Formalizing Delayed Adaptive Corruptions and the Security of Flooding Networks

Christian Matt[1], Jesper Buus Nielsen*[2], and Søren Eller Thomsen[2]

[1]Concordium, Zurich, Switzerland
cm@concordium.com
[2]Concordium Blockchain Research Center, Aarhus University, Denmark
{jbn, sethomsen}@cs.au.dk

February 24, 2022

Abstract

Many decentralized systems rely on flooding protocols for message dissemination. In such a protocol, the sender of a message sends it to a randomly selected set of peers. These peers again send the message to their randomly selected peers, until every network participant has received the message. This type of protocols clearly fail in face of an adaptive adversary who can simply corrupt all peers of the sender and thereby prevent the message from being delivered. Nevertheless, flooding protocols are commonly used within protocols that aim to be cryptographically secure, most notably in blockchain protocols. While it is possible to revert to static corruptions, this gives unsatisfactory security guarantees, especially in the setting of a blockchain that is supposed to run for an extended period of time.

To be able to provide meaningful security guarantees in such settings, we give precise semantics to what we call δ-*delayed adversaries* in the Universal Composability (UC) framework. Such adversaries can adaptively corrupt parties, but there is a delay of time δ from when an adversary decides to corrupt a party until they succeed in overtaking control of the party. Within this model, we formally prove the intuitive result that flooding protocols are secure against δ-delayed adversaries when δ is at least the time it takes to send a message from one peer to another plus the time it takes the recipient to resend the message. To this end, we show how to reduce the adaptive setting with a δ-delayed adversary to a static experiment with an Erdős–Rényi graph. Using the established theory of Erdős–Rényi graphs, we provide upper bounds on the propagation time of the flooding functionality for different neighborhood sizes of the gossip network. More concretely, we show the following for security parameter κ, point-to-point channels with delay at most Δ, and n parties in total, with a sufficiently delayed adversary that can corrupt any constant fraction of the parties: If all parties send to $\Omega(\kappa)$ parties on average, then we can realize a flooding functionality with maximal delay $\mathcal{O}\big(\Delta \cdot \log(n)\big)$; and if all parties send to $\Omega\big(\sqrt{\kappa n}\big)$ parties on average, we can realize a flooding functionality with maximal delay $\mathcal{O}(\Delta)$.

*Partially funded by The Concordium Foundation; The Danish Independent Research Council under Grant-ID DFF-8021-00366B (BETHE); The Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

Contents

1 Introduction

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 variation of NSBs that security can be guaranteed if a majority of the puzzles are solved by honest parties and if blocks can be propagated fast enough to ensure with 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* [ACD+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) [GKKZ11] is assumed.

Indeed, several papers [GKL15, GKL17, PSs17] have proven the security of Bitcoin's consensus against adaptive corruptions, and Ouroboros Praos [DGKR18] 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 [Can01]. Using this model, one can quantify the minimum amount of time δ it takes from 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 describe exactly what kind of adversaries different P2P networks and protocols build 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 machine 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 our paper, delayed adversaries have much broader applications and were in fact already used in

other works, with varying degree of formality. For example, the original Ouroboros [KRDO17], in contrast to its successor Ouroboros Praos [DGKR18], 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 [PS17], 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 [LNZ+16, KJG+18, 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. Setting the difficulty based on a too optimistic assumption on the delay jeopardizes the security of the system, and setting it based on a too loose upper bound degrades efficiency. Knowing a tight bound on the propagation delay is thus key 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 [HKZG15, 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 simply 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 nodes. Common terminology includes "broadcast", "flood" and "multicast". In this paper, 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.

1.2 Contributions and Results

Our contributions are twofold:

1. We give precise semantics to δ-*delayed corruptions* (introduced in [PS17] as δ-agile corruptions) within the UC framework [Can20]. We define the semantics via corruption shells which allows us to prove how this type of 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 exactly 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 which uses at least δ time to perform a corruption. We define this notion precisely within the

UC framework using the notion of time from [BDD+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 for 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 1). Furthermore, we show that if a protocol is secure against a "fast" adversary, then this implies that it is also secure against a "slow" adversary (Theorem 2). 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 realise our flooding functionality with both a naive protocol, $\pi_{\mathsf{NaiveFlood}}$, where everybody simply sends to everybody, and a more advanced protocol, $\pi_{\mathsf{ERFlood}}(\rho)$, where all parties choose to send to other parties with probability ρ.

In order to realise 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 $\sigma = 0$ then 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: For $\rho \coloneqq \sqrt{\frac{\kappa}{h}}$, the guaranteed delivery time is $\Delta' \coloneqq 2 \cdot \Delta$.

Instantiation 2: For $\rho \coloneqq \frac{\kappa}{h}$, the guaranteed delivery time is $\Delta' \coloneqq \Delta \cdot \left(5 \log\left(\frac{n}{2\kappa}\right) + 2\right)$.

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 1. Furthermore, standard probability bounds ensure that each instantiation has a neighborhood of $\mathcal{O}(n \cdot \rho)$ with high probability.

Outline of the paper. In the remainder of this section, we review some selected related work and provide a high-level overview of our techniques used to prove our technical results. In Section 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, we introduce our new model for δ-delayed adversaries, in Section 4 we prove concrete bounds for the diameter of Erdős–Rényi graphs, and in Section 5 present our ideal functionalities. Finally, in Section 6 we present our two implementations of the flooding functionality, and in Section 7 we conclude and provide directions for future work.

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 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 towards infinity is 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 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 presence of a *static* adversary. However, for an adaptive adversary that is 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 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 rules of the game is 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 but with undirected edges, and do a couple of simple gamehops 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 additional 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 for this game. However, the adversary can still dynamically control the size of the graph that is returned. At first, this may seem innocent, but in fact, 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 game, which is similar to the other, except that the oracle now at random fixes the size of the graph 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 which specifically embeds an Erdős–Rényi graph of the fixed size. This allows us to apply results bounding diameter of Erdős–Rényi graphs to bound the probability that that a message is not delivered timely.

1.4 Related Work

Hybrid consensus. Hybrid Consensus [PS17] is a 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 simply corrupt the majority of its members to break the security of the protocol. 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 essentially 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 to that, 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 has been several suggestions for modelling time in UC. [KMTZ13] models time using a clock functionality that is local to each protocol. This functionality synchronizes the parties by only allowing the adversary to advance time when all parties have reported that they have been activated. As this is a *local* functionality, other ideal functionalities has 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 [CDPW07]. 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 modelled with this a synchronous protocol.

A different approach is taken in [BDD+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 modelled without having synchrony as an inherent assumption. The specific timing-assumptions can then be captured by adding an extra ideal functionality which exposes relevant information to the parties.

In this work we use the modelling of time from [BDD+21]. This allows us to model general timing assumptions on the capabilities of adversary without tying our modelling 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. [DGH+87], and have been studied extensively since then, see e.g., [BHÖ+99, KSSV00, HHL06]. This line of work has been motivated by how epidemics and rumors spread among a population. There are many different variations of such protocols with different properties. Most closely related to our flooding protocols is what Demers et al. refer to as the blind counter model, in which all parties send to a fixed number of randomly chosen neighbors. More advanced variations keep sending to new random peers until a certain number of recipients replied that they already knew the message. This line of work considers only random failures and not adaptive corruptions of a malicious adversary. Hence, while some of the protocols are applicable to our setting, their analysis is not.

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 already received the message from other peers. They instead propose a structured network based on Kademlia [MM02], in which every node has $\mathcal{O}(\log n)$ neighbors and the diameter of the graph is also $\mathcal{O}(\log n)$. Additionally, their protocol includes a parameter for controlling the redundancy and thus the resistance to attacks. Due to the structured nature, the suggested network is, however, not secure against adaptive corruptions of any kind.

The hidden graph model. Chandran et al. [CCG+15a] 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 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. To be secure against adaptive corruptions, they however need to assume that the communication graph between honest parties remains hidden, i.e., they allow honest honest parties to communicate securely without an adversary learning who is communicating with whom. Furthermore, they only prove very loose bounds on the locality and diameter of the obtained graph by showing that both are poly-logarithmic. In Appendix A, we replicate this result but with concrete bounds.

2 Preliminaries

2.1 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. In our proofs we will use the acronyms LHS and RHS to refer to respectively the left-hand side and the right-hand side of an equality.

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 $n := |\mathcal{P}|$ to denote the total number of parties in a protocol (except when we state and prove general results about graphs) and $h := n - t$ 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.

2.2 Graphs

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

2.2.1 Probabilistic Bounds

For completeness we record the Chernoff bound which states that the sum of independent random variables concentrate around their mean.

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

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

2.2.2 Basic Definitions

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

Definition 1 (Undirected graph)**.** A 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 being 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 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

$$\begin{aligned} \theta(v, 0) &\triangleq \{v\} \\ \theta(v, i+1) &\triangleq \Gamma^{i+1}(v) \setminus \Gamma^i(v). \end{aligned}$$

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

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

Remark 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 (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 being 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 a directed graph as well as each time we use the function it will be clear from the context if the graph is directed or undirected.

Definition 4 (Neighbors for directed graphs)**.** For a digraph $G = (\mathcal{V}, E)$ we define the outgoing neighbourhood 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 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 d such that $v_2 \in \Gamma^d(v_1)$. If no such d 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 6 (Distance from node)**.** Let G be a directed or undirected graph, v be a node in the graph and $d \in \mathbb{N}$ be a distance. The property $\phi_{\text{Dist}}(v, G, d)$ decides if all nodes in G are within distance d of v. Formally,

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

Definition 7 (Diameter)**.** Let $G = (\mathcal{V}, E)$ be a graph. We define the diameter of a graph to be the smallest $d \in \mathbb{N}$ s.t. $\forall v_1, v_2 \in \mathcal{V}$ we have that $\texttt{dist}(v_1, v_2) \leq d$. We define the property that a graph has diameter at most d as

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

2.2.3 Erdős–Rényi Graphs

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

Definition 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 $G \xleftarrow{\$} \mathbb{G}(n, \rho)$.

Remark 2 (Expected degree). If $G = (\mathcal{V}, E) \xleftarrow{\$} \mathbb{G}(n, p)$ and $p = \frac{d}{n}$, then the expected degree of each node is d, i.e., for all $v \in \mathcal{V}$,

$$\mathbb{E}[\texttt{deg}(v)] = d. \tag{1}$$

Following from Chernoff Lemma 1, the degree of all nodes in Erdős–Rényi graphs concentrates around the expected value.

Lemma 2 (Maximum degree of Erdős–Rényi Graph)**.** *Let $n, d \in \mathbb{N}$, $\rho \coloneqq \frac{d}{n}$, $G = (\mathcal{V}, E) \xleftarrow{\$} \mathbb{G}(n, p)$. For any $\delta \in [0, 1]$ we have*

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

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

$$\mathbb{E}[X_{v,u}] = \rho, \tag{3}$$

which again implies that

$$\mathbb{E}[\texttt{deg}(v)] = \sum_{u \in \mathcal{V}} \mathbb{E}[X_{v,u}] = n \cdot \phi = d. \tag{4}$$

Chernoff (Lemma 1) now implies that

$$\Pr[\texttt{deg}(v) \geq (1+\delta)d] \leq e^{-\frac{\delta^2 d}{3}}. \tag{5}$$

By a union bound we get that

$$\Pr[\max_{v \in \mathcal{V}} \texttt{deg}(v) \geq (1+\delta)d] \leq \sum_{v \in \mathcal{V}} \Pr[\texttt{deg}(v) \geq (1+\delta)d] \leq ne^{-\frac{\delta^2 d}{3}}. \tag{6}$$

□

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 modelling of delayed adversaries (Section 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 that is 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 9 (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 $\mathcal{A}$.

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

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 10 (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 to independent ideal functionalities as being in the "ideal world".

2.3.2 Corruptions

The UC-framework has no built-in semantics for 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. 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 common 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 necessarily needs to be *corruption preserving*. That is, it should not require more corruptions to attack the ideal functionality than what it takes to attack the real protocol. In order to ensure this, an additional Interactive Turing Machine Instance (ITI) called the *corruption aggregation* ITI is run aside the parties in protocol. Whenever a party is corrupted, it registers as corrupted by the corruption aggregation ITI. The environment can then query the corruption aggregation ITI in order 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 modelling purposes and thus not present when deploying a protocol. In that way, if the simulator corrupts differently than the adversary, the environment is immediately able to distinguish. A depiction of the flow of information about corruptions can be found in Figure 1.

Identity masking function and PIDs. The UC-framework allows for a 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).

