFlood will occur. Therefore it will become very unlikely that such rare events coincide for shares sent to the same party that receives the minimum number of shares. We note that by adjusting the parameter d, ECFlood can be tuned to achieve a similar latency to that of the protocol from [Liu+22b] while still maintaining a significantly lower redundancy.

As noted in Section 5.3, the protocol ECCast, has a redundancy of only γ^{-1} (which for the parameters of our simulations are only 2) and a latency of just 2. However, to achieve this low redundancy and latency, ECCast requires each party to talk to 8191 neighbors, and the two protocols, therefore, present a tradeoff between low redundancy and diameter and a low number of neighbors.

Bibliography

[Abr+19] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. "Communication Complexity of Byzantine Agreement, Revisited". In: *PODC*. ACM, 2019, pp. 317–326 (cit. on p. 48).

[Abr+20] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. "Sync HotStuff: Simple and Practical Synchronous State Machine Replication". In: *IEEE Symposium on Security and Privacy*. IEEE, 2020, pp. 106–118 (cit. on pp. 5, 103).

[Ala+21] Bithin Alangot, Daniël Reijsbergen, Sarad Venugopalan, Pawel Szalachowski, and Kiat Seng Yeo. "Decentralized and Lightweight Approach to Detect Eclipse Attacks on Proof of Work Blockchains". In: *IEEE Trans. Netw. Serv. Manag.* 18.2 (2021), pp. 1659–1672 (cit. on pp. 111, 112).

[AZV17] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies". In: *IEEE Symposium on Security and Privacy*. IEEE, 2017, pp. 375–392 (cit. on pp. 8, 14, 111).

[Ate+14] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. "Proofs of Space: When Space Is of the Essence". In: *SCN*. Vol. 8642. Lecture Notes in Computer Science. Springer, 2014, pp. 538–557 (cit. on p. 4).

[Bad+20] Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. "Universal Composition with Global Subroutines: Capturing Global Setup Within Plain UC". In: *TCC (3)*. Vol. 12552. Lecture Notes in Computer Science. Springer, 2020, pp. 1–30 (cit. on p. 62).

[Bad+18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. "Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability". In: *CCS*. ACM, 2018, pp. 913–930 (cit. on p. 107).

[Bad+17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. "Bitcoin as a Transaction Ledger: A Composable Treatment". In: *CRYPTO (1)*. Vol. 10401. Lecture Notes in Computer Science. Springer, 2017, pp. 324–356 (cit. on pp. 3, 24, 62).

[BP97] Niko Baric and Birgit Pfitzmann. "Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees". In: *EUROCRYPT*. Vol. 1233. Lecture Notes in Computer Science. Springer, 1997, pp. 480–494 (cit. on p. 151).

[Bau+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)*. Vol. 12698. Lecture Notes in Computer Science. Springer, 2021, pp. 429–459 (cit. on pp. 50, 53, 54, 60, 61, 75).

[BPS18] Anna Ben-Hamou, Yuval Peres, and Justin Salez. “Weighted sampling without replacement”. In: *Brazilian Journal of Probability and Statistics* 32.3 (2018), pp. 657–669 (cit. on pp. 108, 116).

[BM93] Josh Cohen Benaloh and Michael de Mare. “One-Way Accumulators: A Decentralized Alternative to Digital Sinatures (Extended Abstract)”. In: *EUROCRYPT*. Vol. 765. Lecture Notes in Computer Science. Springer, 1993, pp. 274–285 (cit. on p. 151).

[Bha+22] Amey Bhangale, Chen-Da Liu-Zhang, Julian Loss, and Kartik Nayak. “Efficient Adaptively-Secure Byzantine Agreement for Long Messages”. In: *ASIACRYPT*. Lecture Notes in Computer Science 13791 (2022) (cit. on p. 149).

[Bir+99] Kenneth P. Birman, Mark Hayden, Öznur Özkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. “Bimodal Multicast”. In: *ACM Trans. Comput. Syst.* 17.2 (1999), pp. 41–88 (cit. on pp. 7, 54).

