Churn- and DoS-resistant Overlay Networks Based on Network Reconfiguration

Maximilian Drees
Paderborn University
maxdrees@mail.upb.de

Robert Gmyr
Paderborn University
gmyr@mail.upb.de

Christian Scheideler
Paderborn University
scheideler@upb.de

ABSTRACT

We present three robust overlay networks: First, we present a network that organizes the nodes into an expander and is resistant to even massive adversarial churn. Second, we develop a network based on the hypercube that maintains connectivity under adversarial DoS-attacks. For the DoS-attacks we use the notion of a $\Omega(\log \log n)$-late adversary which only has access to topological information that is at least $\Omega(\log \log n)$ rounds old. Finally, we develop a network that combines both churn- and DoS-resistance. The networks gain their robustness through constant network reconfiguration, i.e., the topology of the networks changes constantly. Our reconfiguration algorithms are based on node sampling primitives for expanders and hypercubes that allow each node to sample a logarithmic number of nodes uniformly at random in $O(\log \log n)$ communication rounds. These primitives are specific to overlay networks and their optimal runtime represents an exponential improvement over known techniques. Our results have a wide range of applications, for example in the area of scalable and robust peer-to-peer systems.

Keywords

Churn; DoS-Attack; Expander; Hypercube; Overlay Network; Random Walks

1. INTRODUCTION

Large-scale and highly decentralized overlay networks have become increasingly popular since the rise of peer-to-peer systems and social networks. Issues like churn (the membership rapidly changes over time) and adversarial attacks pose significant challenges for these networks, so a significant amount of work has been invested in recent years to find effective ways of protecting overlay networks against churn and attacks. A standard approach for this is to continuously refresh the topology of the overlay network so that defects in the network caused by churn or attacks are repaired. Early attempts in practice go back to solutions like JXTA, where at the core the peers continuously exchange random neighbors with their neighbors in an attempt to keep the network well-connected. In theory, it is already known that there are simple rules for randomly switching edges in a local fashion so that eventually a random graph emerges (e.g., [12, 24]). However, the runtime bounds shown so far for these rules are fairly large, and a rigorous analysis showing that such a rule would be sufficient to get to a random graph in polylogarithmically many parallel rounds seems to be out of reach so far, even if there is no churn. An alternative approach is to keep the nodes in some hypercubic topology and to perform load balancing under churn in order to make sure that all places of the hypercubic topology are kept well-occupied by the nodes (e.g., [21]). However, here the best result known so far is a solution that can just tolerate a logarithmic churn per round. Currently, the most promising approach to rigorously handle a high churn is to perform random walks, which is also the approach used in our paper. However, instead of just using random walks in a standard fashion, which would take $\Omega(\log n / \log \log n)$ communication rounds in graphs of polylogarithmic degree in order to sample nodes uniformly at random, we combine random walks with pointer doubling, which *exponentially* improves the runtime needed for random walks to sample nodes uniformly at random. Pointer doubling (i.e., a node introduces its neighbors to its neighbors) is a well-known technique in the area of parallel computing [19], but to our great surprise, it seems that it has never been combined with random walks so far. With our rapid node sampling approach we can significantly improve the state of the art concerning the maximum churn rate that we can allow and the number of nodes that can be blocked by an adversary in each round while preserving connectivity among the remaining nodes.

SPAA '16, July 11 - 13, 2016, Pacific Grove, CA, USA

ISBN 978-1-4503-4210-0/16/07...$15.00

DOI: http://dx.doi.org/10.1145/2935764.2935783

1.1 Model and Problem Statement

We assume that we have a set of nodes V that can potentially change over time, where n denotes the current size of V. Every node v is assumed to have a unique identifier $\mathrm{id}(v)$ of size $O(\log n)$ (which, in reality, is its IP address or MAC address) that has to be known by a node w to send messages to it. The nodes aim at maintaining a topology for their overlay network that can be represented as a bidirected graph $G = (V, E)$, i.e., if $(v, w) \in E$ then also $(w, v) \in E$, where $(v, w) \in E$ requires v to know $\mathrm{id}(w)$. Apart from the ids needed for G, each node v may hold additional ids that it does not currently use for communication, so they do not appear in E. Since G is bidirected, we will henceforth simply consider G to be undirected. An undirected graph $G = (V, E)$ is *connected* if for every pair of nodes $v, w \in V$

there is a path along edges in E from v to w. We consider the synchronous message passing model for our protocols, which means that all nodes operate in synchronized *rounds*. Each round consists of three steps. In the first step, a node v receives all messages sent to it in the previous round, in the second step, it may perform any kind of local computation, and in the third step, it may send out a distinct message to each of its neighbors, where the *neighborhood* of a node v is defined as $N(v) = \{w \in V \mid (v, w) \in E\}$.

We consider the problem of *maintaining connectivity* in overlay networks under *large-scale adversarial churn* or *adversarial DoS-attacks* (or both) using only polylogarithmic communication work for each node in each round, where the *communication work* of a node is defined as the total number of bits that it receives and sends in a round. In the following, we define the exact nature of the attacks we consider.

Adversarial churn. Here, we assume that there is an adversary that for each round i prescribes a set of nodes W_i. However, the network has some flexibility in adapting its node set to the W_i's, as described below. The adversary has a *churn rate* of r if for all i, $|W_i|/r \leq |W_{i+1}| \leq r \cdot |W_i|$. For each node $v \in W_{i+1} \setminus W_i$ (i.e., a *new* node added to the system), we require that it is introduced to exactly one node in $W_i \cap W_{i+1}$ (i.e., a node *staying* in the system), and altogether at most $\lceil r \rceil$ new nodes are introduced to any node in a round. Formally, whenever a node w is introduced to a node v, it means that v learns about $\mathrm{id}(w)$. A node v is *leaving* if $v \in W_i \setminus W_{i+1}$. For simplicity, we assume that every node v is prescribed to enter and leave the system only once (or more precisely, every id can be used at most once). The decisions of the adversary can be based on any information about the past or current state of the system, so we allow it to be omniscient.

We allow the network to adapt its node set to the prescribed node sets within a certain delay T, i.e., the goal is to exclude the leaving nodes from G and to integrate the new nodes into G within T rounds while maintaining connectivity, where T is as small as possible. Let V_i be the set of nodes that are part of the network in round i. Formally, we require for each i that $\bigcap_{j=i-T}^{i} W_j \subseteq V_i \subseteq \bigcup_{j=i-T}^{i} W_j$. The protocols presented in this work all satisfy $T = O(\log \log n)$. Furthermore, we require the network to be monotonic in the sense that there is a unique round i where $v \in V_{i+1} \setminus V_i$ (i.e., v is entering V) and a unique round $j > i$ where $v \in V_j \setminus V_{j+1}$ (i.e., v is excluded from V). Our approach to satisfy these conditions is to keep V fixed over certain periods of time. In addition to that, any new node v introduced to a node w not yet in V will be delegated to the node in V that w was delegated (or introduced) to itself, so that w.l.o.g. we will assume in our protocols that a new node is always introduced to a node in V.