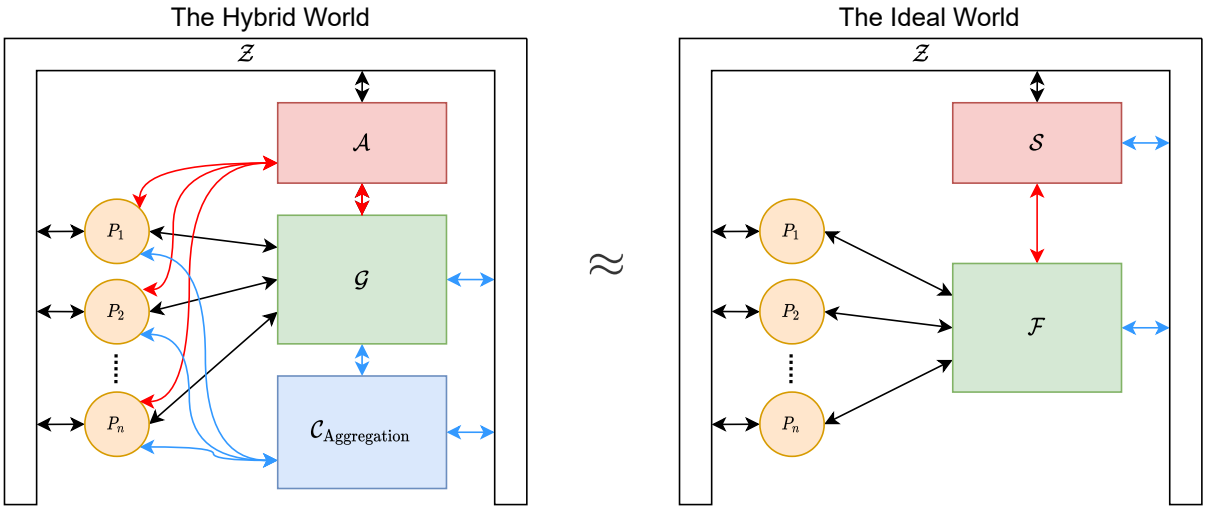

Figure 1: A depiction of how corruptions are propagated in both the hybrid world and the ideal world. Corruption requests are passed along the red arrows, and corruption information is propagated along the blue arrows. In the hybrid world the adversary can corrupt either a party $(p_1, \ldots, p_n)$ 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 merge this information with its own information, before responding the environment with the aggregated information. In order for this behaviour 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.

Standard corruption models within UC. Canetti presents among others how *adaptive corruptions* and *static corruptions* can be modelled within UC. A brief recap of this modelling 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.

2.3.3 Time

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

In TARDIS time is modelled 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 that 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 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 in that we do not have raw access to time, only clocks. The level of information functionalities provide to parties about time is what determines possible assumptions about synchrony.

For completeness, the ticker functionality as described in TARDIS is referenced below.

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, $\mathcal{P} = F = L_P = L_F \coloneqq \varnothing$.

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 $L_P \coloneqq \varnothing$ and $L_F \coloneqq \varnothing$ 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 paper, we adopt the convention (also used in [BDD+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 that is described by **Tick**. We furthermore adopt the convention that for brevity we leave out registration of functionalities and parties by the global ticker. All of the functionalities and protocols we consider will upon initialization as the first thing register 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 figure out 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 and thereby prevent 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 that parties activate the respective functionalities before passing (*Activated*) to the ticker. Intuitively, this corresponds to that if a protocol is expected to work correctly, then sub-protocols that depend upon 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 simply 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 simply 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. Technically, the ticker functionality in TARDIS is defined within the GUC framework [CDPW07]. However, as pointed out in [BCH+20], the GUC framework has not been updated since its introduction, even though that it relies on the UC framework which has been revised and updated several times since. Furthermore, [BCH+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, [BCH+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. Additionally, they show how examples of global functionalities that instead can be modelled as global subroutines. One of their examples [BCH+20, Section 4.2] of such a transformation is, that they show that [BMTZ17] that implements a transaction ledger using a global clock (similar to the one from [KMTZ13]), instead could have been done directly within UC, by modelling 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 [BCH+20, Section 4.2] (in particular can adopt the same *identity bound* for the environment to ensure that the ticker works as expected) to keep our modelling within plain UC.

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, it takes at least δ time to execute a corruption. 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 and 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 that are 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 precise description of how this corruption model can be captured within the UC-framework.

3.1 The δ-delay Shell

It is tedious and error-prone to include code that models corruption behavior in each protocol description and ideal functionality description. We therefore separate the concern of describing corruption behaviors to that of describing the protocol, by introducing a precise name for protocol transformers, dubbed *shells*, which exactly 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 keep track of when *Precorrupt* tokens are delivered and only accept corruption tokens for a particular party δ time later. Having two different shells is, however, necessary as the protocol shell needs to wrap the individual ITMs actually executing the protocol, whereas the ideal shell needs to wrap only the ITM running the ideal functionality.

Additionally, both shells allow the first message that is 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 additionally letting the $\mathcal{D}_{\mathsf{Real}}$ spawn a *corruption-clock* (written $\mathcal{F}_{\text{CorruptionClock}}$) which exactly 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. The corruption clock is therefore only an artifact of our modelling and will not appear when actually running the protocol.

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

The functionality maintains a counter $\texttt{Time}$. Initially, $\texttt{Time} := 0$.

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

Tick: It updates $\texttt{Time} := \texttt{Time} + 1$.

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

We amend the corruption aggregation ITI presented in [Can20] to also make information about the precorruptions an adversary have 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 in order to ensure that the simulator appended to the ideal functionalities can actually gain functionality-specific powers when performing a corruption. For example it might be that a certain channel does not need to respect delivery guarantees when the sender gets corrupted (for an example of this see Section 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 $\texttt{PrecorruptionTime}_i$. Initially, $\texttt{PrecorruptionTime}_i := \bot$. When receiving precorruptions and corruptions the wrapper has the behaviour 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 (*Initialize*, τ) as the first message, then the party updates $\texttt{PrecorruptionTime}_i := \tau$ and if $\tau \neq \bot$ then also notifies the corruption-aggregation ITI.

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

Corruption: When p_i receives *Corrupt* at time τ, then p_i checks if $\texttt{PrecorruptionTime}_i + \delta \leq \tau$. If that is not the case the request is ignored. Otherwise the party first notifies the corruption-aggregation ITI by sending (*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 furthermore 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 subroutine for the first time, it sends (*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 $\mathit{CorruptionStatus}$ 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 which 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 initially is instantiated to be 0. When receiving precorruptions, corruptions and corruption-status requests it has the following behaviour and on all other inputs it forwards the inputs to $\mathcal{F}$.

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

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

Inputs: If the functionality receives (Input, p, v) from the adversary and $p \in \texttt{Corrupted}$, 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 $\texttt{Time} := \texttt{Time} + 1$.

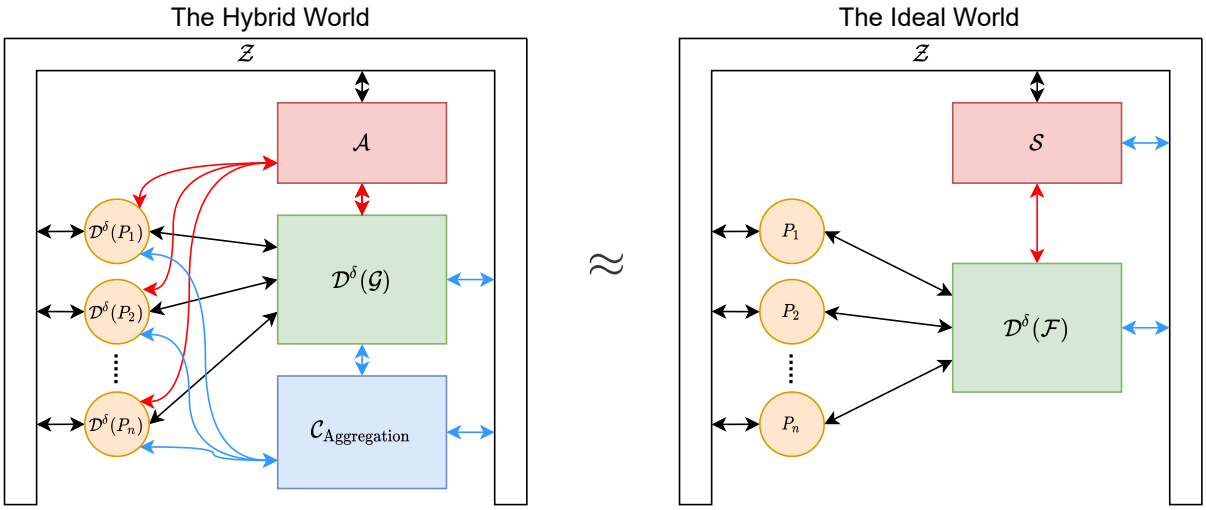

Figure 2: A depiction of the security statement for a protocol that implements an ideal functionality $\mathcal{F}$ using the functionality $\mathcal{G}$ against a δ-delayed adversary.

We next formally define what it means for a protocol to securely implement a functionality against a δ-delayed adversary, see also Figure 2 for a graphical depiction.

Definition 11 (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

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

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 that is 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 as defined in UC. To be able to precisely 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].

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

The shell adds the following behaviour 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 *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 furthermore 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 $\texttt{Corrupted} := p :: \texttt{Corrupted}$.

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`, which initially is 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 $\texttt{Corrupted} := p :: \texttt{Corrupted}$ and returns to the adversary all the values received from p and output to p so far. From now on inputs from p are ignored but are instead given via the backdoor tape by the adversary. Furthermore it propagates $(\mathit{Corrupt}, p)$ to $\mathcal{F}$.

Inputs: If the functionality receives (Input, p, v) from the adversary and $p \in \texttt{Corrupted}$ 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.

We believe that 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.

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

Theorem 1. *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,

$$\begin{aligned} &\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z}, \texttt{EXEC}(\mathcal{Z}, \mathcal{A}, \mathcal{B}_{\mathsf{Real}}(\pi)) \approx \texttt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{B}_{\mathsf{Ideal}}(\mathcal{F})) \\ &\iff \forall \mathcal{A}'\ \exists \mathcal{S}' \forall \mathcal{Z}', \texttt{EXEC}(\mathcal{Z}', \mathcal{A}', \mathcal{D}^0_{\mathsf{Real}}(\pi)) \approx \texttt{EXEC}(\mathcal{Z}', \mathcal{S}', \mathcal{D}^0_{\mathsf{Ideal}}(\mathcal{F})). \end{aligned} \tag{7}$$

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 $\mathcal{A}'$ 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 (7) to obtain a simulator $\mathcal{S}$ s.t.

$$\forall \mathcal{Z}, \texttt{EXEC}(\mathcal{Z}, \mathcal{A}, \mathcal{B}_{\mathsf{Real}}(\pi)) \approx \texttt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{B}_{\mathsf{Ideal}}(\mathcal{F})). \tag{8}$$

Given $\mathcal{S}$ we construct $\mathcal{S}'$ by running $\mathcal{S}$ inside $\mathcal{S}'$. Each time $\mathcal{S}$ outputs $(\textit{Precorrupt}, p_i)$ to the environment then $\mathcal{S}'$ outputs $(\textit{Precorrupt}, p_i)$ to $\mathcal{D}^0_{\mathsf{Ideal}}(\mathcal{F})$. All other inputs and outputs are forwarded to and from $\mathcal{S}$ directly. 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.,

$$\texttt{EXEC}(\mathcal{Z}', \mathcal{A}', \mathcal{D}^0_{\mathsf{Real}}(\pi)) \not\approx \texttt{EXEC}(\mathcal{Z}', \mathcal{S}', \mathcal{D}^0_{\mathsf{Ideal}}(\mathcal{F})) \tag{9}$$

Let us now show how to construct an environment, $\mathcal{Z}$, that can distinguish for the byzantine setting and thereby contradict Equation (8).

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

- Whenever a *CorruptionStatus* command is issued by $\mathcal{Z}'$ to the corruption aggregation ITI, we amend the answer with an additional list of precorruptions which we have received from $\mathcal{A}$ so far.
- Whenever a $(\textit{Initialize}, \tau)$ command is send to some party it is not forwarded by $\mathcal{Z}$ but instead recorded as a precorruption of this party. This does not change the behavior of the protocol nor the ideal functionality as these are ignored.

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

We observe that

$$\texttt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{B}_{\mathsf{Ideal}}(\mathcal{F})) \approx \texttt{EXEC}(\mathcal{Z}', \mathcal{S}', \mathcal{D}^0_{\mathsf{Ideal}}(\mathcal{F})), \tag{10}$$

and

$$\texttt{EXEC}(\mathcal{Z}, \mathcal{A}, \mathcal{B}_{\mathsf{Real}}(\pi)) \approx \texttt{EXEC}(\mathcal{Z}', \mathcal{A}', \mathcal{D}^0_{\mathsf{Real}}(\pi)). \tag{11}$$

Together with Equation (9) this contradicts Equation (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 1 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 is able to distinguish, build an environment $\mathcal{Z}'$ using this (removing *Precorrupt*-tokens and initializations), and derive a contradiction similarly 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 that is 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 allows us to capture this intuition in the lemma below.