[22a] *Bitnodes.io*. `https://bitnodes.io/`. [Online; accessed 16-September-2022]. 2022 (cit. on p. 138).

[BHK20] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*. Cambridge University Press, 2020 (cit. on p. 70).

[Bol01] Béla Bollobás. *Random Graphs*. 2nd ed. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001 (cit. on pp. 52, 132).

[BG17] Vitalik Buterin and Virgil Griffith. *Casper the Friendly Finality Gadget*. 2017 (cit. on pp. 5, 11, 22, 45).

[Can20] Ran Canetti. “Universally Composable Security”. In: *J. ACM* 67.5 (2020), 28:1–28:94 (cit. on pp. 12, 49, 50, 58, 64, 66, 67, 147).

[Can+07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. “Universally Composable Security with Global Setup”. In: *TCC*. Vol. 4392. Lecture Notes in Computer Science. Springer, 2007, pp. 61–85 (cit. on pp. 53, 62).

[Can+17] Ran Canetti, Kyle Hogan, Aanchal Malhotra, and Mayank Varia. “A Universally Composable Treatment of Network Time”. In: *CSF*. IEEE Computer Society, 2017, pp. 360–375 (cit. on p. 53).

[CL99] Miguel Castro and Barbara Liskov. “Practical Byzantine Fault Tolerance”. In: *OSDI*. USENIX Association, 1999, pp. 173–186 (cit. on p. 5).

[Cha+15] Nishanth Chandran, Wutichai Chongchitmate, Juan A. Garay, Shafi Goldwasser, Rafail Ostrovsky, and Vassilis Zikas. “The Hidden Graph Model: Communication Locality and Optimal Resiliency with Adaptive Faults”. In: *ITCS*. ACM, 2015, pp. 153–162 (cit. on pp. 8, 13, 14, 54, 110).

[CGO10] Nishanth Chandran, Juan A. Garay, and Rafail Ostrovsky. “Improved Fault Tolerance and Secure Computation on Sparse Networks”. In: *ICALP (2)*. Vol. 6199. Lecture Notes in Computer Science. Springer, 2010, pp. 249–260 (cit. on p. 110).

[CGO15] Nishanth Chandran, Juan A. Garay, and Rafail Ostrovsky. “Almost-Everywhere Secure Computation with Edge Corruptions”. In: *J. Cryptol.* 28.4 (2015), pp. 745–768 (cit. on p. 110).

[CM19] Jing Chen and Silvio Micali. “Algorand: A secure and efficient distributed ledger”. In: *Theor. Comput. Sci.* 777 (2019), pp. 155–183 (cit. on pp. 4, 103, 105, 106, 114, 148, 151).

[Cor+22] Sandro Coretti, Aggelos Kiayias, Cristopher Moore, and Alexander Russell. “The Generals’ Scuttlebutt: Byzantine-Resilient Gossip Protocols”. In: *CCS*. ACM, 2022, pp. 595–608 (cit. on pp. 9, 15, 55, 112, 149).

[Cri+09] Sérgio Crisóstomo, Udo Schilcher, Christian Bettstetter, and João Barros. “Analysis of Probabilistic Flooding: How Do We Choose the Right Coin?” In: *ICC*. IEEE, 2009, pp. 1–6 (cit. on pp. 7, 54).

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. “Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake”. In: *Financial Cryptography*. Vol. 11598. Lecture Notes in Computer Science. Springer, 2019, pp. 23–41 (cit. on pp. 4, 49, 103, 106, 148).

[Dav+18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. “Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain”. In: *EUROCRYPT (2)*. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 66–98 (cit. on pp. 4, 9, 15, 48, 49, 55, 103, 105, 106, 112, 114, 148, 149, 151).

[Dem+20] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. “Everything is a Race and Nakamoto Always Wins”. In: *CCS*. ACM, 2020, pp. 859–878 (cit. on p. 3).

[Dem+87] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. “Epidemic Algorithms for Replicated Database Maintenance”. In: *PODC*. ACM, 1987, pp. 1–12 (cit. on pp. 7, 54).