Adversarial DoS-attacks. In this case, we allow an *r-bounded* adversary to block any r-fraction of the current nodes in a round. A blocked node cannot send or receive messages in that round. We assume that a message sent out from node v to node w at round i can only be successfully received and processed by node w if v is non-blocked in round i and w is non-blocked in round i and $i + 1$. In this case, we also call w *available* in round $i + 1$. Note that for a protocol to make progress, it is crucial that nodes available in round i can send messages to nodes available in round $i+1$ for any i. The *only* information about the state of the system that the adversary is allowed to use for its DoS-attacks is the topology of the overlay network, i.e., we do not allow it to inspect messages or the state of nodes or even count the number of messages sent along an edge (which can easily be made uniform in our protocols, so it would not be of any use). An adversary is called *t-late* if it only has access to topological information that is at least t rounds old. We say that an overlay network is connected under a DoS-attack if the network restricted to its non-blocked nodes is still connected. Certainly, an overlay network of degree d cannot maintain connectivity under a r-bounded 0-late DoS-attack as long as $r > d/n$ because it would be very easy for such an adversary to isolate individual nodes.

1.2 Related Work

A central idea used throughout this work is network reconfiguration, i.e., switching the topology of an overlay network to a new topology that is independent of the old topology. Various ways of achieving this have already been studied in the literature. One way is to use random local edge switching rules in order to eventually obtain a random graph (e.g., [7, 12, 24]). However, the runtime bounds shown so far for these rules are fairly large, and a rigorous analysis showing that such a rule would be sufficient to get to a random graph in polylogarithmically many parallel rounds seems to be out of reach so far, even without churn.

A much simpler approach is to use routing or sorting, with the goal of randomly reordering the nodes in an overlay network. To illustrate that, consider the skip graph, which is known to be an expander with high probability [2]. Suppose that the skip graph is formed by nodes with labels that are chosen uniformly at random from $[0, 1]$. Then the formation of a new, independent skip graph can essentially be reduced to a routing problem in the old skip graph where each node v sends a message to the node w closest to a randomly chosen $x \in [0, 1]$, which is supposed to be v's new label. Once the routing has completed, it is easy to use the old skip graph to establish a skip graph on the new labels in altogether just $O(\log n)$ communication rounds. However, with overlay networks of polylogarithmic degree, routing or sorting cannot be done faster than in $o(\log n / \log \log n)$ time while just allowing a polylogarithmic communication work at each node in each round.

There are various other approaches of maintaining connectivity under churn. A standard approach in practice is to use a multi-tier architecture where the older, more stable peers form the actual overlay network while the young peers just connect to one or more of the stable peers (see, e.g., [33] and the references therein). Another way is to keep the nodes in some hypercubic topology and to perform local load-balancing in order to make sure that all places of the hypercubic topology are kept well-occupied by nodes under churn (e.g., [1, 21]). However, just eventual recovery from adversarial churn, or adversarial churn allowing only a logarithmic number of arrivals and departures per round has been considered there. Various other solutions have been proposed in theory that can handle either stochastic churn or adversarial churn that does not cause disconnectivity in a round, either by using a high enough degree or by constraining the adversary (e.g., [5, 16, 18, 26, 29]). Also, self-healing networks (e.g., [27]) have been proposed, but in these networks each insertion or deletion of a constant number of nodes is followed by some process of repairing the network. Finally, self-stabilizing overlay networks can be used to han-

dle churn (e.g., [6, 14, 17]), but no concrete bounds on the churn rate are known for these.

Some of the solutions above cannot just handle churn but also limited-DoS attacks (e.g., [5, 21, 29, 33]). Within our model, DoS-attacks are more severe than churn in a sense that if an adversary decides to block a node, it will be instantly blocked without warning. If the adversary is not aware of the network topology, a standard approach to prevent disconnectivity is to randomly spread nodes in an overlay network and to use redundant connections, which is satisfied by numerous peer-to-peer networks. However, once the adversary knows the topology, nothing can be done to prevent disconnectivity unless the degree (defined as the maximum number of edges into or out of a node) is higher than the maximum number of nodes that the adversary can block at the same time: if an adversary wants to isolate node v, it simply blocks all nodes w that can either send a message to v or receive a message from v. Apart from blocking nodes, also other attacks on the connectivity of an overlay network have been studied like Sybil-attacks [9] and Eclipse-attacks [32]. There have also been several works on protecting distributed hash tables (DHTs) against DoS-attacks from outsiders [20], past insiders [4], or even insiders [10], but for past insiders and insiders the nodes are assumed to form a clique to avoid disconnectivity problems.

In [3] the authors show that under a model similar to ours it is impossible to maintain connectivity when a topology-oblivious adversary can subject $\Omega(\sqrt{n})$ nodes to churn in each round. The key difference between the model in [3] and our model is that in [3] nodes that are prescribed by the adversary to leave the network have to do so *immediately* while we allow the nodes to leave within the next $T = O(\log \log n)$ rounds. This shows that allowing a very small delay is sufficient to handle churn rates that would otherwise be impossible to handle.

Our network reconfiguration algorithms are based on node sampling primitives that combine random walks with pointer doubling. Pointer doubling is a well-known technique in the area of parallel computing [19]. It has originally been applied to rapidly contract trees by each node continuously introducing its current parent as the new parent to its children. In this way, a tree of depth D can be contracted in $O(\log D)$ parallel rounds to a tree of depth 1. Another application of pointer doubling is to rapidly form a clique: if in a graph of diameter D every node continuously introduces its neighbors to each other, then it just takes $O(\log D)$ communication rounds until a clique is formed. However, the communication work per round when using message passing is huge towards the end for both cases. To the best of our knowledge, pointer doubling has not been applied in the context of random walks yet.

The presented node sampling primitives allow each node in a network to sample a logarithmic number of nodes uniformly at random from the set of nodes in the network using only $O(\log \log n)$ communication rounds. The primitives are specific to overlay networks, i.e., they exploit the fact that a node can send a message to any other node whose identifier is known to it. Also, our primitives are restricted to $\mathbb{H}$-graphs (or, more generally, regular multigraphs with appropriate expansion properties) and hypercubes. For the case of arbitrary networks in which only nodes sharing an edge can communicate, Das Sarma et al. [30, 31] provide distributed algorithms for different variations of the problem of creating random walks. For the problem of letting each node in the network sample a logarithmic number of nodes uniformly at random using random walks, we achieve an exponential improvement in running time compared to Das Sarma et al., breaking through the lower bound of Nanongkai et al. [25], and an exponential improvement in message complexity (though it should be mentioned that their work did not aim at minimizing the message complexity).