Theorem 2 (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,

$$\begin{aligned}&\forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{Z}, \texttt{EXEC}(\mathcal{Z}, \mathcal{A}, \mathcal{D}^{\delta}_{\mathsf{Real}}(\pi)) \approx \texttt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{D}^{\delta}_{\mathsf{Ideal}}(\mathcal{F}))\\ &\implies \forall \mathcal{A}', \exists \mathcal{S}', \forall \mathcal{Z}', \texttt{EXEC}(\mathcal{Z}', \mathcal{A}', \mathcal{D}^{\delta'}_{\mathsf{Real}}(\pi)) \approx \texttt{EXEC}(\mathcal{Z}', \mathcal{S}', \mathcal{D}^{\delta'}_{\mathsf{Ideal}}(\mathcal{F})).\end{aligned} \tag{12}$$

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

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

$$\forall \mathcal{Z}, \texttt{EXEC}(\mathcal{Z}, \mathsf{Filter}(\mathcal{A}', \delta'), \mathcal{D}^{\delta}_{\mathsf{Real}}(\pi)) \approx \texttt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{D}^{\delta}_{\mathsf{Ideal}}(\mathcal{F})). \tag{13}$$

Let us now show,

$$\forall \mathcal{Z}, \texttt{EXEC}(\mathcal{Z}, \mathsf{Filter}(\mathcal{A}', \delta'), \mathcal{D}^{\delta}_{\mathsf{Real}}(\pi)) \approx \texttt{EXEC}(\mathcal{Z}, \mathcal{S}, \mathcal{D}^{\delta'}_{\mathsf{Ideal}}(\mathcal{F})). \tag{14}$$

Assume for the sake of contradiction that there exists an environment $\mathcal{Z}$ that is able to distinguish in Equation (14). We use this to build an environment $\mathcal{Z}'$ which is able to distinguish in Equation (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.

As every time that this happens the environment is correct, and every time this does not happen the execution is exactly similar to that of Equation (13) this implies Equation (14).

We now define $\mathcal{S}' \triangleq \mathcal{S}$ and let $\mathcal{Z}'$ be any environment. We specialise Equation (14) with $\mathcal{Z}'$ and obtain

$$\texttt{EXEC}(\mathcal{Z}', \mathsf{Filter}(\mathcal{A}', \delta'), \mathcal{D}^{\delta}_{\mathsf{Real}}(\pi)) \approx \texttt{EXEC}(\mathcal{Z}', \mathcal{S}, \mathcal{D}^{\delta'}_{\mathsf{Ideal}}(\mathcal{F})). \tag{15}$$

Furthermore,

$$\begin{aligned}\texttt{EXEC}(\mathcal{Z}', \mathsf{Filter}(\mathcal{A}', \delta'), \mathcal{D}^{\delta}_{\mathsf{Real}}(\pi)) &\approx \texttt{EXEC}(\mathcal{Z}', \mathsf{Filter}(\mathcal{A}', \delta'), \mathcal{D}^{\delta'}_{\mathsf{Real}}(\pi)) &(16)\\ &\approx \texttt{EXEC}(\mathcal{Z}', \mathcal{A}', \mathcal{D}^{\delta'}_{\mathsf{Real}}(\pi)). &(17)\end{aligned}$$

Equation (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 (17) holds as it is not observable by the environment if the corruption is ignored by the filter or the shell. Together Equations (15) and (17) finishes the proof. □

Theorems 1 and 2 together imply that any protocol that is secure against a standard adaptive adversary in UC, is also secure against any δ-delayed adversary.

4 Concrete Bounds for Diameters of Erdős–Rényi Graphs

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

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

$$e^{-d\gamma} + \frac{\gamma\alpha}{1-\gamma} \leq 1 \quad \textit{and } (1-\delta_2)\cdot\alpha > 1, \tag{18}$$

then

$$\Pr[\neg\phi_{Diam}(G, t_0+1)] \leq n\left(e^{-\frac{\delta_1^2 d}{2}} + t_0 e^{-\frac{\delta_2^2\alpha(1-\delta_1)d}{2}}\right) + e^{-n\cdot(d\gamma^2-2)}. \tag{19}$$

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 $D_v \coloneqq (|\Gamma^{t_0}(v)| < \gamma\cdot n)$ 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

$$A_0 \coloneqq (\mathtt{deg}(v) > (1-\delta_1)d) \tag{20}$$

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

$$B_i \coloneqq (|\theta(v,i+1)| > (1-\delta_2)\alpha|\theta(v,i)|), \quad C_i \coloneqq \left(|\Gamma^i(v)| > \gamma n\right), \quad \text{and } A_i \coloneqq B_i \vee C_i, \tag{21}$$

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

$$t_0 = \frac{\log\left(\frac{\gamma n}{(1-\delta_1)\cdot d}\right)}{\log((1-\delta_2)\alpha)} + 1 \quad \Longleftrightarrow \quad (1-\delta_1)d((1-\delta_2)\alpha)^{t_0-1} = \gamma n. \tag{22}$$

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

$$\begin{aligned}
|\Gamma^{t_0}(v)| &= \sum_{i=0}^{t_0} |\theta(v,i)| \\
&= 1 + \sum_{i=1}^{t_0} |\theta(v,i)| \\
&= 1 + \sum_{i=0}^{t_0-1} |\theta(v,i+1)| \\
&\geq 1 + \sum_{i=0}^{t_0-1} ((1-\delta_2)\alpha)^i |\theta(v,1)| \\
&\geq 1 + (1-\delta_1)d \sum_{i=0}^{t_0-1} ((1-\delta_2)\alpha)^i \\
&\geq (1-\delta_1)d((1-\delta_2)\alpha)^{t_0-1} \\
&= \gamma n.
\end{aligned} \tag{23}$$

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

$$\left(\bigwedge_{i=0}^{t_0-1} A_i \Longrightarrow \neg D_v\right) \quad \Longleftrightarrow \quad \left(D_v \Longrightarrow \bigvee_{i=0}^{t_0-1} \neg A_i\right). \tag{24}$$

Hence, we get the following:

$$\begin{aligned}
\Pr[D_v] &\leq \Pr\left[\bigcup_{i=0}^{t_0-1} \neg A_i\right] \\
&\leq \sum_{i=0}^{t_0-1} \Pr\left[\neg A_i \mid \bigcap_{j<i} A_j\right] \\
&= \Pr[\neg A_0] + \sum_{i=1}^{t_0-1} \Pr\left[\neg A_i \mid \bigcap_{j<i} A_j\right] \\
&= \Pr[\neg A_0] + \sum_{i=1}^{t_0-1} \Pr\left[\neg B_i \cap \neg C_i \mid \bigcap_{j<i} A_j\right] \\
&= \Pr[\neg A_0] + \sum_{i=1}^{t_0-1} \Pr\left[\neg B_i \mid \bigcap_{j<i} A_j \cap \neg C_i\right] \cdot \Pr\left[\neg C_i \mid \bigcap_{j<i} A_j\right] \\
&\leq \Pr[\neg A_0] + \sum_{i=1}^{t_0-1} \Pr\left[\neg B_i \mid \bigcap_{j<i} A_j \cap \neg C_i\right] \\
&= \Pr[\neg A_0] + \sum_{i=1}^{t_0-1} \Pr\left[\neg B_i \mid \bigcap_{1\leq j<i} B_j \cap \neg C_i \cap A_0\right].
\end{aligned} \tag{25}$$

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

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

$$\Pr\left[\neg B_i \mid \bigcap_{1\leq j<i} B_j \cap \neg C_i \cap A_0\right] \leq e^{-\frac{\delta_2^2 \alpha (1-\delta_1) d}{2}}. \tag{26}$$

Proof. We look at the probability space where $\bigcap_{1\leq j<i} B_j \cap \neg C_i \cap A_0$ holds. Let $r \coloneqq |\theta(v, i)|$ and let $U \coloneqq \mathcal{V} \setminus \Gamma^i(v)$. 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,

$$\Pr[X_u = 1] = 1 - (1-\rho)^r \geq 1 - e^{-\rho r}. \tag{27}$$

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

$$|\theta(v, i+1)| = \sum_{u \in U} X_u. \tag{28}$$

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

$$\mathbb{E}[|\theta(v, i+1)|] \geq n(1-\gamma)\left(1 - e^{-\rho r}\right). \tag{29}$$

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

$$\begin{aligned}
\mathbb{E}[|\theta(v, i+1)|] - \alpha r &\geq n(1-\gamma)\left(1 - e^{-\rho r} - \frac{\alpha r}{n(1-\gamma)}\right) \\
&= n(1-\gamma)\left(1 - e^{-d\frac{r}{n}} - \frac{\alpha r}{n(1-\gamma)}\right).
\end{aligned} \tag{30}$$

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 implies 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 $\mathbb{E}[|\theta(v, i+1)|] \geq \alpha \cdot |\theta(v, i)|$.

$$f(0) = 1 - e^{-d \cdot 0} - 0 = 0. \tag{31}$$

$$f(\gamma) = 1 - e^{-d\gamma} - \frac{\gamma\alpha}{1-\gamma} \geq 0 \quad \iff \quad e^{-d\gamma} + \frac{\gamma\alpha}{1-\gamma} \leq 1. \tag{32}$$

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

$$\Pr[|\theta(v, i+1)| \leq (1-\delta_2)\alpha|\theta(v, i)|] \leq e^{-\frac{\delta_2^2 \alpha |\theta(v,i)|}{2}}. \tag{33}$$

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

$$\begin{aligned} |\theta(v, i)| &\geq ((1-\delta_2)\alpha)^i \cdot (1-\delta_1)d \\ &\geq (1-\delta_1)d. \end{aligned} \tag{34}$$

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

$$\Pr[|\theta(v, i+1)| \leq (1-\delta_2)\alpha|\theta(v, i)|] \leq e^{-\frac{\delta_2^2 \alpha (1-\delta_1) d}{2}}. \tag{35}$$

□

Using Claim 1 and Chernoff (Lemma 1) to bound A_0 we get that

$$\begin{aligned} \Pr[D_v] &\leq e^{-\frac{\delta_1^2 d}{2}} + \sum_{i=1}^{t_0 - 1} e^{-\frac{\delta_2^2 \alpha (1-\delta_1) d}{2}} \\ &\leq e^{-\frac{\delta_1^2 d}{2}} + t_0 e^{-\frac{\delta_2^2 \alpha (1-\delta_1) d}{2}}. \end{aligned} \tag{36}$$

Furthermore, by the union-bound we get that

$$\begin{aligned} \Pr[\text{there exists } v \in \mathcal{V} \text{ s.t. } D_v] &= \Pr\left[\bigcup_{v \in \mathcal{V}} D_v\right] \\ &\leq \sum_{v \in \mathcal{V}} \Pr\left[D_v\right] \\ &\leq n \cdot \left(e^{-\frac{\delta_1^2 d}{2}} + t_0 e^{-\frac{\delta_2^2 \alpha (1-\delta_1) d}{2}}\right). \end{aligned} \tag{37}$$

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'.

$$\begin{aligned} \Pr[\text{No edges from } S \text{ to } S'] &\leq ((1-\rho)^{\gamma n})^{\gamma n} \\ &= \left(\left(1 - \frac{d}{n}\right)^n\right)^{\gamma^2 n} \\ &\leq e^{-d \cdot n \gamma^2} \end{aligned} \tag{38}$$

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

$$\begin{aligned} &\Pr[\text{exist two non-overlapping sets of size} \geq \gamma n \text{ that are not connected}] \\ &\leq 2^{2n} e^{-d \cdot n \gamma^2} \\ &\leq e^{-n \cdot (d\gamma^2 - 2)}. \end{aligned} \tag{39}$$

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. □

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

$$\Pr[\neg\phi_{Diam}(G, 2)] \leq n^2 \cdot e^{-k \cdot \frac{(n-2)}{n}}. \tag{40}$$

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 are missing is $1 - \rho^2$. As there are $n - 2$ possible intermediary nodes we get by the exponential inequality that

$$\begin{aligned} \Pr[X_{v,u} = 1] &= (1 - \rho^2)^{n-2} \\ &\leq e^{-\rho^2 \cdot (n-2)} \\ &= e^{-k \cdot \frac{(n-2)}{n}}. \end{aligned} \tag{41}$$

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

$$\begin{aligned} \Pr[\text{exists } v, u \in \mathcal{V} \text{ s.t.} X_{v,u} = 1] &\leq \sum_{v,u \in \mathcal{V}} \Pr[X_{v,u} = 1] \\ &= \binom{n}{2} \cdot e^{-\rho^2 \cdot (n-2)} \\ &\leq n^2 \cdot e^{-k \cdot \frac{(n-2)}{n}} \end{aligned} \tag{42}$$

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

5 Functionalities

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

MessageTransfer: A functionality that allows one party to send messages to another party. This is modelling a point-to-point channel.