[Did09] Frédéric Didier. “Efficient erasure decoding of Reed-Solomon codes”. In: *CoRR* abs/0901.1886 (2009) (cit. on p. 150).

[Din+20] Thomas Dinsdale-Young, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. “Afgjort: A Partially Synchronous Finality Layer for Blockchains”. In: *SCN*. Vol. 12238. Lecture Notes in Computer Science. Springer, 2020, pp. 24–44 (cit. on pp. 5, 11, 22, 45, 103, 105, 114).

[DF11] Benjamin Doerr and Mahmoud Fouz. “Asymptotically Optimal Randomized Rumor Spreading”. In: *ICALP (2)*. Vol. 6756. Lecture Notes in Computer Science. Springer, 2011, pp. 502–513 (cit. on pp. 7, 16).

[DS83] Danny Dolev and H. Raymond Strong. “Authenticated Algorithms for Byzantine Agreement”. In: *SIAM J. Comput.* 12.4 (1983), pp. 656–666 (cit. on p. 110).

[Dwo+88] Cynthia Dwork, David Peleg, Nicholas Pippenger, and Eli Upfal. “Fault Tolerance in Networks of Bounded Degree”. In: *SIAM J. Comput.* 17.5 (1988), pp. 975–988 (cit. on p. 110).

[Dzi+15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. “Proofs of Space”. In: *CRYPTO (2)*. Vol. 9216. Lecture Notes in Computer Science. Springer, 2015, pp. 585–605 (cit. on p. 4).

[ER60] Paul Erdos and Alfred Renyi. “On the evolution of random graphs”. In: *Publ. Math. Inst. Hungary. Acad. Sci.* (1960), pp. 17–61 (cit. on pp. 7, 51).

[22b] *ethernodes.org*. https://ethernodes.org/. [Online; accessed 16-September-2022]. 2022 (cit. on p. 138).

[Eya+16] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. “Bitcoin-NG: A Scalable Blockchain Protocol”. In: *NSDI*. USENIX Association, 2016, pp. 45–59 (cit. on pp. 4, 11).

[Fei+90] Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. “Randomized Broadcast in Networks”. In: *SIGAL International Symposium on Algorithms*. Vol. 450. Lecture Notes in Computer Science. Springer, 1990, pp. 128–137 (cit. on p. 7).

[FF82] Trevor I. Fenner and Alan M. Frieze. “On the connectivity of random m-orientable graphs and digraphs”. In: *Combinatorica* 2.4 (1982), pp. 347–359 (cit. on p. 132).

[FH06] Matthias Fitzi and Martin Hirt. “Optimally efficient multi-valued byzantine agreement”. In: *PODC*. ACM, 2006, pp. 163–168 (cit. on p. 149).

[GP16] Chaya Ganesh and Arpita Patra. “Broadcast Extensions with Optimal Communication and Round Complexity”. In: *PODC*. ACM, 2016, pp. 371–380 (cit. on p. 149).

[GKL20] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. *Full Analysis of Nakamoto Consensus in Bounded-Delay Networks*. Cryptology ePrint Archive, Paper 2020/277. 2020. URL: https://eprint.iacr.org/2020/277 (cit. on p. 3).

[Gar+11] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. “Adaptively secure broadcast, revisited”. In: *PODC*. ACM, 2011, pp. 179–186 (cit. on p. 48).

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. “The Bitcoin Backbone Protocol: Analysis and Applications”. In: *EUROCRYPT (2)*. Vol. 9057. Lecture Notes in Computer Science. Springer, 2015, pp. 281–310 (cit. on pp. 3, 20–27, 35, 45, 48, 103, 106, 114, 151).

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. “The Bitcoin Backbone Protocol with Chains of Variable Difficulty”. In: *CRYPTO (1)*. Vol. 10401. Lecture Notes in Computer Science. Springer, 2017, pp. 291–323 (cit. on pp. 3, 20–22, 25, 48).

[GO08] Juan A. Garay and Rafail Ostrovsky. “Almost-Everywhere Secure Computation”. In: *EUROCRYPT*. Vol. 4965. Lecture Notes in Computer Science. Springer, 2008, pp. 307–323 (cit. on p. 110).