1.3 Our Contribution

We develop three overlay networks: First, we present a network that organizes the nodes into an expander and maintains connectivity under adversarial churn by an omniscient adversary with constant churn rate. Second, we develop a network based on the hypercube that maintains connectivity under DoS-attacks by a $(1/2 - \varepsilon)$-bounded $\Omega(\log \log n)$-late adversary for any $0 < \varepsilon \leq 1/2$. We finally extend this approach to get a network that maintains connectivity under DoS-attacks and adversarial churn by a $(1/2 - \varepsilon)$-bounded $\Omega(\log \log n)$-late adversary with churn rate $\gamma^{1/\Theta(\log \log n)}$ for any $0 < \varepsilon \leq 1/2$ and any constant γ. All of our overlay networks have polylogarithmic degree and require polylogarithmic communication work for each node in each round. Our algorithms are based on novel node sampling primitives for expanders and hypercubes that allow each node to sample a logarithmic number of nodes uniformly at random in $O(\log \log n)$ communication rounds. These primitives are specific to overlay networks and their optimal runtime represents an exponential improvement over known techniques. Our results have a wide range of applications in the area of scalable and robust P2P-networks and beyond; we outline some applications in a dedicated section. Due to space constraints, most of the proofs have been omitted.

2. PRELIMINARIES

Before we get to the main part of this work, we introduce some basic tools, network topologies, and node sampling techniques for overlay networks.

2.1 Chernoff Bounds

We extensively use the following bounds which are known as *Chernoff bounds*.

LEMMA 1. *Suppose that $X_1, \ldots, X_n$ are independent binary random variables, let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathrm{E}[X]$. Then it holds for all $\delta > 0$ that*

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\min\{\delta^2, \delta\} \cdot \mu/3}.$$

Furthermore, it holds for all $0 < \delta < 1$ that

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\delta^2 \mu/2}.$$

2.2 Network Topologies

We use two different network topologies throughout this work, namely the well-known *hypercube* and a graph class we refer to as $\mathbb{H}$-*graphs*, following the notation used in [22]. A d-dimensional hypercube is an undirected, simple graph $G = (V, E)$ where $V = \{0,1\}^d$ is the set of all binary d-tuples and E is such that two vertices are connected if and only if they differ in exactly one coordinate. Our definition of $\mathbb{H}$-graphs slightly differs from the definition given in [22]. Let $d \geq 8$ be an even constant. An $\mathbb{H}$-graph is an undirected multi-graph $G = (V, E)$ where $E = \bigcup_{i=1}^{d/2} C_i$ such

that each C_i is a set of edges that forms a Hamilton cycle over the nodes in V.[1] Each C_i has an orientation, that is, a node u stores a reference to its predecessor and its successor in C_i. Note that a $\mathbb{H}$-graph is a connected d-regular multi-graph that can have parallel edges but no loops. We refer to the set of all possible $\mathbb{H}$-graphs over n nodes as $\mathbb{H}_n$. We can sample a graph from $\mathbb{H}_n$ by choosing the Hamilton cycles $C_1, C_2, \ldots, C_{d/2}$ independently and uniformly at random from the set of all possible Hamilton cycles over V.

2.3 Node Sampling

Our algorithms heavily rely on the ability of nodes to efficiently sample nodes uniformly at random from a network. In the d-dimensional hypercube we can achieve uniform node sampling using the following well-known random walk technique: A node u in the hypercube creates a token containing $\mathrm{id}(u)$ that traverses the graph for d rounds. Let $v = (b_1, \ldots, b_d)$ be the node that holds this token in round i. The node v flips a fair coin. If the coin comes up tails, v keeps the token; otherwise v forwards the token to the neighboring node $n_i(v) := (b_1, \ldots, b_{i-1}, 1 - b_i, b_{i+1}, \ldots, b_d)$. The node w that holds the token at the end of the random walk sends $\mathrm{id}(w)$ to u. It is easy to see that w is chosen uniformly at random from V. Therefore, a node in a d-dimensional hypercube can perform uniform node sampling in $O(\log n)$ rounds.

For $\mathbb{H}$-graphs we can also use a random walk to perform node sampling with a probability distribution that deviates only slightly from the uniform distribution over V. For a given n and $d \geq 8$ let G be a graph chosen uniformly at random from $\mathbb{H}_n$. Consider the simple random walk over G, that is, at a node u the random walk chooses an edge incident to u uniformly at random and then moves along that edge to another node. Since G is a d-regular graph, the stationary distribution of this random walk is the uniform distribution over V. Furthermore, for a graph G chosen uniformly at random from $\mathbb{H}_n$ we have the following theorem shown by Friedman [15].

THEOREM 1 (FRIEDMAN [15]). *For any fixed real $\epsilon > 0$ there is a constant c such that*

$$\Pr\left[\ |\lambda_i| \leq 2\sqrt{d-1} + \epsilon \ \ \forall i > 1\ \right] \geq 1 - \frac{c}{n^\tau}$$

where λ_i are the eigenvalues of the adjacency matrix of G in descending order and $\tau = \lceil \sqrt{d-1} \rceil - 1$.

Theorem 1 implies the following corollary.

COROLLARY 1. *For d and n sufficiently large we have $|\lambda_i| \leq 2\sqrt{d}$ for all $i > 1$, w.h.p.*[2]

By Corollary 1, G is an expander with diameter $O(\log n)$, w.h.p. As a result, the simple random walk on G is rapidly mixing, that is, it quickly approaches the stationary distribution. Formally, we have the following lemma, which is a consequence of Theorem 5.1 in [23].

LEMMA 2. *Let G be a d-regular multi-graph with $|\lambda_i| \leq 2\sqrt{d}$ for all $i > 1$ and let $t = \lceil 2\alpha \log_{d/4} n \rceil$ for any constant $\alpha \geq 1$. Consider the simple random walk of length t starting at a node u. Then for any node $v \in V$, we have*

$$\left| \Pr[\textit{random walk ends at } v] - \frac{1}{n} \right| \leq n^{-\alpha}.$$

We refer to the probability distribution given in Lemma 2 as *almost uniform*. If n and d are sufficiently large, a node in a random $\mathbb{H}$-graph can perform almost uniform node sampling in $O(\log n)$ communication rounds, w.h.p. Lemma 2 immediately implies the following lemma, which allows us to ignore the imperfectness of the random walks in our applications to overlay networks.