Flood: A functionality that allows all honest parties to disseminate to all other parties.

Conventions for ideal functionalities. Our functionalities needs 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 make use of the shells described in Section 3 to make the corruption-model explicit when implementing the functionalities.

Additionally, the behaviour of both our functionalities depend on which parties are pre-corrupted and which parties that 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 $\texttt{Precorrupted} \coloneqq \texttt{Precorrupted} \cup \{p_i\}$.

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

Furthermore, both of our ideal functionalities are parameterized by a type of messages that can be propagated which we denote MESSAGES.

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 [BDD+21], but instead of hardcoding whether we assume AMS (as done in [BDD+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 needs to stay honest for the delivery guarantee Δ to apply. It maintains a mailbox for p_r, $\mathsf{Mailbox} : \text{MESSAGES}$.

Initialize: Initially, $\mathsf{Mailbox} \coloneqq \varnothing$.

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 $\mathsf{Mailbox}$ to party p_r.

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

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 $m \in \mathsf{Mailbox}$.

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

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.

5.2 Flood

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

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

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 that are to be delivered to each party $\mathsf{Mailbox} : \mathcal{P} \to \textsc{Messages}$.

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

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

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

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

At any time after all parties has been initialized the functionality automatically enforces the following two properties:

1. Let m be a message that are 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 $m \in \mathsf{Mailbox}[p_j]$.
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., $m \in \mathsf{Mailbox}[p_i]$. By 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 $m \in \mathsf{Mailbox}[p_j]$.

The properties are ensured by the functionality automatically makes the minimal possible additional calls with $\mathit{SetMessage}$.

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.

6.1 Naive Flood

We present here a protocol that implements Flood with a message complexity that is quadratic in the number of messages that is 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 very small diameter and resilience against *fairly fast* adaptive adversaries at the cost of a large 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 each in time-step 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 does not have sufficient time to succeed with this as the properties only needs 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 actually realises $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$.

Lemma 5. *Let $\sigma, \Delta \in \mathbb{N}$. The protocol $\pi_{\mathsf{NaiveFlood}}$ perfectly realises $\mathcal{F}^{\Delta}_{\mathsf{Flood}}$ 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 $\bar{\mathcal{G}}_{\mathrm{Ticker}}$ 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 behaviour of $\mathcal{A}$ interacting with $\pi_{\mathsf{NaiveFlood}}$ is equal to the I/O behaviour of $\mathcal{S}$ interacting with $\mathcal{F}^{\sigma,\Delta}_{\mathsf{Flood}}$ 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 exact same

[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 lead 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.

behaviour. 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 ensure 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. □

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 if a particular message should be relayed to this party. Compared to the naive implementation of Flood presented in previous section the protocol presented here will have significantly smaller neighborhoods at the cost of larger diameter in the communication graph (the parameter Δ of Flood). Furthermore, the construction is only able to tolerate adversaries that are slightly more delayed than the 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 that is to be sent/relayed. By letting the random subset be large enough we ensure that we establish a connected graph with 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 in order 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 $\texttt{Relayed}_i \coloneqq \varnothing$.

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 $\texttt{Relayed}_i \coloneqq \texttt{Relayed}_i \cup \{m\}$.

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 atomic time-step 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 $\texttt{Relayed}_i \coloneqq \texttt{Relayed}_i \cup \{m\}$.

Depending on the parameter ρ the protocol π_{ERFlood} can achieve a variety of properties. We provide two different instantiations that uses 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 main intuition is that such an adversary cannot influence how the communication graph between the parties that are honest are 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 make use of the adaptive corruptions to disrupt the propagation of a message.

Each of the instantiations that are presented below provides a trade-off between the diameter of the graph, the average size of the neighborhood and the probability that the graph in fact 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 witha neighbourhood of average size $\Omega\left(\kappa\right)$.

Lemma 6. *Let $\Delta \in \mathbb{N}$ be any delay, let $\sigma \in \mathbb{N}$, let $t < n$, and let $\kappa \in \mathbb{R}$ 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 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 $\rho \coloneqq \sqrt{\frac{\kappa}{h}}$ and let $\Delta' \coloneqq 2\Delta$ then*

$$p_{bad} \leq r \cdot (t+1) \cdot n^2 \cdot e^{-\kappa \cdot \frac{(h-2)}{h}}. \tag{43}$$

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

$$e^{-\kappa\gamma} + \frac{\gamma\alpha}{1-\gamma} \leq 1, \quad \frac{\gamma n}{(1-\delta_1)\kappa} > 1, \quad \text{and} \quad (1-\delta_2)\cdot\alpha > 1, \tag{44}$$

then

$$p_{bad} \leq r \cdot (t+1) \cdot \left(n \cdot \left(e^{-\frac{\delta_1^2 \kappa}{2}} + t_0 e^{-\frac{\delta_2^2 \alpha (1-\delta_1)\kappa}{2}} \right) + e^{-h\cdot(\kappa\gamma^2 - 2)} \right). \tag{45}$$

Proof Sketch. For an adversary we construct a simulator similar to how it is done in the proof of Lemma 5. 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 7 random experiments:

$\mathsf{FloodToErdős}_1$: 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 in order to be able to reveal more nodes. The adversary has the possibility to remove up to t nodes, but at the point of removal the adversary cannot have learned any edges connecting to the removed node. If at any point there is a cut in the graph the adversary can stop the game.

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

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

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

$\mathsf{FloodToErdős}_5$: An experiment similar to $\mathsf{FloodToErdős}_4$ 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.

$\mathsf{FloodToErdős}_6$: An experiment similar to $\mathsf{FloodToErdős}_5$ 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 $d := \frac{\Delta'}{\Delta}$. 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 $\mathsf{FloodToErdős}_1$ return a graph where the distance from the sender to some node is larger than d with at least as high a probability.

2. If any adversary can make $\mathsf{FloodToErdős}_1$ return a graph with a diameter larger than d with probability p, then there exists some adversary that can make $\mathsf{FloodToErdős}_2$ return a graph where the distance from the sender to some node is larger than d with at least as high a probability.

3. If any adversary can make $\mathsf{FloodToErdős}_2$ return a graph with a diameter larger than d with probability p, then there exists some adversary that can make $\mathsf{FloodToErdős}_3$ return a graph where the distance from the sender to some node is larger than d with at least as high a probability.

4. If any adversary can make $\mathsf{FloodToErdős}_3$ return a graph with a diameter larger than d with probability p, then there exists some adversary that can make $\mathsf{FloodToErdős}_4$ return a graph with a diameter larger than d with at least as high a probability.

5. If any adversary can make the $\mathsf{FloodToErdős}_4$ game return a graph with a diameter larger than d with probability p, then the same adversary can make $\mathsf{FloodToErdős}_5$ return a graph with a diameter larger than d with probability at least $p \cdot (t + 1)$.

6. The experiments $\mathsf{FloodToErdős}_5$ and $\mathsf{FloodToErdős}_6$ are distributed identically.

7. The probability that $\mathsf{FloodToErdős}_6$ returns a graph with larger diameter than d must be less than the probability that an Erdős–Rényi graph with the *worst* size has a larger diameter than d.

8. We can now use the Erdős–Rényi graph results from Section 2.2 (in particular Lemmas 3 and 4) 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 that is input to the functionality. The detailed proof can be found in Section 6.3.

□

As the results in Lemma 6 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 Lemma 6 can be used to obtain tighter bounds.

Corollary 1. *Let $\Delta \in \mathbb{N}$ be any delay, let $\sigma \in \mathbb{N}$, let $t < n$, and let $\kappa \in \mathbb{R}$ 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 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 $\rho := \sqrt{\frac{\kappa}{h}}$ and let $\Delta' := 2\Delta$ then*

$$p_{bad} \leq r \cdot (t+1) \cdot n^2 \cdot e^{-\kappa \cdot \frac{(h-2)}{h}}. \tag{46}$$

2. *Let $\rho := \frac{\kappa}{h}$, and $\Delta' := \Delta \cdot (5 \log\left(\frac{n}{2\kappa}\right) + 2)$, if $\frac{n}{2\kappa} > 1$ then*

$$p_{bad} \leq r \cdot (t+1) \cdot \left(7n \log\left(\frac{n}{2\kappa}\right) e^{-\frac{\kappa}{18}} + e^{-\frac{h(\kappa-18)}{9}}\right). \tag{47}$$

Proof. Instantiation 1 immediately follows from Lemma 6 (Instantiation 1). To derive instantiation Instantiation 2 we again use Lemma 6 (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 (44) is fulfilled when $\kappa \geq 1$. Furthermore, we see that

$$\begin{aligned} p_{\text{bad}} &\leq r \cdot (t+1) \cdot \left(n \cdot \left(e^{-\frac{\kappa}{18}} + \left(5 \log\left(\frac{n}{2\kappa}\right) + 1\right) e^{-\frac{7\kappa}{108}}\right) + e^{-\frac{h(\kappa-18)}{9}}\right) \\ &\leq r \cdot (t+1) \cdot \left(7n \log\left(\frac{n}{2\kappa}\right) e^{-\frac{\kappa}{18}} + e^{-\frac{h(\kappa-18)}{9}}\right). \end{aligned} \tag{48}$$

□

In Section 2.2.3 we provide Lemma 2 which shows that the number of neighbours 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 union-bound. Concretely, for Instantiation 1 we get that the number of neighbours is upper-bounded by $\mathcal{O}\left(\sqrt{\kappa n}\right)$ except with a negligible probability, and for Instantiation 2 we get that the number of neighbours is upper-bounded by $\mathcal{O}\left(\kappa\right)$ except with a negligible probability.

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 send via $\mathcal{F}_{\mathsf{MessageTransfer}}$ and ρ is an instantiation of $\pi_{\mathsf{ERFlood}}(\rho)$ that ensures a certain diameter assuming reliable transfer. If we let $\rho' := \frac{\rho}{1-\beta}$ then $\pi_{\mathsf{ERFlood}}(\rho')$ with unreliable transfer is ensured to have the same diameter as $\pi_{\mathsf{ERFlood}}(\rho)$ with reliable transfer. This is because that 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 about $\pi_{\mathsf{ERFlood}}(\rho)$.

6.3 Reducing from π_{ERFlood} to Erdős–Rényi Graphs

In this section we prove Lemma 6. Our goal is to show that the probability that a message from π_{ERFlood} is not delivered timely, is only a small 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 a good a probability of attacking the game as the old one has. A depiction of this approach can be found in Figure 3.

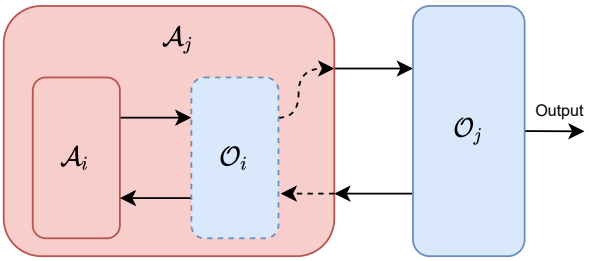

Figure 3: A depiction of the proof methodology for bounding that "something bad" happens when $\mathcal{A}_i$ interacts with the oracle $\mathcal{O}_i$ by the probability that there exists an $\mathcal{A}_j$ which can make "something bad" when interacting with $\mathcal{O}_j$. The idea as that $\mathcal{A}_j$ runs $\mathcal{A}_i$ inside it self while interacting with a simulated $\mathcal{O}_i$ which outputs are correlated with the outputs from the oracle $\mathcal{O}_j$.

In Table 1, we provide an overview of the different experiments we consider and which properties that are bounded in the game.

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

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 `Killed`, and a set of directed edges `Edges`.