[GRR22] Peter Gazi, Ling Ren, and Alexander Russell. “Practical Settlement Bounds for Proof-of-Work Blockchains”. In: *CCS*. ACM, 2022, pp. 1217–1230 (cit. on p. 3).

[Ger+15] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. “Tampering with the Delivery of Blocks and Transactions in Bitcoin”. In: *CCS*. ACM, 2015, pp. 692–705 (cit. on p. 145).

[Gil+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. “Algorand: Scaling Byzantine Agreements for Cryptocurrencies”. In: *SOSP*. ACM, 2017, pp. 51–68 (cit. on p. 5).

[GTA19] Simon Oddershede Gregersen, Søren Eller Thomsen, and Aslan Askarov. “A Dependently Typed Library for Static Information-Flow Control in Idris”. In: *POST*. Vol. 11426. Lecture Notes in Computer Science. Springer, 2019, pp. 51–75 (cit. on p. 17).

[Gue+19] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. “Scalable Byzantine Reliable Broadcast”. In: *DISC*. Vol. 146. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 22:1–22:16 (cit. on p. 8).

[Gut+15] Daniel Gutiérrez-Reina, Sergio L. Toral, Princy Johnson, and Federico Barrero. “A survey on probabilistic broadcast schemes for wireless ad hoc networks”. In: *Ad Hoc Networks* 25 (2015), pp. 263–292 (cit. on p. 7).

[HHL06] Zygmunt J. Haas, Joseph Y. Halpern, and Li (Erran) Li. “Gossip-based ad hoc routing”. In: *IEEE/ACM Trans. Netw.* 14.3 (2006), pp. 479–491 (cit. on pp. 7, 54).

[HHL88] Sandra Mitchell Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. “A survey of gossiping and broadcasting in communication networks”. In: *Networks* 18.4 (1988), pp. 319–349 (cit. on p. 7).

[Hei+15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. “Eclipse Attacks on Bitcoin’s Peer-to-Peer Network”. In: *USENIX Security Symposium*. USENIX Association, 2015, pp. 129–144 (cit. on pp. 8, 14, 49, 107, 111, 112).

[Hu+12] Ruijing Hu, Julien Sopena, Luciana Arantes, Pierre Sens, and Isabelle M. Demeure. “Fair Comparison of Gossip Algorithms over Large-Scale Random Topologies”. In: *SRDS*. IEEE Computer Society, 2012, pp. 331–340 (cit. on pp. 7, 54).

[JRV20] Siddhartha Jayanti, Srinivasan Raghuraman, and Nikhil Vyas. “Efficient Constructions for Almost-Everywhere Secure Computation”. In: *EUROCRYPT (2)*. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020, pp. 159–183 (cit. on p. 110).

[KYA22] Ioannis Kaklamanis, Lei Yang, and Mohammad Alizadeh. “Poster: Coded Broadcast for Scalable Leader-Based BFT Consensus”. In: *CCS*. ACM, 2022, pp. 3375–3377 (cit. on p. 143).

[Kam+20] Simon Holmgaard Kamp, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, Søren Eller Thomsen, and Daniel Tschudi. *Weight-Based Nakamoto-Style Blockchains*. Cryptology ePrint Archive, Paper 2020/328. 2020. URL: `https://eprint.iacr.org/2020/328` (cit. on pp. 10, 11).

[Kam+21] Simon Holmgaard Kamp, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, Søren Eller Thomsen, and Daniel Tschudi. “Weight-Based Nakamoto-Style Blockchains”. In: *LATINCRYPT*. Vol. 12912. Lecture Notes in Computer Science. Springer, 2021, pp. 299–319 (cit. on p. 10).

[Kam+22] Simon Holmgaard Kamp, Jesper Buus Nielsen, Søren Eller Thomsen, and Daniel Tschudi. *Enig: Player Replaceable Finality Layers with Optimal Validity*. Cryptology ePrint Archive, Paper 2022/201. 2022. URL: `https://eprint.iacr.org/2022/201` (cit. on pp. 5, 11, 17).