LEMMA 3. *For any set of k independent random walks of length $t = \lceil 2\alpha \log_{d/4} n \rceil$ with $\alpha > 2$ and any $v_1, \ldots, v_k \in V$ we have*

$$\left| \Pr[\textit{for all } i, \textit{ random walk } i \textit{ ends in } v_i] - \frac{1}{n^k} \right| \leq 2n^{2-\alpha} \cdot \frac{1}{n^k}.$$

3. RAPID NODE SAMPLING

In this section, we show how pointer doubling can be used to achieve an exponential speed-up in the random walk techniques for node sampling presented in Section 2.3. Our goal is to let each node in a network sample at least $\beta \log n$ nodes uniformly and independently at random from the set of all nodes in the network for any given constant β. The presented *rapid node sampling* primitives solve this problem in $O(\log \log n)$ communication rounds for both $\mathbb{H}$-graphs and hypercubes. The following lemma implies that this is optimal.

LEMMA 4. *Let $G = (V, E)$ be a graph with diameter d. Any algorithm that allows a node u to sample a node from V uniformly at random requires $\Omega(\log d)$ rounds.*

PROOF. Let $u, v \in V$ be such that the shortest path between the nodes is d. Consider the following algorithm: In each round every node mutually introduces all its neighbors by sending all references it holds to all nodes it knows. This algorithm requires $\Omega(\log d)$ rounds to introduce v to u. Clearly, no algorithm can introduce v to u faster than this algorithm. Hence, for any algorithm that samples a node from V and stores it at u in $o(\log d)$ rounds we have $\Pr[u \text{ chooses } v] = 0$. Therefore, no algorithm that uses $o(\log d)$ rounds can perform uniform node sampling over V. □

Throughout this section we assume that all nodes know n. However, when we use the primitives in the following sections, we explain how to circumvent this assumption.

3.1 H-Graphs

In our rapid node sampling primitives, the nodes collaborate to construct random walks of length $\Theta(\log n)$ in $O(\log \log n)$ rounds. Each node locally executes the algorithm shown in Algorithm 1. The algorithm is divided into four phases. While Phase 1 is executed only once, the remaining phases are executed in a loop. The execution of a phase corresponds to one communication round. The variable T in line 5 determines the length of the generated random walks. To generate random walks of length at least $\lceil 2\alpha \log_{d/4} n \rceil$, we choose $T = \lceil \log(2\alpha \log_{d/4} n) \rceil \leq \log \log n + O(1)$. Each node uses a multiset M holding ids of

[1] The union in the definition of the edge set E is a multi-set union.

[2] We write *w.h.p.* (i.e., *with high probability*) to refer to an event that occurs with probability at least $1 - n^{-c}$ for any constant $c > 1$.

nodes. During the execution of the algorithm, M is repeatedly emptied and then filled to a certain size. Specifically, the size of M before the first iteration of the loop in line 5 is defined as m_0 and the size of M after the i-th iteration of the loop is defined to be m_i. After the execution of the algorithm, M contains ids that were chosen uniformly at random from V.

Algorithm 1 Rapid Node Sampling in $\mathbb{H}$-Graphs

```
    Phase 1:
 1: M ← ∅
 2: for j ← 1 to m_0 do
 3:     choose neighbor v uniformly at random
 4:     M ← M ∪ {v}
 5: for i ← 1 to T do
    Phase 2:
 6:     for j ← 1 to m_i do
 7:         choose and remove v ∈ M uniformly at random
 8:         send request to v
    Phase 3:
 9:     receive k requests from nodes u_1, u_2, ..., u_k
10:     for j ← 1 to k do
11:         choose and remove v ∈ M uniformly at random
12:         send response v to u_j
    Phase 4:
13:     receive responses v_1, v_2, ..., v_{m_i}
14:     M ← {v_1, v_2, ..., v_{m_i}}
```

The correctness of the algorithm depends on the choice of the sizes m_i: an inappropriate choice might lead to M being empty when an element should be extracted from it in line 7 or 11. We say the algorithm *succeeds for node v in iteration i* if in the i-th iteration of the loop in line 5 during the execution of the algorithm by v, M is never empty when an element is extracted. We say the algorithm *succeeds* if it succeeds for all nodes and for all iterations. Before we turn to the choice of the m_i, we show that the algorithm is correct under the assumption that it succeeds. Specifically, we show that after the execution of the algorithm, M contains nodes that were chosen by following independent random walks of length at least $\lceil 2\alpha \log_{d/4} n \rceil$.

LEMMA 5. *The algorithm satisfies the following invariant: After the i-th iteration of the loop in line 5, M contains ids of nodes that were chosen by following independent random walks of length 2^i.*

In Lemma 7 we establish a choice for the m_i such that the algorithm succeeds, w.h.p., and it holds $m_T \geq \beta \log n$ for any given β. The proof of Lemma 7 requires the following auxiliary lemma.

LEMMA 6. *Consider a regular undirected multi-graph $G = (V, E)$. Assume that each node in V creates $k \in \mathbb{N}$ tokens. The tokens traverse G in synchronous rounds according to a simple random walk. For $v \in V$ and $t \in \mathbb{N}$ let X be the number of tokens at node v after t rounds. Then X is a sum of independent binary random variables and $\mathrm{E}[X] = k$.*

LEMMA 7. *For any $0 < \varepsilon \leq 1$ we can choose a constant $c \geq \beta$ such that with $m_i = (2+\varepsilon)^{T-i} c \log n$, the algorithm succeeds, w.h.p.*

PROOF. Consider the i-th iteration of the algorithm in the execution by a node v. Let X_i denote the number of requests received by v in Phase 3 (denoted by k in the pseudo-code). It is easy to see that the algorithm succeeds for v in iteration i if $m_{i-1} \geq m_i + X_i$. By the definition of m_i we have

$$\begin{aligned}\Pr[m_i + X_i \geq m_{i-1}] &= \Pr[m_i + X_i \geq (2+\varepsilon)\, m_i] \\ &= \Pr[X_i \geq (1+\varepsilon)\, m_i].\end{aligned}$$

By Lemma 6, X_i is a sum of independent binary random variables and we have $\mathrm{E}[X_i] = m_i$. Therefore, we can apply Chernoff bounds to get

$$\Pr[X_i \geq (1+\varepsilon)\, m_i] \leq e^{-\varepsilon^2 m_i/3} \leq e^{-\varepsilon^2 c \log n/3},$$

where the second inequality holds because $m_i \geq c \log n$. The lemma follows by applying the union bound. □

The following theorem summarizes our analysis of Algorithm 1.