Initially, `Revealed` $\coloneqq \varnothing$, `Pending` $\coloneqq \{p_s\}$, `Killables` $\coloneqq \mathcal{P} \setminus \{p_s\}$, `Killed` $\coloneqq \varnothing$, and `Edges` $\coloneqq \varnothing$. The oracle accepts the following inputs from the adversary:

Reveal: On input (Reveal, p_i) the oracle checks if $p_i \in$ `Pending` and otherwise ignores the input. The oracle now continues by adding p_i to `Revealed` and removing p_i from `Pending`. Furthermore, it adds an edge (p_i, p_j) with probability ρ to `Edges` for all $p_j \in$ `Pending`$\cup$`Killables`$\cup$`Revealed`. 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`$| < t$. The oracle then removes p_i from `Killables` and sets `Killed` $\coloneqq$ `Killed` $\cup \{p_i\}$. If not the

Game	Description	Bounded Property
π_{Gossip}	The real protocol.	Everybody receives a message no later than Δ' after it was sent.
$\mathsf{FloodToErdős}_1$	Directed graph with variable size, inject of edges, a specific sender, and early stopping.	Maximum distance from sender to any node is less than $\frac{\Delta'}{\Delta}$.
$\mathsf{FloodToErdős}_2$	Undirected graph with variable size, inject of edges, a specific sender, and early stopping.	Maximum distance from sender to any node is less than $\frac{\Delta'}{\Delta}$.
$\mathsf{FloodToErdős}_3$	Undirected graph with variable size, inject of edges, and a specific sender.	Maximum distance from sender to any node is less than $\frac{\Delta'}{\Delta}$.
$\mathsf{FloodToErdős}_3$	Undirected graph with variable size, inject of edges, and a specific sender.	Diameter of graph is less than $\frac{\Delta'}{\Delta}$.
$\mathsf{FloodToErdős}_4$	Undirected graph with variable size.	Any property.
$\mathsf{FloodToErdős}_5$	Undirected graph with fixed size.	Monotone property.
$\mathsf{FloodToErdős}_6$	Undirected graph with 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 1: Overview of the different games in our reduction in order 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 $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}$. $\mathsf{FloodToErdős}_3$ appears twice as we in separate lemmas bound the distance from a particular sender by the diameter of the graph.

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 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 return $G = (\texttt{Revealed} \cup \texttt{Killables}, \texttt{Edges})$. If not the input is ignored.

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

Lemma 7. *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 send 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 there exists some party at time $\tau + \Delta'$ that is honest and have not yet received m. Furthermore let $d \coloneqq \frac{\Delta'}{\Delta}$. There exists an adversary $\mathcal{A}'$ s.t. if $G_{\mathsf{FloodToErdős}_1} \xleftarrow{\$} \mathsf{FloodToErdős}_1(\mathcal{P}, \rho, \mathcal{A}', p_s)$ then

$$p_{bad} \leq \Pr[\neg \phi_{Dist}(p_s, G_{\mathsf{FloodToErdős}_1}, d)]. \tag{49}$$

Proof. The adversary $\mathcal{A}'$ simulates the execution of $\pi_{\mathsf{Gossip}}(\rho)$ against $\mathcal{A}$ and $\mathcal{Z}$ insides 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 $p_i \in \texttt{Precorrupted} \cup \texttt{Corrupted}$ 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 $p_i \in \texttt{Killables}$ then $\mathcal{A}'$ finds the party p_j in $\texttt{Revealed}$ that is furthest away from the sender p_s. It then inputs $(\mathit{Inject}, p_j, p_i)$.
  2. If $p_i \in \texttt{Pending}$ 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 $p_j \in \texttt{Killed}$, (Send, m) is input to $\mathcal{F}^{\sigma,\Delta}_{\mathsf{MessageTransfer}}(p_i, p_j)$ with probability ρ.
- If at any point $\texttt{Pending} == \varnothing$ then $\mathcal{A}'$ inputs Stop to the oracle.
- At 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 $G_{\mathsf{FloodToErdős}_1} = (\mathcal{V}, E) \xleftarrow{\$} \mathsf{FloodToErdős}_1(\mathcal{P}, \rho, \mathcal{A}', p_s)$ and let the set of nodes that are honest at time $\tau + \Delta'$ be denoted $\mathcal{H}$.

First, we observe that all inputs $\mathcal{A}$ obtain from $\mathcal{A}'$ is 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 have not received the message at time $\tau + \Delta'$ then $\neg\phi_{\mathrm{Dist}}(p_s, G_{\mathsf{FloodToErdős}_1}, d)$.

We first observe the following invariant.

Claim 2. *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 do 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 which has not been removed from the graph is not precorrupted early enough to prevent delivery the party from forwarding the message to their neighbours, 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. *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 Claims 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 gives us 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 $\texttt{Pending} = \varnothing$ for the time-span which would make $\mathcal{A}'$ stop the game and we are done, or there is some party $p_j \in \texttt{Pending}$ at time $\tau + r \cdot \Delta$. However, Claim 2 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 certainly 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 following two commands:

- **(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 $p_i \in \texttt{Pending}$ at time $\tau + r \cdot \Delta$. Claim 2 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 have 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 $p_i \notin \Gamma^d(p_s)$ to show $\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToErdős}_1}, d)$. We now make a case distinction on whether or not p_i has been revealed by $\mathcal{A}'$.

$p_i \in \texttt{Revealed}$: It must be that $p_i \in \texttt{Killables}$ when $\mathcal{A}'$ gave the input Stop to the oracle. Furthermore, only parties that have no incoming edges can be in $\texttt{Killables}$, as whenever an edge is added to a killable party, this party will be moved to $\texttt{Pending}$ by the oracle. Therefore $p_i \notin \Gamma^d(p_s)$.

[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).

$p_i \in \texttt{Revealed}$: 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 actually 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 A) ensures that any party in $\Gamma^d(p_s)$ must have been revealed before time $\tau + d \cdot \Delta$. To show that $p_i \notin \Gamma^d(p_s)$ it is therefore suffices to show that $\Delta' + 1 \geq d \cdot \Delta$. However, $d \cdot \Delta = \frac{\Delta'}{\Delta} \cdot \Delta = \Delta'$ and the inequality is therefore trivially satisfied by definition of d. □

Next, we present a undirected version of the same game but where edges are only added to all parties that have not been revealed before. Furthermore, an adversary has the possibility of revealing any node and not just those that have an edge to them in this game.

Game $\mathsf{FloodToErdős}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$

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, $\texttt{Revealed} \coloneqq \varnothing$, $\texttt{Pending} \coloneqq \{p_s\}$, $\texttt{Killables} \coloneqq \mathcal{P} \setminus \{p_s\}$, $\texttt{Killed} \coloneqq \varnothing$, and $\texttt{Edges} \coloneqq \varnothing$. The oracle accepts the following inputs from the adversary:

Reveal: On input (Reveal, p_i) the oracle checks if $p_i \in \texttt{Pending} \cup \texttt{Killables}$ 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 $p_j \in \texttt{Pending} \cup \texttt{Killables}$. 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 `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 $|\texttt{Killed}| \leq t$. The oracle then removes p_i from `Killables` and sets $\texttt{Killed} \coloneqq \texttt{Killed} \cup \{p_i\}$. 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 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 $\texttt{Pending} == \varnothing$. 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.

We now relate $\mathsf{FloodToErdős}_1$ to $\mathsf{FloodToErdős}_2$ via a simulation argument.

Lemma 8. *Let $\mathcal{A}$ be an adversary, $p_s \in \mathcal{P}$, $d \in \mathbb{N}$ and $\rho \in [0, 1]$. Furthermore let $G_{\mathsf{FloodToErdős}_1} \xleftarrow{\$} \mathsf{FloodToErdős}_1(\mathcal{P}, \rho, \mathcal{A}, p_s)$. There exists an adversary $\mathcal{A}'$ s.t. if $G_{\mathsf{FloodToErdős}_2} \xleftarrow{\$} \mathsf{FloodToErdős}_2(\mathcal{P}, \rho, \mathcal{A}', p_s)$ then*

$$\Pr[\neg\phi_{Dist}(p_s, G_{\mathsf{FloodToErdős}_1}, d)] \leq \Pr[\neg\phi_{Dist}(p_s, G_{\mathsf{FloodToErdős}_2}, d)]. \tag{50}$$

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 $\mathsf{FloodToErdős}_1$ 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 $\mathsf{FloodToErdős}_2$ 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 has already been added with probability ρ. Initially, $\texttt{Revealed}_1 \coloneqq \varnothing$, $\texttt{Pending}_1 \coloneqq \{p_s\}$, $\texttt{Killables}_1 \coloneqq \mathcal{P} \setminus \{p_s\}$, $\texttt{Killed}_1 \coloneqq \varnothing$, $\texttt{HiddenEdges} = \varnothing$ and $\texttt{Flipped}$ the empty map. Furthermore, we let Γ_1 be the neighborhood function for $G_{\mathsf{FloodToErdős}_1}$, and let Γ_2 be the neighborhood function for $G_{\mathsf{FloodToErdős}_2}$. Similar we subscript the distance function with either 1 or 2 to indicate which graph the function calculates the distance on.

- On input (Reveal, p_i) from $\mathcal{A}$ the adversary makes a set $E_{\texttt{Tmp}} \coloneqq \varnothing$ 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 $\texttt{Flipped}(p_j, p_i) \coloneqq \top$ and otherwise $\texttt{Flipped}(p_j, p_i) \coloneqq \bot$.
  2. For $p_j \in \texttt{Pending}$, if $\texttt{Flipped}(p_j, p_i)$ the adversary $\mathcal{A}'$ sets $E_{\texttt{Tmp}} \coloneqq E_{\texttt{Tmp}} \cup \{(p_i, p_j)\}$ with probability ρ.
  3. $\mathcal{A}'$ forwards (Reveal, p_i) to the oracle of $\mathsf{FloodToErdős}_2$. Let E be the set of edges that are 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 $\texttt{HiddenEdges} \coloneqq \texttt{HiddenEdges} \cup \{(p_j, p_i)\}$.
     - Else the adversary adds an edge (p_i, p_j) to $E_{\texttt{Tmp}}$.
  4. The adversary $\mathcal{A}'$ sets $E_{\texttt{Tmp}} \coloneqq E_{\texttt{Tmp}} \cup \{(p_i, p_j) \mid p_j \in \mathcal{P} \wedge (p_i, p_j) \in \texttt{HiddenEdges}\}$.
  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, $\texttt{Edges}_1 \coloneqq \texttt{Edges}_1 \cup E_{\texttt{Tmp}}$, the sets maintained by $\mathcal{A}'$ are updated similarly to how the oracle in $\mathsf{FloodToErdős}_1$ would have updated them, and the set $\texttt{Edges}_1$ is returned to $\mathcal{A}$.
- On input $(\mathit{Inject}, p_i, p_j)$ 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 $\mathcal{A}$ obtain from $\mathcal{A}'$ is distributed identically to those the adversary would see when interacting with the oracle from game $\mathsf{FloodToErdős}_1$. We state this in the claim below.

Claim 4. *Let c be the number of commands executed by $\mathcal{A}'$ and let any set with superscript c denote the set after having executed 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 $\mathsf{FloodToErdős}_1$.*

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 do a case distinction based on the command given by $\mathcal{A}$:

$(\mathcal{R}eveal, p_i)$: 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 such an edge is returned by the oracle, 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

$$\begin{aligned}&\Pr[(p_i, p_j) \in E_{\texttt{Tmp}}] \\ &= \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \texttt{Flipped}(p_j, p_i)] \cdot \Pr[\texttt{Flipped}(p_j, p_i)] \\ &\quad + \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \neg\texttt{Flipped}(p_j, p_i)] \cdot \Pr[\neg\texttt{Flipped}(p_j, p_i)].\end{aligned} \tag{51}$$

  Let E be the set of edges returned by the oracle in $\mathsf{FloodToErdős}_2$. 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

$$\begin{aligned}&\Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \texttt{Flipped}(p_j, p_i)] \\ &= \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \neg\texttt{Flipped}(p_j, p_i)] \\ &= \Pr[(p_i, p_j) \in E] \\ &= \rho.\end{aligned} \tag{52}$$

  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 command $(\mathcal{R}eveal, p_j)$ 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

$$\begin{aligned}&\Pr[(p_i, p_j) \in E_{\texttt{Tmp}}] \\ &= \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \texttt{Flipped}(p_j, p_i)] \cdot \Pr[\texttt{Flipped}(p_j, p_i)] \\ &\quad + \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \neg\texttt{Flipped}(p_j, p_i)] \cdot \Pr[\neg\texttt{Flipped}(p_j, p_i)].\end{aligned} \tag{53}$$

  If $\texttt{Flipped}(p_j, p_i)$ then an edge (p_i, p_j) was added to $\texttt{HiddenEdges}$ with probability ρ when $(\mathcal{R}eveal, p_j)$ was given as input. Such an edge always end 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,

$$\begin{aligned}&\Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \texttt{Flipped}(p_j, p_i)] \\ &= \Pr[(p_i, p_j) \in E_{\texttt{Tmp}} \mid \neg\texttt{Flipped}(p_j, p_i)] \\ &= \rho.\end{aligned} \tag{54}$$

  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.

(Kill, p_i)**:** As the sets are in synchrony for c' commands any kill command that would be valid in $\mathsf{FloodToErdős}_1$ is also valid in $\mathsf{FloodToErdős}_2$. Therefore the sets stay synchronized.

($\mathit{Inject}, p_i, p_j$)**:** As the sets are in synchrony for c' commands any inject command that would be valid in $\mathsf{FloodToErdős}_1$ is also valid in $\mathsf{FloodToErdős}_2$. Therefore the sets stay synchronized.

(Stop)**:** This command doesn't change any of the involved sets.