[Kar+00] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. “Randomized Rumor Spreading”. In: *FOCS*. IEEE Computer Society, 2000, pp. 565–574 (cit. on pp. 7, 54).

[Kat+13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. “Universally Composable Synchronous Computation”. In: *TCC*. Vol. 7785. Lecture Notes in Computer Science. Springer, 2013, pp. 477–498 (cit. on pp. 53, 62).

[KDG03] David Kempe, Alin Dobra, and Johannes Gehrke. “Gossip-Based Computation of Aggregate Information”. In: *FOCS*. IEEE Computer Society, 2003, pp. 482–491 (cit. on p. 7).

[KKD04] David Kempe, Jon M. Kleinberg, and Alan J. Demers. “Spatial gossip and resource location protocols”. In: *J. ACM* 51.6 (2004), pp. 943–967 (cit. on p. 7).

[KMG03] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. “Probabilistic Reliable Dissemination in Large-Scale Systems”. In: *IEEE Trans. Parallel Distributed Syst.* 14.3 (2003), pp. 248–258 (cit. on pp. 7, 13, 54, 103, 109, 142, 146–149).

[KLS16] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. “Proofs of Proofs of Work with Sublinear Complexity”. In: *Financial Cryptography Workshops*. Vol. 9604. Lecture Notes in Computer Science. Springer, 2016, pp. 61–78 (cit. on p. 23).

[KMZ17] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. *Non-interactive proofs of proof-of-work*. Cryptology ePrint Archive, Report 2017/963. 2017. URL: `https://eprint.iacr.org/2017/963` (cit. on p. 23).

[Kia+17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. “Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol”. In: *CRYPTO (1)*. Vol. 10401. Lecture Notes in Computer Science. Springer, 2017, pp. 357–388 (cit. on pp. 4, 49).

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. "Fair and Robust Multi-party Computation Using a Global Transaction Ledger". In: *EUROCRYPT (2)*. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 705–734 (cit. on p. 53).

[Kin+06] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. "Towards Secure and Scalable Computation in Peer-to-Peer Networks". In: *FOCS*. IEEE, 2006, pp. 87–98 (cit. on p. 110).

[Kok+18] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding". In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2018, pp. 583–598 (cit. on p. 49).

[Kot+07] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. "Zyzzyva: speculative byzantine fault tolerance". In: *SOSP*. ACM, 2007, pp. 45–58 (cit. on p. 5).

[Lam78] Leslie Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System". In: *Commun. ACM* 21.7 (1978), pp. 558–565 (cit. on p. 1).

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pp. 382–401 (cit. on p. 1).

[LSZ15] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. "Inclusive Block Chain Protocols". In: *Financial Cryptography*. Vol. 8975. Lecture Notes in Computer Science. Springer, 2015, pp. 528–547 (cit. on pp. 4, 5, 11).

[Liu+22a] Chen-Da Liu-Zhang, Christian Matt, Ueli Maurer, Guilherme Rito, and Søren Eller Thomsen. *Practical Provably Secure Flooding for Blockchains*. Cryptology ePrint Archive, Paper 2022/608. 2022. URL: `https://eprint.iacr.org/2022/608` (cit. on pp. 14, 15).

[Liu+22b] Chen-Da Liu-Zhang, Christian Matt, Ueli Maurer, Guilherme Rito, and Søren Eller Thomsen. "Practical Provably Secure Flooding for Blockchains". In: *ASIACRYPT*. Vol. 13791. Lecture Notes in Computer Science. Springer, 2022, pp. 774–805 (cit. on pp. 14, 54, 142, 144–146, 148, 149, 151, 155–157, 167, 168, 174, 175).

[LMT22] Chen-Da Liu-Zhang, Christian Matt, and Søren Eller Thomsen. *Asymptotically Optimal Message Dissemination with Applications to Blockchains*. Cryptology ePrint Archive, Paper 2022/1723. 2022. URL: `https://eprint.iacr.org/2022/1723` (cit. on pp. 15, 16).