THEOREM 2. *For any $0 < \varepsilon \leq 1$ and any β, the algorithm samples at least $\beta \log n$ nodes almost uniformly at random from V in $O(\log \log n)$ rounds, and the communication work for every node in every round is $O(\log^{2+\log(2+\varepsilon)} n)$, w.h.p.*

PROOF. The correctness of the algorithm follows by Lemmas 5 and 7. The upper bound on the number of rounds immediately follows from our choice of T. Since $m_{i+1} \leq m_i$, the number of identifiers sent or received by a node in every round is at most $m_0 = (2+\varepsilon)^T c \log n = O(\log^{1+\log(2+\varepsilon)} n)$ which immediately implies the given bound on the communication work. □

Note that the presented primitive does not use any properties of $\mathbb{H}$-graphs aside from their regularity and their expansion. Accordingly, the presented primitive works for arbitrary regular graphs with appropriate expansion properties.

3.2 Hypercube

The rapid node sampling algorithm for the hypercube and its analysis are analogous to the above, though technically slightly more involved. We only provide the algorithm and the main results. For ease of presentation we assume that $d = 2^k$ for some $k \in \mathbb{N}$. Each node of the hypercube executes the algorithm given in Algorithm 2. In the pseudo-code we refer to the node executing the algorithm as u.

LEMMA 8. *Consider the execution by a node u. The following invariant holds: After the i-th iteration of the loop in line 8 for any $1 \leq j \leq \log n$ such that $j \equiv 1 \mod 2^i$ we have that for any node $v \in M_j$ the coordinates $j, \ldots, j + 2^i - 1$ of v were chosen independently and uniformly at random while the remaining coordinates of v are identical to the corresponding coordinates of u.*

LEMMA 9. *For $0 < \varepsilon \leq 1$ we can choose a constant $c \geq \beta$ such that with $m_i = (1+\varepsilon)^{\log\log n - i} c \log n$, the algorithm succeeds, w.h.p.*

THEOREM 3. *For any $0 < \varepsilon \leq 1$ and any β, the algorithm samples at least $\beta \log n$ nodes uniformly at random from V in $O(\log \log n)$ rounds, and the communication work for every node in every round is $O(\log^{2+\log(1+\varepsilon)} n)$, w.h.p.*

Algorithm 2 Rapid Node Sampling in the Hypercube

Phase 1:
1: **for** $j \leftarrow 1$ to $\log n$ **do**
2: $M_j \leftarrow \emptyset$
3: **for** $k \leftarrow 1$ to m_0 **do**
4: **if** coin flip comes up heads **then**
5: $M_j \leftarrow M_j \cup \{n_j(u)\}$ ▷ see Section 2
6: **else**
7: $M_j \leftarrow M_j \cup \{u\}$
8: **for** $i \leftarrow 1$ to $\log\log n$ **do**

Phase 2:
9: **for** $j \leftarrow 1$ to $\log n$ with step-size 2^i **do**
10: **for** $k \leftarrow 1$ to m_i **do**
11: choose and remove $v \in M_k$ unif. at random
12: send request (u, j) to v

Phase 3:
13: receive requests $(u_1, j_1), \ldots, (u_\ell, j_\ell)$
14: **for** $k \leftarrow 1$ to ℓ **do**
15: choose and remove $v \in M_{j_k+2^{i-1}}$ unif. at random
16: send response (v, j_k) to u_k

Phase 4:
17: **for** $j \leftarrow 1$ to $\log n$ **do**
18: $M_j \leftarrow \emptyset$
19: receive responses $(u_1, j_1), \ldots, (u_\ell, j_\ell)$
20: **for** $k \leftarrow 1$ to ℓ **do**
21: $M_{j_k} \leftarrow M_{j_k} \cup \{u_k\}$

4. MAINTAINING CONNECTIVITY UNDER ADVERSARIAL CHURN

The rapid node sampling primitives will be our primary tools for the rest of the paper. In this section, we use rapid node sampling for the fast reconfiguration of an overlay network based on the $\mathbb{H}$-graph topology. The new network structure is chosen uniformly at random. Our reconfiguration process only takes $O(\log\log n)$ communication rounds, enabling the nodes to handle continuous adversarial churn with any constant churn rate.

We present the network reconfiguration algorithm shown in Algorithm 3. Executed locally by each node, it transforms an existing Hamilton cycle into a new one, which is chosen uniformly at random. To reconfigure the whole overlay network, every node executes $d/2$ independent instances of the algorithm simultaneously, one instance for each Hamilton cycle. As soon as the reconfiguration is finished, a new iteration begins. As a result, the network structure changes periodically. For the rest of this section, unless otherwise noted, we always consider the execution of the algorithm with respect to a fixed Hamilton cycle. Similar to the rapid node sampling primitive, we split the algorithm into phases for ease of presentation. This time, however, a phase may take multiple communication rounds. Our approach requires the nodes to have some knowledge of the size of the network. Specifically, we need estimates on $\log\log n$ and $\log n$ (for example, both are required for the rapid node sampling primitive for $\mathbb{H}$-graphs, see Section 3.1, which is used in our approach). Since polynomial changes in n just cause additive changes in $\log\log n$ and for all realistic values of n, $\log\log n$ is very small, we assume for simplicity that the nodes have an upper bound k on $\log\log n$ that is precise up to some additive deviation of at most some constant c (i.e., $k - c \leq \log\log n \leq k$). Accordingly, we can use 2^k as an estimate for $\log n$ that is precise up to a constant multiplicative factor.

Algorithm 3 Network Reconfiguration

Phase 1:
1: choose node $u \in V$ via rapid node sampling primitive
2: send own id to u

Phase 2:
3: $U \leftarrow$ set of all received ids
4: $m \leftarrow |U|$
5: **if** $U \neq \emptyset$ **then**
6: $(u_1, u_2, \ldots, u_m) \leftarrow$ unif. random permutation of U

Phase 3:
7: **if** $U \neq \emptyset$ **then**
8: send u_1 to closest active predecessor
9: send u_m to closest active successor
10: receive u_0 from closest active predecessor
11: receive u_{m+1} from closest active successor

Phase 4:
12: **for** $i = 1$ to m **do**
13: send (u_{i-1}, u_{i+1}) to u_i
14: receive (u_p, u_s), predecessor $\leftarrow u_p$, successor $\leftarrow u_s$

Algorithm 3 reconfigures the network without considering churn. Phase 1 of the algorithm relies on rapid node sampling to randomly choose a node from the network. If a node u is chosen in Phase 1 by at least one node, we call u *active* in the respective Hamilton cycle. At the end of the reconfiguration algorithm, each node receives its new neighbors in a randomly chosen Hamilton cycle. We extend the reconfiguration algorithm to handle churn in the following way: When a new node v joins the network, it is introduced to a node w. In Phase 1 of the next reconfiguration, w sends $\mathrm{id}(v)$ to a randomly chosen node in the network for each node v that was introduced to w (along with its own id to another random node as usual). As a result, at the end of the reconfiguration each new node receives the ids of its neighbors in the newly build Hamilton cycle. We are going to show that our algorithm takes $O(\log\log n)$ communication rounds to reconfigure the network. Therefore, the number of new nodes being introduced to w is at most $r^{c\cdot\log\log n} = (\log n)^{c\cdot\log r}$ for some constant c. To provide a sufficiently large number of random nodes in Phase 1, polylogarithmically many instances of the rapid node sampling primitive have to be executed in parallel. If a node w wants to leave the network, it simply skips sending its own id to a random node in Phase 1 of the next reconfiguration. Therefore, w will not be incorporated in the newly build cycle. However, w still has to take part in the reconfiguration which includes sending the ids of nodes that were introduced to w before the adversary determined that w should leave the network. Once the reconfiguration is complete, w can leave the network.