□

Given the above claim it suffices to show that $\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToErdős}_1}, d)$ implies $\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToErdős}_2}, d)$. To show this we let $G_{\mathsf{FloodToErdős}_1} = (\mathcal{V}_1, E_1)$ and let $G_{\mathsf{FloodToErdős}_2} = (\mathcal{V}_2, E_2)$.

Now, let $p_i \in \mathcal{V}_1$ s.t. $p_i \notin \Gamma_1^d(p_s)$ and let us show that $p_i \in \mathcal{V}_2$ and $p_i \notin \Gamma_2^d(p_s)$. 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 $p_i \notin \Gamma_2^d(p_s)$. We show the lemma below.

Claim 5. *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 do 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 on how this edge was added to $\texttt{Edges}_2$:

- (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 `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 $p_j \in \Gamma_1^k(p_s)$.
- (Reveal, p_j)**:** If the edge was added by this command 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 $p_i \in \Gamma_2^d(p_s)$. Claim 5 implies that $p_i \in \Gamma_1^d(p_s)$ which contradicts with the original assumption which was that $p_i \notin \Gamma_1^d(p_s)$. □

Next, we present a version of the game which cannot be stopped before all notes are either revealed or killed.

Game $\mathsf{FloodToErdős}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$

The game is identical to $\mathsf{FloodToErdős}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$ except that the oracle in this game ignores *any* (*Stop*) inputs from the adversary. All other inputs are treated identically to how the oracle from $\mathsf{FloodToErdős}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$ treats them.
When $\texttt{Killables} \cup \texttt{Pending} = \varnothing$ the game ends by the oracle returning $G = (\texttt{Revealed}, \texttt{Edges})$.

Lemma 9. *Let $\mathcal{A}$ be an adversary, $p_s \in \mathcal{P}$, $d \in \mathbb{N}$ and $\rho \in [0,1]$. Furthermore let $G_{\mathsf{FloodToErdős}_2} \xleftarrow{\$} \mathsf{FloodToErdős}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$. There exists an adversary $\mathcal{A}'$ s.t. if $G_{\mathsf{FloodToErdős}_3} \xleftarrow{\$} \mathsf{FloodToErdős}_3(\mathcal{P}, \rho, \mathcal{A}', p_s)$ then*

$$\Pr[\neg\phi_{Dist}(p_s, G_{\mathsf{FloodToErdős}_2}, d)] \leq \Pr[\neg\phi_{Dist}(p_s, G_{\mathsf{FloodToErdős}_3}, d)]. \tag{55}$$

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 $\mathsf{FloodToErdős}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$ end the game (i.e., if $\texttt{Pending} == \varnothing$). If that is the case the adversary inputs (*Reveal*, p_i) for any party p_i that was in $\texttt{Killables}$ 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 $\mathsf{FloodToErdős}_2(\mathcal{P}, \rho, \mathcal{A}, p_s)$. It therefore suffices to show that $\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToErdős}_2}, d)$ implies $\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToErdős}_3}, d)$. 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 $\mathsf{FloodToErdős}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$ ends. If no such input is given we have that $G_{\mathsf{FloodToErdős}_2} = G_{\mathsf{FloodToErdős}_3}$ and we are done.

If there is a (*Stop*) input from $\mathcal{A}$ that is not ignored by $\mathcal{A}'$ before $\mathsf{FloodToErdős}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$ ends, then there must have been a point in time, τ, where $\texttt{Pending} = \varnothing$ and $\texttt{Killables} \neq \varnothing$. 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 concludes the proof. □

That the distance from a specific node is less than some distance, d, is a strictly weaker property than the diameter of a graph being less than d. The below lemma follows immediately.

Lemma 10. *Let $\mathcal{A}$ be an adversary, $p_s \in \mathcal{P}$, $d \in \mathbb{N}$, $\rho \in [0,1]$, and let $G_{\mathsf{FloodToErdős}_3} \xleftarrow{\$} \mathsf{FloodToErdős}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$. We have that*

$$\Pr[\neg\phi_{Dist}(p_s, G_{\mathsf{FloodToErdős}_3}, d)] \leq \Pr[\neg\phi_{Diam}(G_{\mathsf{FloodToErdős}_3}, d)]. \tag{56}$$

Next, we present a version of the game where there are no specific node that starts being pending and the adversary can no longer inject additional edges to the graph.

Game $\mathsf{FloodToErdős}_4(\mathcal{P}, \rho, \mathcal{A})$

The game is identical to $\mathsf{FloodToErdős}_3$ except two things:

- Initially, $\texttt{Pending} \coloneqq \varnothing$ and $\texttt{Killables} \coloneqq \mathcal{P}$.
- The oracle ignores *any* $(\textit{Inject}, p_i, p_j)$ inputs from the adversary.

All other inputs are treated identically to how the oracle from $\mathsf{FloodToErdős}_3$ treats them.

Before relating $\mathsf{FloodToErdős}_4(\mathcal{P}, \rho, \mathcal{A})$ to $\mathsf{FloodToErdős}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$ we define a special kind of graph properties called *monotone* properties.

Definition 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 11. *Let $\mathcal{A}$ be an adversary, $p_s \in \mathcal{P}$, $d \in \mathbb{N}$, $\rho \in [0, 1]$, and let ϕ be a monotone graph property. Furthermore let $G_{\mathsf{FloodToErdős}_3} \xleftarrow{\$} \mathsf{FloodToErdős}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$. There exists an adversary $\mathcal{A}'$ s.t. if $G_{\mathsf{FloodToErdős}_4} \xleftarrow{\$} \mathsf{FloodToErdős}_4(\mathcal{P}, \rho, \mathcal{A}')$ then*

$$\Pr[\neg\phi(G_{\mathsf{FloodToErdős}_3})] \leq \Pr[\neg\phi(G_{\mathsf{FloodToErdős}_4})]. \tag{57}$$

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 $\mathsf{FloodToErdős}_4$, and let $\mathcal{A}'$ maintain similar sets indexed with 3 which will be presented to $\mathcal{A}$ similar to the sets the oracle in $\mathsf{FloodToErdős}_3$ presents to $\mathcal{A}$. Initially $\texttt{Pending}_3 \coloneqq \{p_s\}$ and $\texttt{Killables}_3 = \mathcal{P} \setminus \{p_s\}$.

- On input $(\textit{Inject}, p_i, p_j)$, $\mathcal{A}'$ 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 $\mathsf{FloodToErdős}_3$). If they should not be ignored by these rules, 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 $\mathsf{FloodToErdős}_3(\mathcal{P}, \rho, \mathcal{A}, p_s)$. 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 $\mathsf{FloodToErdős}_4$.

Moreover, we have that $\texttt{Edges}_4 \subseteq \texttt{Edges}_3$. Therefore if $\phi(G_{\mathsf{FloodToErdős}_4})$ then, as ϕ is monotone, $\phi(G_{\mathsf{FloodToErdős}_3})$. □

The game $\mathsf{FloodToErdős}_4(\mathcal{P}, \rho, \mathcal{A})$ 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 is able to dynamically choose the size of the graph. A hypothetical example where this is useful is in a situation where 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 definitely a better strategy to kill one and reveal the other compared to revealing/killing 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 influence of the adversary.

Game $\mathsf{FloodToErdős}_5(\mathcal{P}, \rho, \mathcal{A})$

The game is identical to $\mathsf{FloodToErdős}_4$ 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 $\texttt{Revealed} \coloneqq \{p_1, \ldots, p_s\}$ and $\texttt{Edges} \coloneqq \texttt{Revealed} \times \texttt{Revealed}$ before immediately ending the game by outputting $G = (\texttt{Revealed}, \texttt{Edges})$.[b]

All other behaviour is identical to the oracle from $\mathsf{FloodToErdős}_4$.

[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 which cannot be swapped with any party in the final kill set.

Lemma 12. *Let ϕ be a graph property. For any adversary $\mathcal{A}$, let $G_{\mathsf{FloodToErdős}_4} \xleftarrow{\$} \mathsf{FloodToErdős}_4(\mathcal{P}, \rho, \mathcal{A})$, and let $G_{\mathsf{FloodToErdős}_5} \xleftarrow{\$} \mathsf{FloodToErdős}_5(\mathcal{P}, \rho, \mathcal{A})$. We have that*

$$\Pr[\phi(G_{\mathsf{FloodToErdős}_4})] \leq \Pr[\phi(G_{\mathsf{FloodToErdős}_5})] \cdot (t+1). \tag{58}$$

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 did 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

$$\begin{aligned}\Pr[\phi(G_{\mathsf{FloodToErdős}_5})] &= \Pr[\phi(G_{\mathsf{FloodToErdős}_5}) \mid \texttt{GG}] \cdot \Pr[\texttt{GG}] + \Pr[\phi(G_{\mathsf{FloodToErdős}_5}) \mid \neg\texttt{GG}] \cdot \Pr[\neg\texttt{GG}] \\ &\geq \Pr[\phi(G_{\mathsf{FloodToErdős}_5}) \mid \texttt{GG}] \cdot \Pr[\texttt{GG}].\end{aligned} \tag{59}$$

We now note that the adversary cannot influence the probability that a guess is correct, as the behavior of $\mathcal{O}_5$ is exactly equal to the behavior of $\mathcal{O}_4$ until the game terminates. When this happens it is anyway to 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 to see $\texttt{GG}$ is

$$\Pr[\texttt{GG}] = (t+1)^{-1}. \tag{60}$$

By Equations (59) and (60) it is sufficient to show that

$$\Pr[\phi(G_{\mathsf{FloodToErdős}_5}) \mid \texttt{GG}] = \Pr[\phi(G_{\mathsf{FloodToErdős}_4})]. \tag{61}$$

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 $\mathsf{FloodToErdős}_6(\mathcal{P}, \rho, \mathcal{A})$

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, $s \xleftarrow{\$} \mathcal{U}(\{h, \ldots, n\})$, and then samples a graph $G_s = (\mathcal{V}, E) \xleftarrow{\$} \mathbb{G}(s, \rho)$.
Initially, $\texttt{Revealed} \coloneqq \varnothing$, $\texttt{Pending} \coloneqq \varnothing$, $\texttt{Killables} \coloneqq \mathcal{P}$, $\texttt{Killed} \coloneqq \varnothing$, $\texttt{Edges} \coloneqq \varnothing$. Additionally, the oracle maintains a map $\texttt{Naming} : \mathcal{P} \to \{1, \ldots, s\}$ and a set of available names `AvailableNames`. The map starts out being empty and $\texttt{AvailableNames} \coloneqq \{1, \ldots, s\}$.
The oracle accepts the following inputs from the adversary:

Reveal: On input (Reveal, 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 $\texttt{Naming}[p_i] \coloneqq \eta$, and sets $\texttt{AvailableNames} \coloneqq \texttt{AvailableNames} \setminus \{\eta\}$.
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 $e \in \{\{\texttt{Naming}[p_i], j\} \in E \mid j \in \{1, \ldots, s\}\}$ let $e = \{\texttt{Naming}[p_i], j\}$ for some j and do the following:
   (a) If $\texttt{Naming}^{-1}[j] == \perp$ then the oracle samples $p \xleftarrow{\$} \mathcal{U}(\texttt{Killables})$, sets $\texttt{Naming}[p] \coloneqq j$, and sets $\texttt{AvailableNames} \coloneqq \texttt{AvailableNames} \setminus \{j\}$, and moves p from `Killables` to `Pending`.
   (b) Set $\texttt{Edges} \coloneqq \texttt{Edges} \cup \{\{p_i, \texttt{Naming}^{-1}[j]\}\}$.
4. Now for each $\{1, \ldots, n - s - |\texttt{Killed}|\}$ 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 $\texttt{Naming}[p] \coloneqq \eta$, 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 (Kill, p_i), the oracle checks if $p_i \in \texttt{Killables}$ and if $|\texttt{Killed}| \leq t$. The oracle then removes p_i from `Killables` and sets $\texttt{Killed} \coloneqq \texttt{Killed} \cup \{p_i\}$. 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 $\texttt{Revealed} \coloneqq \{p_1, \ldots, p_s\}$

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 13. *Let $\rho \in [0,1]$. For any adversary $\mathcal{A}$,*

$$\mathsf{FloodToErdős}_5(\mathcal{P}, \rho, \mathcal{A}) \approx \mathsf{FloodToErdős}_6(\mathcal{P}, \rho, \mathcal{A}). \tag{62}$$

Proof. Let $\mathcal{O}_5$ be the oracle from $\mathsf{FloodToErdős}_5$ and let $\mathcal{O}_6$ be the oracle from $\mathsf{FloodToErdős}_6$. 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 having executed the c'th command.

We now state and prove two claims that are needed for the main result.