[Lon+21] Teng Long, Shan Qu, Qi Li, Huquan Kang, Luoyi Fu, Xinbing Wang, and Chenghu Zhou. "Efficient Block Propagation in Wireless Blockchain Networks and Its Application in Bitcoin". In: *IEEE Trans. Netw. Sci. Eng.* 8.4 (2021), pp. 3349–3368 (cit. on p. 8).

[Luu+16] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. "A Secure Sharding Protocol For Open Blockchains". In: *CCS*. ACM, 2016, pp. 17–30 (cit. on p. 49).

[MMR99] Dahlia Malkhi, Yishay Mansour, and Michael K. Reiter. “On Diffusing Updates in a Byzantine Environment”. In: *SRDS*. IEEE, 1999, pp. 134–143 (cit. on pp. 8, 110).

[MPS01] Dahlia Malkhi, Elan Pavlov, and Yaron Sella. “Optimal Unconditional Information Diffusion”. In: *DISC*. Vol. 2180. Lecture Notes in Computer Science. Springer, 2001, pp. 63–77 (cit. on pp. 8, 110).

[Mao+20] Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. “Perigee: Efficient Peer-to-Peer Network Design for Blockchains”. In: *PODC*. ACM, 2020, pp. 428–437 (cit. on p. 8).

[MHG18] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. “Low-Resource Eclipse Attacks on Ethereum’s Peer-to-Peer Network”. In: (2018). URL: https://eprint.iacr.org/2018/236 (cit. on pp. 8, 14, 49, 107, 111).

[MNT22a] Christian Matt, Jesper Buus Nielsen, and Søren Eller Thomsen. *Formalizing Delayed Adaptive Corruptions and the Security of Flooding Networks*. Cryptology ePrint Archive, Paper 2022/010. 2022. URL: https://eprint.iacr.org/2022/010 (cit. on pp. 12, 13).

[MNT22b] Christian Matt, Jesper Buus Nielsen, and Søren Eller Thomsen. “Formalizing Delayed Adaptive Corruptions and the Security of Flooding Networks”. In: *CRYPTO (2)*. Vol. 13508. Lecture Notes in Computer Science. Springer, 2022, pp. 400–430 (cit. on pp. 12, 15, 103, 106, 107, 110, 113–115, 122, 123, 142, 144, 146–149, 155, 156, 159).

[MM02] Petar Maymounkov and David Mazières. “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric”. In: *IPTPS*. Vol. 2429. Lecture Notes in Computer Science. Springer, 2002, pp. 53–65 (cit. on p. 54).

[Mer89] Ralph C. Merkle. “A Certified Digital Signature”. In: *CRYPTO*. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 218–238 (cit. on p. 151).

[Mil+16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. “The Honey Badger of BFT Protocols”. In: *CCS*. ACM, 2016, pp. 31–42 (cit. on p. 5).

[MS03] Yaron Minsky and Fred B. Schneider. “Tolerating malicious gossip”. In: *Distributed Comput.* 16.1 (2003), pp. 49–68 (cit. on pp. 8, 110).

[Nak08] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. manuscript. 2008. URL: http://www.bitcoin.org/bitcoin.pdf (cit. on pp. 2, 20, 24, 25, 48, 103, 142, 148).

[NC17] Arvind Narayanan and Jeremy Clark. “Bitcoin’s academic pedigree”. In: *Commun. ACM* 60.12 (2017), pp. 36–45 (cit. on p. 2).

[Nay+16] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. “Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack”. In: *EuroS&P*. IEEE, 2016, pp. 305–320 (cit. on p. 112).

[Nay+20] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. “Improved Extension Protocols for Byzantine Broadcast and Agreement”. In: *DISC*. 2020 (cit. on p. 149).

[Ngu05] Lan Nguyen. “Accumulators from Bilinear Pairings and Applications”. In: *CT-RSA*. Vol. 3376. Lecture Notes in Computer Science. Springer, 2005, pp. 275–292 (cit. on p. 151).