To prove the utility of our algorithm, we first show that the new topology is chosen uniformly at random from the set $\mathbb{H}_n$, provided n is the number of nodes after reconfiguration.

LEMMA 10. *Consider Hamilton cycles C, C_1, and C_2 with $|V_{C_1}| = |V_{C_2}|$ and denote the probability that Algorithm 3 reconfigures C into C_j with $\Pr_C[C_j]$. We have $\Pr_C[C_1] = \Pr_C[C_2]$.*

Next, we prove that the total congestion caused by all instances of Algorithm 3 is small. Since the number of Hamil-

ton cycles is constant, it suffices to bound the congestion for a single cycle.

LEMMA 11. *For a fixed Hamilton cycle and n sufficiently large, the number of times a node is chosen in Phase 1 is at most polylogarithmic, w.h.p.*

To realize Phase 3 efficiently, we use pointer doubling to quickly bridge the distance between active nodes. We have to show that these distances are logarithmic in n, so that Phase 3 can be executed in $O(\log\log n)$ communication rounds. For this, we introduce the concept of *empty segments*. For an Hamilton cycle C and $v \in V_C$, denote the successor of v with respect to the orientation of C as $\text{succ}(v)$. For $u, v \in V_C$, let $[u, v] := \{u, \text{succ}(u), \text{succ}(\text{succ}(u)), \ldots, v\}$. We call $[u, v]$ an empty segment if $\text{succ}(v)$ is active but no node in $[u, v]$ is active.

LEMMA 12. *For a fixed Hamilton cycle, the size of the largest empty segment is at most polylogarithmic, w.h.p.*

Finally, we show that the complete reconfiguration process is asymptotically as fast the rapid node sampling method it relies upon.

LEMMA 13. *Algorithm 3 terminates after $O(\log\log n)$ communication rounds.*

We now state the main results of this section.

THEOREM 4. *Let $G \in \mathbb{H}_n$ and $G' \in \mathbb{H}_m$, assuming m is the number of nodes which want to join or remain in the network. Algorithm 3 reconfigures G into G' with probability $1/|\mathbb{H}_m|$ in $O(\log\log n)$ communication rounds. The communication work is polylogarithmic, w.h.p.*

By Theorem 4, the continuous reconfiguration based on Algorithm 3 correctly transforms the network into a new $\mathbb{H}$-graph every $O(\log\log n)$ rounds, even under churn. This implies the following Theorem.

THEOREM 5. *The given procedure maintains the connectivity of the network under adversarial churn by an omniscient adversary with any constant churn rate, w.h.p.*

5. MAINTAINING CONNECTIVITY UNDER ADVERSARIAL DOS-ATTACKS

Next we present an overlay network that maintains connectivity under constantly changing DoS-attacks. In this section we assume that the number of nodes n is fixed; we lift this restriction in Section 6. We consider a $(1/2 - \varepsilon)$-bounded $\Omega(\log\log n)$-late adversary and our goal is to maintain connectivity among the non-blocked nodes.

In our approach, we organize the nodes into a topology that is derived from the d-dimensional hypercube where d is the largest integer such that $2^d \leq n/(c\log n)$ for some constant c that is chosen throughout the analysis. We denote the number of nodes in the hypercube as $N = 2^d$. To distinguish the nodes of the hypercube from the physical nodes in the network, we refer to them as *supernodes*. For each supernode x there is a set of nodes $R(x)$ that we call the *group of representatives* (or just *group*) of x and that has logarithmic size in n. Each node is part of exactly one group. We say that two groups are neighbors if their corresponding supernodes are neighbors. The exact topology of our network generally depends on which nodes are blocked. However, if no node is blocked, it has the following simple structure: The nodes in a group are connected to a clique, and nodes of neighboring groups form a complete bipartite graph. Our goal is to make it difficult for the adversary to block a large fraction of any group. To achieve this, we randomly rebuild the groups every $\Theta(\log\log n)$ rounds so that the $\Omega(\log\log n)$-late adversary never knows which nodes currently form a group. We assume that initially the network is such that each node chose its group independently and uniformly at random and that the nodes are connected as described above. We then rebuild the groups as follows.

First, the groups simulate the rapid node sampling primitive for their respective supernodes in the hypercube. We denote the state of the rapid node sampling primitive for a supernode x as $S(x)$. $S(x)$ contains all variables used by the primitive and a counter that represents the current iteration and phase. $S(x)$ further contains references to all nodes in $R(x)$ as well as reference to all nodes in $R(y)$ for every supernode y stored in x. The simulation proceeds in *steps*. Each step consists of two rounds and corresponds to one round of the rapid node sampling primitive for the supernodes. In addition to the simulation described below, each available node sends $S(x)$ to all nodes in $R(x)$ in every round. This ensures that newly available nodes can join the simulation and guarantees that as long as there is at least one available node per group in each round the network stays connected. Initially, all nodes in $R(x)$ know the initial state of x. Afterwards, each step is executed as follows for each supernode x.

- **Simulation round**: First, all available nodes in $R(x)$ receive the messages that x is supposed to receive in that round. Afterwards, each available node v simulates the local computation of x and sends a message m_v to all nodes in $R(x)$ containing the new state of x from the viewpoint of v including all messages that x is supposed to send out in that round. Note that m_v might differ because of different random decisions.

- **Synchronization round**: Every available node u in $R(x)$ first receives all messages sent out in the simulation round and chooses among these the message m_v of the node v of lowest $\text{id}(v)$. It then adopts the new state of x as given in m_v and for each message m that x is supposed to send out to some node in y, u sends m to all nodes in $R(y)$.

The following lemma is easy to show by induction on the number of rounds.

LEMMA 14. *If every $R(x)$ contains at least one available node in every round, the nodes can correctly simulate the rapid node sampling primitive for their respective supernodes.*

Hence, it follows from Theorem 3 that at the end of the simulation $S(x)$ contains $\beta\log n$ many supernodes (represented by their groups) that were sampled uniformly at random. Let $R(x) = \{v_1, v_2, \ldots, v_k\}$, where the nodes are ordered according to their ids, i.e., $\text{id}(v_1) < \text{id}(v_2) < \ldots < \text{id}(v_k)$, and let $x_1, x_2, \ldots, x_k$ be the first k supernodes that were sampled. In a final phase, these supernodes are used in the

following way to reorganize the groups, given that $|R(x)| \leq \beta \log n$ for each x.