Claim 6. *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 7. *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

$$|\texttt{AvailableNames}^0| = |\{1, \ldots, s\}| = s. \tag{63}$$

We now argue about the induction case $c = c' + 1$. As (Kill, p_i) inputs doesn't change any of the sets, it is sufficient to argue about the case where the c'th command is of the form (Reveal, 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 do 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. Afterwards 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 8. *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 are 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 initialised 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 (Kill, p_i) then the claim follows by the induction hypothesis. It is therefore sufficient to look at the case when (Reveal, p_i) is input. Furthermore, as sets are updated identically based upon the output of the command for both oracles it is sufficient to argue about the output distribution of the edges. We do 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 6 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 have not yet had assigned an edge, or it can be added by the additional flips made afterwards. For any node $v \in \texttt{AvailableNames}$ (which is exactly 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 afterwards, 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 7 we have that

$$\begin{aligned} &|\texttt{AvailableNames}| + (n - s - |\texttt{Killed}_6^{c'}|) \\ &= s - (|\texttt{Revealed}_6^{c'} + |\texttt{Pending}_6^{c'}||) + n + s - |\texttt{Killed}_6^{c'}| \\ &= n - |\texttt{Revealed}_6^{c'}| - |\texttt{Pending}_6^{c'}| - |\texttt{Killed}_6^{c'}| \\ &= |\texttt{Killables}_6^{c'}|. \end{aligned} \tag{64}$$

  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 8 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 8) and $\mathcal{O}_6$ rules for early stopping is a super set 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 7 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 have 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 killables if $n - |\texttt{Killed}_6| < s$. Let us assume that $\texttt{Killables} = \varnothing$ and $n - |\texttt{Killed}_6| \geq s$. We have that:

$$\begin{aligned} & n - |\texttt{Killed}_6| \\ & = |\texttt{Revealed}_6| + |\texttt{Pending}_6| + |\texttt{Killables}_6| + |\texttt{Killed}_6| - |\texttt{Killed}_6| \\ & = |\texttt{Revealed}_6| + |\texttt{Pending}_6|. \end{aligned} \tag{65}$$

However, as $\texttt{Naming}$ is injective, we have by Claim 6 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 $\texttt{Naming}^{-1} == \bot$ 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 $\mathsf{FloodToErdős}_6$-game to the probability that an Erdős–Rényi graph has the property. This is however only true for a restricted class of properties which we define below.

Definition 13 (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, $\mathtt{r} : \mathcal{V} \to \mathcal{V}'$, we have that

$$\phi(G) \implies \phi((\{\mathtt{r}(v) \mid v \in \mathcal{V}\}, \{\{\mathtt{r}(v), \mathtt{r}(u)\} \mid \{v, u\} \in E\})).$$

Lemma 14. *Let ϕ be a monotone graph property that is preserved under renaming and let $G_s \xleftarrow{\$} \mathbb{G}(s, \rho)$ for any $s \in \mathbb{N}$. For any adversary $\mathcal{A}$ let $G_{\mathsf{FloodToErdős}_6} \xleftarrow{\$} \mathsf{FloodToErdős}_6(\mathcal{P}, \rho, \mathcal{A})$ then*

$$\Pr[\neg\phi(G_{\mathsf{FloodToErdős}_6})] \leq \max_{s \in \{h, \ldots, n\}} \Pr[\neg\phi(G_s)]. \tag{66}$$

Proof. Let $G = (\mathcal{V}, E) \xleftarrow{\$} \mathsf{FloodToErdős}_6(\mathcal{P}, \rho, \mathcal{A})$. By law of total probabilities we have

$$\begin{aligned} \Pr[\neg\phi(G)] &= \sum_{s=h}^{n} \Pr[\neg\phi(G) \mid s = |G|] \cdot \Pr[s = |G|] \\ &\leq \left(\max_{s \in \{h, \ldots, n\}} \Pr[\neg\phi(G) \mid s = |G|] \right) \cdot \sum_{s=h}^{n} \Pr[s = |G|] \\ &= \max_{s \in \{h, \ldots, n\}} \Pr[\neg\phi(G) \mid s = |G|]. \end{aligned} \tag{67}$$

Let $s_0 \coloneqq \operatorname{argmax}_{s \in \{h, \ldots, n\}} \Pr[\neg\phi(G) \mid s = |G|]$. As

$$\max_{s \in \{h, \ldots, n\}} \Pr[\neg\phi(G_s)] \geq \Pr[\neg\phi(G_{s_0})], \tag{68}$$

it suffices to show that

$$\Pr[\neg\phi(G_{s_0})] \geq \Pr[\neg\phi(G) \mid s_0 = |G|]. \tag{69}$$

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:

$$\Pr[\phi(G_{s_0})] \leq \Pr[\phi(G) \mid s_0 = |G|]. \tag{70}$$

Let $G' = (\mathcal{V}', E')$ be the Erdős–Rényi graph held internally by the oracle in $\mathsf{FloodToErdős}_6$. 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 (70) is trivially fulfilled as both sides are equal to 0). Therefore, if the game ended early then $\phi(G)$ as the full graph is returned. Otherwise, if the game did not end early then let $G'' = (\mathcal{V}'', E'') = (\texttt{Naming}^{-1}(\mathcal{V}), \texttt{Naming}^{-1}(E))$. 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). □

6.3.2 Proving π_{Gossip} Secure

We are now finally ready to prove our main result.

Lemma 6. *Let $\Delta \in \mathbb{N}$ be any delay, let $\sigma \in \mathbb{N}$, let $t < n$, and let $\kappa \in \mathbb{R}$ 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 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 $\rho \coloneqq \sqrt{\frac{\kappa}{h}}$ and let $\Delta' \coloneqq 2\Delta$ then*

$$p_{bad} \leq r \cdot (t+1) \cdot n^2 \cdot e^{-\kappa \cdot \frac{(h-2)}{h}}. \tag{43}$$

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

$$e^{-\kappa\gamma} + \frac{\gamma\alpha}{1-\gamma} \leq 1, \quad \frac{\gamma n}{(1-\delta_1)\kappa} > 1, \quad \text{and} \quad (1-\delta_2)\cdot\alpha > 1, \tag{44}$$

then

$$p_{bad} \leq r \cdot (t+1) \cdot \left(n \cdot \left(e^{-\frac{\delta_1^2 \kappa}{2}} + t_0 e^{-\frac{\delta_2^2 \alpha (1-\delta_1)\kappa}{2}} \right) + e^{-h\cdot(\kappa\gamma^2 - 2)} \right). \tag{45}$$

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 5.

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 $\mathcal{F}_{\mathsf{Flood}}^{\Delta'}$ 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 $\mathcal{F}_{\mathsf{MessageTransfer}}^{\sigma,\Delta}(p_i, p_j)$, $\mathcal{S}$ forwards $(\mathit{SetMessage}, m, p_j)$ to $\mathcal{F}_{\mathsf{Flood}}^{\Delta'}$.
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 properties of the ideal functionality is violated in $\pi_{\mathsf{ERFlood}}(\rho)$. In this case the ideal functionality will enforce the properties it self where as 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 is 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 send 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

$$\begin{aligned} p_{\text{bad}} &\leq \Pr\left[\bigcup_{m \in M} \texttt{NoDelivery}(m)\right] \\ &\leq \sum_{m \in M} \Pr\left[\texttt{NoDelivery}(m)\right]. \end{aligned} \tag{71}$$

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 send from some dishonest party) to some honest party p_s. Now, for $d := \frac{\Delta'}{\Delta}$, Lemma 7 ensures that there exists an adversary $\mathcal{A}_1$ s.t. if $G_{\mathsf{FloodToErdős}_1} \xleftarrow{\$} \mathsf{FloodToErdős}_1(\mathcal{P}, \rho, \mathcal{A}_1, p_s)$ then

$$\Pr[\texttt{NoDelivery}(m)] \leq \Pr[\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToErdős}_1}, d)]. \tag{72}$$

Lemma 8 ensures that there exists an adversary $\mathcal{A}_2$ s.t. if $G_{\mathsf{FloodToErdős}_2} \xleftarrow{\$} \mathsf{FloodToErdős}_2(\mathcal{P}, \rho, \mathcal{A}_2, p_s)$ then

$$\Pr[\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToErdős}_1}, d)] \leq \Pr[\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToErdős}_2}, d)]. \tag{73}$$

Lemmas 9 and 10 ensures that there exists an adversary $\mathcal{A}_3$ s.t. if $G_{\mathsf{FloodToErdős}_3} \xleftarrow{\$} \mathsf{FloodToErdős}_3(\mathcal{P}, \rho, \mathcal{A}_3, p_s)$ then

$$\begin{aligned} \Pr[\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToErdős}_2}, d)] &\leq \Pr[\neg\phi_{\text{Dist}}(p_s, G_{\mathsf{FloodToErdős}_3}, d)] \\ &\leq \Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToErdős}_3}, d)]. \end{aligned} \tag{74}$$

As ϕ_{Dist} is monotone, Lemma 11 ensures that there exists an adversary $\mathcal{A}_4$ s.t. if $G_{\mathsf{FloodToErdős}_4} \xleftarrow{\$} \mathsf{FloodToErdős}_4(\mathcal{P}, \rho, \mathcal{A}_4)$ then

$$\Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToErdős}_3}, d)] \leq \Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToErdős}_4}, d)]. \tag{75}$$

Lemma 12 ensures that if $G_{\mathsf{FloodToErdős}_5} \xleftarrow{\$} \mathsf{FloodToErdős}_5(\mathcal{P}, \rho, \mathcal{A}_4)$ then

$$\Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToErdős}_4}, d)] \leq \Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToErdős}_5}, d)] \cdot (t+1). \tag{76}$$

Furthermore, as ϕ_{Diam} is also preserved under renaming, Lemmas 13 and 14 ensures that if $G_s \xleftarrow{\$} \mathbb{G}(s, \rho)$ then

$$\Pr[\neg\phi_{\text{Diam}}(G_{\mathsf{FloodToErdős}_5}, d)] \leq \max_{s \in \{h, \ldots, n\}} \Pr[\neg\phi_{\text{Diam}}(G_s, d)]. \tag{77}$$

Combining Equations (71) to (77) and using that $|M| = r$ we get that

$$p_{\text{bad}} \leq r \cdot (t+1) \cdot \max_{s \in \{h, \ldots, n\}} \Pr[\neg\phi_{\text{Diam}}(G_s, d)]. \tag{78}$$

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 (43) using Lemma 4. For Instantiation 2 we obtain Equation (45) using Lemma 3. □

7 Conclusion and Future Work

We have formally defined the model of δ-delayed adversaries within the UC framework. This has allowed us to precisely characterize and prove the security guarantees of the flooding protocol, π_{ERFlood}. Thereby, we have taken a first step at putting the widely assumed flooding functionalities on firm ground against adaptive adversaries.

Several interesting directions for future work remain. In this work, we have explored a particular type of flooding protocol based upon Erdős–Rényi graphs. However, as discussed earlier, there exist several more complex constructions for different gossip networks in the literature. Analyzing such protocols against δ-delayed corruptions could potentially yield protocols that are even more efficient than what is presented here while also providing a well-understood security guarantee. Another direction could be to optimize for security instead of efficiency. The flooding protocol that we have presented is only secure against adversaries that are delayed by at least $(\sigma + \Delta)$. An interesting question that arises is whether this is inherent for flooding networks, or whether it is possible to implement a flooding network that is secure against a 0-delayed adversary.

Furthermore, this work has considered adaptive but not *mobile* adversaries, which can again uncorrupt parties. For some notion of mobility, it seems that π_{ERFlood} could be secure even in the presence of such mobile adversaries. Extending the model of δ-delayed adversaries to include some notion of mobility would be useful in order to better understand guarantees of blockchain protocols that are supposed to run for a very long time.

Acknowledgements. We thank Ran Cannetti for a careful explanation of a subtle detail of the UC framework, Sabine Oechsner for discussions in the initial phase of the project, and the anonymous reviewers of Eurocrypt for their corrections.

Appendix A Expansion of Erdős–Rényi Graphs

In this section we reprove Corollary 6 from [CCG+15b] in order to obtain concrete bounds comparable to our other results.

Before doing so we prove a couple of elementary graph results about a particular type of graph where all small subsets of nodes in a graph are very strongly connected to the remaining part of the graph.