[Nie03] Jesper Buus Nielsen. “On protocol security in the cryptographic model”. PhD thesis. Aarhus University, 2003 (cit. on p. 78).

[Niu+19] Jianyu Niu, Chen Feng, Hoang Dau, Yu-Chih Huang, and Jingge Zhu. *Analysis of Nakamoto Consensus, Revisited*. Cryptology ePrint Archive, Report 2019/1225. 2019. URL: `https://eprint.iacr.org/2019/1225` (cit. on pp. 3, 20, 26, 27).

[Özç+21] Ilker Özçelik, Sai Medury, Justin T. Broaddus, and Anthony Skjellum. “An Overview of Cryptographic Accumulators”. In: *ICISSP*. SCITEPRESS, 2021, pp. 661–669 (cit. on p. 151).

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. “Analysis of the Blockchain Protocol in Asynchronous Networks”. In: *EUROCRYPT (2)*. Vol. 10211. Lecture Notes in Computer Science. 2017, pp. 643–673 (cit. on pp. 3, 20, 22–24, 26, 27, 48).

[PS17a] Rafael Pass and Elaine Shi. “FruitChains: A Fair Blockchain”. In: *PODC*. ACM, 2017, pp. 315–324 (cit. on pp. 4, 5, 11, 103, 106, 114, 151).

[PS17b] Rafael Pass and Elaine Shi. “Hybrid Consensus: Efficient Consensus in the Permissionless Model”. In: *DISC*. Vol. 91. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 39:1–39:16 (cit. on pp. 13, 23, 49, 50, 53, 106, 110, 148).

[PS18a] Rafael Pass and Elaine Shi. “Thunderella: Blockchains with Optimistic Instant Confirmation”. In: *EUROCRYPT (2)*. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 3–33 (cit. on p. 5).

[PS18b] Rafael Pass and Elaine Shi. “Thunderella: Blockchains with Optimistic Instant Confirmation”. In: *EUROCRYPT (2)*. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 3–33 (cit. on pp. 11, 21, 23, 103, 106, 114, 148, 151).

[RS60] Irving S. Reed and Gustave Solomon. “Polynomial Codes Over Certain Finite Fields”. In: *Journal of The Society for Industrial and Applied Mathematics* 8 (1960), pp. 300–304 (cit. on p. 150).

[Ren19] Ling Ren. “Analysis of Nakamoto Consensus”. In: *IACR Cryptol. ePrint Arch.* (2019), p. 943 (cit. on pp. 3, 20, 22, 23, 26, 27, 29, 32, 48).

[RT19] Elias Rohrer and Florian Tschorsch. “Kadcast: A Structured Approach to Broadcast in Blockchain Networks”. In: *AFT*. ACM, 2019, pp. 199–213 (cit. on pp. 8, 16, 54, 103, 110, 149).

[RT21] Elias Rohrer and Florian Tschorsch. *Kadcast-NG: A Structured Broadcast Protocol for Blockchain Networks*. Cryptology ePrint Archive, Report 2021/996. 2021. URL: `https://ia.cr/2021/996` (cit. on p. 8).

[Sal+22] Gökay Saldamli, Charit Upadhyay, Devika Jadhav, Rohit Shrishrimal, Bapugouda Patil, and Lo’ai Tawalbeh. “Improved gossip protocol for blockchain applications”. In: *Clust. Comput.* 25.3 (2022), pp. 1915–1926 (cit. on p. 8).

[SOA16] Muntadher Fadhil Sallal, Gareth Owenson, and Mo Adda. "A Bitcoin Model for Evaluation of Clustering to Improve Propagation Delay in Bitcoin Network". In: *CSE/EUC/DCABES*. IEEE Computer Society, 2016, pp. 468–475 (cit. on p. 149).

[SCS03] Yoav Sasson, David Cavin, and André Schiper. "Probabilistic broadcast for flooding in wireless mobile ad hoc networks". In: *WCNC*. IEEE, 2003, pp. 1124–1130 (cit. on p. 7).

[Shr+20] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. "On the Optimality of Optimistic Responsiveness". In: *CCS*. ACM, 2020, pp. 839–857 (cit. on p. 23).