- First, every available node in $R(x)$ assigns v_i to $R(x_i)$ for every i by sending out messages to all nodes in $R(x_i)$ informing them about v_i.
- Afterwards, every available node in $R(x)$ collects all nodes sent to it from the previous round and sets them to $R'(x)$, the new group representing supernode x. It then sends $R'(x)$ to all nodes in $R(x)$ as well as all nodes in $R(y)$ where y is a neighbor of x.
- After collecting $R'(x)$ and $R'(y)$ for all neighbors y of x, each available node in $R(x)$ sends $R'(x)$ and $R'(y)$ for all neighbors y of x to all nodes in $R'(x)$.
- Each available node v collects $R'(x')$ of the new supernode x' that was assigned to v and also $R'(y)$ for all neighbors y of x'.

Note that at the end of the last round, all available nodes know their new supernode x in form of references to the nodes in $R(x)$ as well as the new groups of the neighbors of x, given that every old group always had at least one available node. We therefore have the following lemma.

LEMMA 15. *If $|R(x)| \leq \beta \log n$ and $R(x)$ contains at least one available node in every round for every supernode x, then the nodes can correctly reconfigure the network by assigning nodes to groups uniformly at random.*

We now turn to the analysis of the given reconfiguration procedure. First we establish bounds for the sizes of the groups.

LEMMA 16. *For any $0 < \delta < 1$ we can choose a constant c such that, w.h.p., for every supernode x it holds $(1-\delta)c \log n \leq (1-\delta)n/N < |R(x)| < (1+\delta)n/N < 2(1+\delta)c \log n$.*

Based on Lemma 16 and Theorem 3 it is easy to verify that the reconfiguration procedure outlined above requires $\Theta(\log \log n)$ rounds and polylogarithmic communication work per node in each round. Now let t be the exact number of rounds required by the procedure. A $2t$-late adversary has no knowledge about the current organization of nodes into groups. In fact, in the exact round in which the adversary would gain first information about the execution of the reconfiguration procedure that led to the current composition of the groups, a new reconfiguration of the groups finishes. Therefore, the current assignment of nodes to groups is random from the perspective of the adversary. This gives us the following lemma.

LEMMA 17. *For any $0 < \varepsilon \leq 1/2$ we can choose a constant c so that if the adversary blocks any set of $(1/2 - \varepsilon)n$ nodes we have that for every supernode x the adversary blocks strictly less than $|R(x)|/2$ nodes in $R(x)$, w.h.p.*

With the above lemmas we have everything in place to show the following theorem.

THEOREM 6. *The given procedure maintains the connectivity of the network under DoS-attacks by a $(1/2-\varepsilon)$-bounded $\Omega(\log \log n)$-late adversary for any $0 < \varepsilon \leq 1/2$, w.h.p. The communication work per node in each round is polylogarithmic.*

PROOF. The following statements hold by induction over the executions of the reconfiguration procedure: Since the reconfiguration procedure requires $t = \Theta(\log \log n)$ rounds, a $2t$-late adversary has no knowledge about the current composition of the groups. Therefore, by Lemma 17 at least one node is available in each group in every round. By Lemma 16 it is sufficient to choose $\beta = 2(1+\delta)c$ to satisfy the requirements of Lemma 15. Hence, Lemmas 14 and 15 imply that the reconfiguration succeeds. The bound on the communication work follows immediately from Lemma 16 and Theorem 3 as stated above. □

6. MAINTAINING CONNECTIVITY UNDER DOS-ATTACKS AND CHURN

In this section, we extend the network presented in the previous section to get a network that is resistant to both DoS-attacks and churn. We consider a $(1/2 - \varepsilon)$-bounded $\Omega(\log \log n)$-late adversary with a churn rate of $\gamma^{1/\Theta(\log \log n)}$ for any constant γ. A node in the network that receives a reference to a new node u adds u to its group by broadcasting its id within the group. A new node will always be included in the newly built groups. A leaving node informs its group so that it will not be included in the formation of the new groups. In this way, nodes enter or leave the network once a reconfiguration is completed, so both the join- and leave-operation require $O(\log \log n)$ rounds.

To successfully extend the approach to handle churn, we have to keep the group sizes within a suitable range: If the groups become too small, the adversary has a high chance to block an entire group. If the groups become large, the communication work increases. The main idea of our extension is to split and merge supernodes and their groups in order to adapt the group sizes. A supernode $x = (b_1, b_2, \dots b_\ell)$ splits by changing its label to $x = (b_1, b_2, \dots b_\ell, 0)$ and creating a new supernode $x' = (b_1, b_2, \dots b_\ell, 1)$. The nodes in $R(x)$ are divided between the supernodes x and x' uniformly at random. A supernode $x = (b_1, b_2, \dots b_\ell)$ that decides to merge informs its sibling $x' = (b_1, b_2, \dots, 1 - b_\ell)$ about its decision. It then changes its label to $x = (b_1, b_2, \dots b_{\ell-1})$ and takes over the group of its sibling so that $R(x) \leftarrow R(x) \cup R(x')$. If a sibling y of a supernode x does not exist, because y was at some point split, x forces the subtree below y (or more specifically the leaves of that subtree) to merge until y is created; x then merges with y. We refer to the length of the label of a supernode x as the *dimension* $d(x)$ of the supernode. We enforce for every supernode x that

$$c \cdot d(x) - c \leq |R(x)| \leq 2c \cdot d(x) \tag{1}$$

for some positive constant c. Hence, x splits into two nodes if $|R(x)| > 2c \cdot d(x)$, and x merges with its sibling if $|R(x)| < c \cdot d(x) - c$. We assume that initially all supernodes satisfy this restriction and that for all supernodes x, y it holds $|d(x) - d(y)| \leq 2$. We define two supernodes x and y with $d(x) \leq d(y)$ to be connected if the first $d(x)$ bits of their labels differ in exactly one coordinate, and we modify the rapid node sampling primitive for hypercubes such that each supernode x is chosen with probability $2^{-d(x)}$.

The following lemma establishes the main properties of our approach.