Definition 14 (Expanders)**.** A graph $G = (\mathcal{V}, E)$ is an *α-edge-expander* if for any $S \subseteq \mathcal{V}$ where $|S| \leq \frac{|\mathcal{V}|}{2}$ we have that

$$\alpha|S| \leq |\{\{v, u\} \in E \mid v \in S \wedge u \in \mathcal{V} \setminus S\}|.$$

We say that G is just an *α-expander* if for any $S \subseteq \mathcal{V}$ where $|S| \leq \frac{|\mathcal{V}|}{2}$ we have that

$$\alpha|S| \leq |\Gamma(S)|.$$

Lemma 15 (Edge-expansion $\Rightarrow$ vertex-expansion)**.** *Let $G = (\mathcal{V}, E)$ be a graph and set $d \coloneqq \max_{v \in V} \mathtt{deg}(v)$. If G is an α-out-edge-expander, it is an $(1 + \frac{\alpha}{d})$-expander.*

Proof. Let $S \subseteq \mathcal{V}$ s.t. $|S| \leq \frac{|V|}{2}$. We want to show that $(1 + \frac{\alpha}{d})|S| \leq |\Gamma(S)|$. Note that $S \subseteq \Gamma(S)$ which means that it is enough to show that $\frac{\alpha}{d}|S| \leq |\Gamma(S) \setminus S|$.

$\Gamma(S) \setminus S$ is exactly the set of vertices that S is connected to. We know that there are at least $\alpha|S|$ edges out of S as G is α-edge-expander and as each node has at most d incoming edges, this implies that $\frac{\alpha}{d}|S| \leq |\Gamma(S) \setminus S|$, which is what we wanted to show. □

Lemma 16 (Diameter of an α-expander)**.** *If $G = (\mathcal{V}, E)$ is an α-expander then the diameter of G is at most $2 \cdot \frac{\log\left(\frac{|\mathcal{V}|}{2}+1\right)}{\log(\alpha)}$.*

Proof. Consider any two vertices $v, u \in \mathcal{V}$. The set of vertices that is at most distance t from v is given by $\Gamma^t(v)$. As G is a α-expander, we have $|\Gamma^t(v)| \geq \min(\frac{|\mathcal{V}|}{2}+1, \alpha^t)$. Hence, we have $|\Gamma^t(v)| \geq \frac{|\mathcal{V}|}{2}+1$ if

$$\alpha^t \geq \frac{|\mathcal{V}|}{2}+1 \quad \iff \quad t \geq \frac{\log\left(\frac{|\mathcal{V}|}{2}+1\right)}{\log(\alpha)}. \tag{79}$$

We now analogously consider the set of nodes that can reach u in t steps. This set is given by $\Gamma^t(u)$. As G is an α-expander the size of this set is $|\Gamma^t(v)| \geq \min(\frac{|\mathcal{V}|}{2}+1, \alpha^t)$ which again implies that when Equation (79) is satisfied, we have $|\Gamma^t(v)| \geq \frac{|\mathcal{V}|}{2}+1$. This implies that for t satisfying Equation (79), we have that $\Gamma^t(v) \cap \Gamma^t(u) \neq \varnothing$ and the distance from v to u is therefore at most $2 \cdot \frac{\log\left(\frac{|\mathcal{V}|}{2}+1\right)}{\log(\alpha)}$. □

Lemma 17 (Edge-expansion of Erdős–Rényi graphs)**.** *Let $n \in \mathbb{N}$, $\epsilon > 0$, $d \coloneqq \log^{\epsilon+1}(n)$, $\rho \coloneqq \frac{d}{n}$, $\alpha \in [0, 1-\frac{1}{\sqrt{2}}]$, and $c \coloneqq \frac{1}{4}+\frac{\alpha^2}{2}-\alpha$. The probability that a graph $G = (\mathcal{V}, E) \xleftarrow{\$} \mathbb{G}(n, \rho)$ is not a αd-edge-expander is less than $\frac{n^{-c\log^\epsilon(n)+1}}{1-n^{-c\log^\epsilon(n)+1}}$.*

Proof. We look at a subset of vertices $S \subseteq \mathcal{V}$ where $r \coloneqq |S| \leq \frac{n}{2}$ and let $\overline{S} \coloneqq V \setminus S$. Let us introduce a random variable for each pair of nodes $v \in S$ and $u \in \overline{S}$, $X_{v,u}$ which indicates if there is an $\{u, v\} \in E$. For each such variable $X_{v,u}$ we have

$$\mathbb{E}[X_{v,u}] = \rho. \tag{80}$$

Furthermore, the expected amount of edges between S and $\overline{S}$ is

$$\mathbb{E}\left[\sum_{v\in S, u\in\overline{S}} X_{v,u}\right] = \sum_{v\in S, u\in\overline{S}} \mathbb{E}[X_{v,u}] = \rho|S||\overline{S}|. \tag{81}$$

Chernoff (Lemma 1) gives us that for $\delta \in [0, 1]$

$$\Pr\left[\sum_{v\in S, u\in\overline{S}} X_{v,u} \leq (1-\delta)\rho|S||\overline{S}|\right] \leq e^{-\frac{\delta^2\rho|S||\overline{S}|}{2}}. \tag{82}$$

We let $\alpha \in [0, \frac{1}{2})$ and let $\delta \coloneqq 1-\frac{\alpha}{d(n-r)} \in [0, 1]$ (as $n - r \geq \frac{n}{2}$) which implies that

$$\begin{aligned}
\Pr\left[\sum_{v\in S, u\in\overline{S}} X_{v,u} \leq \alpha d|S|\right] &\leq e^{-\frac{\left(1-\frac{\alpha n}{(n-r)}\right)^2 dr(n-r)}{n\cdot 2}} \\
&= e^{-\left(\frac{\left(1-\frac{\alpha n}{(n-r)}\right)^2 (n-r)}{n\cdot 2}\right)^{dr}} \\
&= e^{-\left(\frac{n-r}{2n}+\frac{\alpha^2 n}{2(n-r)}-\alpha\right)^{dr}}.
\end{aligned} \tag{83}$$

As $r \leq \frac{n}{2}$ we have

$$\frac{n-r}{2n}+\frac{\alpha^2 n}{2(n-r)-\alpha} \geq \frac{1}{4}+\frac{\alpha^2}{2}-\alpha =: c. \tag{84}$$

If $\alpha < 1 - \frac{1}{\sqrt{2}}$ this is larger than 0 and we have

$$\Pr\left[\sum_{v\in S, u\in\overline{S}} X_{v,u} \leq \alpha d|S|\right] \leq e^{-c^{dr}}. \tag{85}$$

We now bound the probability that any such $S \subseteq \mathcal{V}, |S| \leq \frac{n}{2}$ exists by union bound

$$\begin{aligned}
&\Pr[\text{exists } S \subseteq \mathcal{V} \wedge |S| \leq \frac{n}{2} \wedge \sum_{v\in S, u\in\overline{S}} X_{v,u} \leq \alpha d|S|] \\
&\leq \sum_{r=1}^{\frac{n}{2}} \sum_{S,|S|=r} \Pr\left[\sum_{v\in S, u\in\overline{S}} X_{v,u} \leq \alpha d|S|\right] \\
&\leq \sum_{r=1}^{\frac{n}{2}} \sum_{S,|S||=r} e^{-c^{dr}} \\
&\leq \sum_{r=1}^{\frac{n}{2}} \binom{n}{r} e^{-c^{dr}}.
\end{aligned} \tag{86}$$

Now for $d \coloneqq \log^{1+\epsilon}(n)$ we have $\left(e^{-c}\right)^{\log^{1+\epsilon}(n)} = n^{-c\log^{\epsilon}(n)}$. Using that $\binom{n}{r} \leq n^r$ this implies

$$\begin{aligned}
&\Pr[\text{exists } S \subseteq V \wedge |S| \leq \frac{n}{2} \wedge \sum_{v\in S, u\in\overline{S}} X_{v,u} \leq \alpha d|S|] \\
&\leq \sum_{r=1}^{\frac{n}{2}} \binom{n}{r} \left(n^{-c\log^{\epsilon}(n)}\right)^r \\
&\leq \sum_{r=1}^{\frac{n}{2}} n^r \left(n^{-c\log^{\epsilon}(n)}\right)^r \\
&= \sum_{r=1}^{\frac{n}{2}} \left(n^{-c\log^{\epsilon}(n)+1}\right)^r \\
&= \sum_{r=0}^{\frac{n}{2}-1} \left(n^{-c\log^{\epsilon}(n)+1}\right)^{r+1} \\
&= \sum_{r=0}^{\frac{n}{2}-1} n^{-c\log^{\epsilon}(n)+1} \left(n^{-c\log^{\epsilon}(n)+1}\right)^r \\
&< \frac{n^{-c\log^{\epsilon}(n)+1}}{1 - n^{-c\log^{\epsilon}(n)+1}}.
\end{aligned} \tag{87}$$

The last inequality holds due to the convergence of a geometric sum. □

Lemma 18 (Vertex-expansion of Erdős–Rényi graphs)**.** *Let $n \in \mathbb{N}$, $\epsilon > 0$, $d \coloneqq \log^{\epsilon+1}(n)$, $\rho \coloneqq \frac{d}{n}$, $\alpha \in [0, 1 - \frac{1}{\sqrt{2}}]$, $c \coloneqq \frac{1}{4} + \frac{\alpha^2}{2} - \alpha$, and $\delta \in [0, 1]$. The probability that a graph $G = (\mathcal{V}, E) \xleftarrow{\$} \mathbb{G}(n, \rho)$ is not a $\left(1 + \frac{\alpha}{1+\delta}\right)$-expander is less than*

$$\frac{n^{-c\log^{\epsilon}(n)+1}}{1 - n^{-c\log^{\epsilon}(n)+1}} + n^{-\frac{\delta^2\log^{\epsilon}(n)}{3}+1}. \tag{88}$$

Proof. Lemma 17 ensures that G is an αd-edge-expander unless with

$$\frac{n^{-c\log^{\epsilon}(n)+1}}{1-n^{-c\log^{\epsilon}(n)+1}}. \tag{89}$$

Lemma 2 ensures that for any $\delta \in [0,1]$ we have the probability

$$\Pr[\max_{v\in\mathcal{V}} \mathtt{deg}(v) \geq (1+\delta)d] \leq n \cdot e^{-\frac{\delta^2 d}{3}} = n \cdot e^{-\frac{\delta^2 \log^{1+\epsilon}(n)}{3}} = n^{-\frac{\delta^2 \log^{\epsilon}(n)}{3}+1}. \tag{90}$$

Using union-bound and Lemma 15 we get that G is an $\left(1+\frac{\alpha}{1+\delta}\right)$-expander except with the desired probability. □

As a consequence of this and Lemma 16 we can conclude that Erdős–Rényi-graphs with an average polylogarithmic degree achieves a logarithmic diameter except with probability negligible in the amount of nodes in the graph.

References

[ACD+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*, pages 317–326. ACM, 2019.

[BCH+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)*, volume 12552 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2020.

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

[BHK20] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*. Cambridge University Press, 2020.

[BHÖ+99] Kenneth P. Birman, Mark Hayden, Öznur Özkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356. Springer, 2017.

[Bol01] Béla Bollobás. *Random graphs*. Number 73 in Cambridge studies in advanced mathematics. Cambridge University Press, 2 edition, 2001.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

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

[CCG+15a] 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 Tim Roughgarden, editor, *ITCS 2015*, pages 153–162, Rehovot, Israel, January 11–13, 2015. ACM.

[CCG+15b] 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*, pages 153–162. ACM, 2015.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

[DGH+87] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In Fred B. Schneider, editor, *6th ACM PODC*, pages 1–12, Vancouver, BC, Canada, August 10–12, 1987. ACM.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 23–41, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany.

[ER60] P. Erdős and A Rényi. On the evolution of random graphs. In *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, pages 17–61, 1960.

[GKKZ11] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In Cyril Gavoille and Pierre Fraigniaud, editors, *30th ACM PODC*, pages 179–186, San Jose, CA, USA, June 6–8, 2011. ACM.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 291–323, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[HHL06] Zygmunt J. Haas, Joseph Y. Halpern, and Li Li. Gossip-based ad hoc routing. *IEEE/ACM Trans. Netw.*, 14(3):479–491, 2006.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In Jaeyeon Jung and Thorsten Holz, editors,

USENIX Security 2015, pages 129–144, Washington, DC, USA, August 12–14, 2015. USENIX Association.

[KJG+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 *2018 IEEE Symposium on Security and Privacy*, pages 583–598, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 477–498. Springer, 2013.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[KSSV00] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *41st FOCS*, pages 565–574, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 705–734. Springer, 2016.

[LNZ+16] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 17–30, Vienna, Austria, October 24–28, 2016. ACM Press.

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

[MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2002.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[Nie03] Jesper Buus Nielsen. *On protocol security in the cryptographic model*. PhD thesis, 2003.

[PS17] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[PSs17] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[Ren19] Ling Ren. Analysis of Nakamoto consensus. Cryptology ePrint Archive, Report 2019/943, 2019. https://eprint.iacr.org/2019/943.

[RT19] Elias Rohrer and Florian Tschorsch. Kadcast: A structured approach to broadcast in blockchain networks. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 199–213. ACM, 2019.

[ZMR18] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. RapidChain: Scaling blockchain via full sharding. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 931–948, Toronto, ON, Canada, October 15–19, 2018. ACM Press.