[SZ15] Yonatan Sompolinsky and Aviv Zohar. "Secure High-Rate Transaction Processing in Bitcoin". In: *Financial Cryptography*. Vol. 8975. Lecture Notes in Computer Science. Springer, 2015, pp. 507–527 (cit. on pp. 4, 5, 11).

[SK20] Alistair Stewart and Eleftherios Kokoris-Kogia. *GRANDPA: a Byzantine Finality Gadget*. 2020 (cit. on pp. 5, 11, 22, 45).

[TS21] Søren Eller Thomsen and Bas Spitters. "Formalizing Nakamoto-Style Proof of Stake". In: *Computer Security Foundations*. IEEE, 2021, pp. 1–15 (cit. on p. 17).

[Tra+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*. IEEE, 2020, pp. 894–909 (cit. on pp. 8, 14, 111).

[TLP22] Georgios Tsimos, Julian Loss, and Charalampos Papamanthou. "Gossiping for Communication-Efficient Broadcast". In: *CRYPTO (3)*. Vol. 13509. Lecture Notes in Computer Science. Springer, 2022, pp. 439–469 (cit. on pp. 8, 110).

[TC84] Russell Turpin and Brian A. Coan. "Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement". In: *Inf. Process. Lett.* 18.2 (1984), pp. 73–76 (cit. on p. 149).

[Upf94] Eli Upfal. "Tolerating a Linear Number of Faults in Networks of Bounded Degree". In: *Inf. Comput.* 115.2 (1994), pp. 312–320 (cit. on p. 110).

[VT19] Huy Vu and Hitesh Tewari. "An Efficient Peer-to-Peer Bitcoin Protocol with Probabilistic Flooding". In: *Emerging Technologies in Computing*. Ed. by Mahdi H. Miraz, Peter S. Excell, Andrew Ware, Safeeullah Soomro, and Maaruf Ali. Cham: Springer, 2019, pp. 29–45 (cit. on pp. 8, 149).

[Vyz+20] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras. *GossipSub: Attack-Resilient Message Propagation in the Filecoin and ETH2.0 Networks*. 2020 (cit. on p. 8).

[Wan+22] Xin Wang, Xin Jiang, Yanxiu Liu, Jiaping Wang, and Yi Sun. "Data Propagation for Low Latency Blockchain Systems". In: *IEEE J. Sel. Areas Commun.* 40.12 (2022), pp. 3631–3644 (cit. on p. 8).

[Woo+14] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32 (cit. on pp. 142, 148).

[Xu+20] Guangquan Xu, Bingjiang Guo, Chunhua Su, Xi Zheng, Kaitai Liang, Duncan S. Wong, and Hao Wang. "Am I eclipsed? A smart detector of eclipse attacks for Ethereum". In: *Comput. Secur.* 88 (2020) (cit. on p. 111).

[YM13] Osman Yagan and Armand M. Makowski. "On the scalability of the random pairwise key predistribution scheme: Gradual deployment and key ring sizes". In: *Perform. Evaluation* 70.7-8 (2013), pp. 493–512 (cit. on pp. 132, 133).

[Yin+19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. "HotStuff: BFT Consensus with Linearity and Responsiveness". In: *PODC*. ACM, 2019, pp. 347–356 (cit. on pp. 5, 143, 151).

[ZMR18] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. "RapidChain: Scaling Blockchain via Full Sharding". In: *CCS*. ACM, 2018, pp. 931–948 (cit. on p. 49).

[ZL19] Shijie Zhang and Jong-Hyouk Lee. "Eclipse-based Stake-Bleeding Attacks in PoS Blockchain Systems". In: *BSCI*. ACM, 2019, pp. 67–72 (cit. on p. 112).

[ZTA21] Haofan Zheng, Tuan Tran, and Owen Arden. "Total Eclipse of the Enclave: Detecting Eclipse Attacks From Inside TEEs". In: *IEEE ICBC*. IEEE, 2021, pp. 1–5 (cit. on p. 111).