LEMMA 18. *For all supernodes x, y it holds $|d(x) - d(y)| \leq 2$, and for each supernode x it holds $0.5 \log n < d(x) < \log n + 2$, w.h.p.*

PROOF. We show this by induction over the reconfigurations. Initially, the lemma holds by our assumptions. For the inductive step, consider a network that has to reconfigure from n nodes to n' nodes. Let d be the unique integer such that $2^d \cdot 2cd < n' \leq 2^{d+1} \cdot 2c(d+1)$. Note that the first inequality gives us $d < d+1 < \log n'$ for $d > 0$. Combining the respective second inequality in the two equations above gives us $d > 0.5 \log n'$ for n' sufficiently large. Hence, we have $0.5 \log n' < d < \log n'$. Consider a supernode x for which $d(x) < d$ before the reconfiguration. Let X be the number of nodes that are assigned to x in the random assignment of nodes to supernodes. It holds $\mathrm{E}[X] = n' \cdot 2^{-d(x)}$ and therefore by our choice of d it holds $\mathrm{E}[X] > 4c \cdot d(x)$. We have

$$\begin{aligned}\Pr[X \text{ does not split}] &= \Pr[X \leq 2c \cdot d(x)] \\ &= \Pr[X \leq (1-\delta)\mathrm{E}[X]]\end{aligned}$$

for some $0.5 < \delta < 1$. Hence, we can apply Chernoff bounds to get

$$\Pr[X \leq (1-\delta)\mathrm{E}[X]] \leq e^{-\delta \mathrm{E}[X]/2} \leq e^{-c \cdot d(x)}.$$

Since $d(x) > 0.5 \log n$ by our induction hypothesis, x splits w.h.p. Upon splitting, x distributes the nodes in $R(x)$ uniformly at random between the two new supernodes. As a consequence, one can show using Chernoff bounds and the union bound that once all supernodes are done splitting, it holds $d(x) \geq d$ for all x. By analogous arguments, one can show that once all supernodes are done merging, it holds $d(x) \leq d+2$ for all x. Hence, if the reconfiguration terminates, we have for all x that $d \leq d(x) \leq d+2$. So to prove the first part of the lemma, it remains to show termination. Note that by the arguments above, any supernode x merges or splits a finite number of times until $d \leq d(x) \leq d+2$. It is easy to show that a supernode x with $d(x) = d$ never merges and a supernode with $d(x) = d+2$ never splits. Therefore, a supernode never leaves the specified interval of values for $d(x)$. Furthermore, note that by the restriction on $|R(x)|$ given in Equation 1 there is no group size that is at the same time too large to be accommodated by a single supernode x and too small to be accommodated by the two supernodes resulting from splitting x. Therefore, within the specified interval in the worst case the supernodes merge to build supernodes of dimension d and then split to form supernodes of dimension $d+2$ to evenly distribute the nodes in the respective groups. Hence, once the specified interval is reached, only a constant number of rounds is needed to execute the merges and splits in an organized fashion. The second part of the lemma follows by combining the bounds on the dimension of the supernodes with the bound on d. □

According to Lemma 18, as long as n stays unchanged (i.e., the number of leaving nodes is equal to the number of joining nodes), also the interval of possible values of $d(x)$ remains unchanged, and therefore the merging and splitting of groups can be realized in an organized fashion using only a constant number of rounds. The specified churn rate of $\gamma^{1/\Theta(\log\log n)}$ means that if the network has n nodes at the beginning of a reconfiguration and n' nodes at the end of that reconfiguration we have $n/\gamma \leq n' \leq \gamma n$. By the arguments given in the proof of Lemma 18, the interval for the values of $d(x)$ can shift up or down by at most $\log \gamma$ during one reconfiguration. Hence, even under churn a constant number of rounds is sufficient to execute all merges and splits.

The lower and upper bounds on $d(x)$ given in Lemma 18 provide an estimate on $\log n$ which is required by the rapid node sampling primitive. Since $|d(x) - d(y)| \leq 2$ for all supernodes x, y, a supernode is connected to at most constantly many other supernodes. Applying the bounds on $d(x)$ to Equation 1 shows that the group sizes are in $O(\log n)$ so the communication work stays polylogarithmic. At the same time, by choosing an appropriate c the group sizes can be made large enough so that, according to Lemma 17 from the previous section, each group contains at least one available node at all times. Therefore, we have the following theorem.

THEOREM 7. *The given procedure maintains the connectivity of the network under DoS-attacks and simultaneous adversarial churn by a $(1/2-\varepsilon)$-bounded $\Omega(\log\log n)$-late adversary with churn rate $\gamma^{1/\Theta(\log\log n)}$ for any $0 < \varepsilon \leq 1/2$ and any constant γ, w.h.p. The communication work per node in each round is polylogarithmic.*

One may wonder whether the churn rate in the theorem can be extended to include crash failures. However, for that it must be possible to distinguish a crash failure from a node that is under DoS-attack. If this is possible, then the departure of a crashed node can simply be emulated by the other nodes in its groups. Otherwise, the nodes do not know how long they have to keep up emulating a non-responding node v in the system. If they do not emulate it long enough, v will be excluded, which will make it hard for it to be reintegrated since after $O(\log\log n)$ steps the adversary will know about all nodes that know v and that are known to v, so a dedicated DoS-attack can easily isolate v.

7. APPLICATIONS

Finally, we demonstrate that our results have many applications including scalable anonymizers, DHTs and publish-subscribe systems that are robust against massive DoS-attacks by $\Omega(\log\log n)$-late adversaries.

7.1 Robust Anonymous Routing

A standard approach to preserve anonymity in a distributed system is to exchange information through relays. A prominent example is the Tor network [8], which consists of several thousand servers spread all over the world so that there is no single authority having access to a significant fraction of its servers. In simple terms, we have a (possibly variable) set U of users and a fixed set V of servers that are used as relays to anonymize the source of a request. The goal of the servers is to ensure that whenever a user v wants to send a request to user w, w can receive the request and send a reply back to v via the servers without knowing v. More precisely, we would like to have a server system satisfying the following minimal set of requirements:

1. For every request, the request and its reply can be delivered reliably under the given attacker (robustness),
2. every message will exit the server system at a server chosen uniformly at random with respect to the current knowledge of the attacker (anonymity), and
3. at any time the servers form a network of polylogarithmic degree and the communication work per server

is polylogarithmic in each round given that in each round every server only receives a constant number of requests (scalability).

We added "with respect to the current knowledge of the attacker" in item 2 because all that matters is that based on its current knowledge, the attacker cannot make a better guess on which servers to monitor than a uniform distribution. If it learns about the final server in hindsight, this is uncritical as long as it does not have any monitoring data for that server since in that case it does not learn about the destination w. Using our reconfiguration techniques, we can set up a server system satisfying all of these requirements w.h.p. against a $\Omega(\log\log n)$-late adversary. It works as follows:

We organize the servers in a hypercube as described in Section 5. Recall that in each reconfiguration round, $\Theta(\log n)$ random supernodes are selected for each server so that after the reconfiguration, $|R(x)| = \Theta(\log n)$ for every supernode x, w.h.p. In addition to that, for each server v, a specific supernode x that v belongs to is picked, and $D(v) = R(x)\setminus\{v\}$ is defined as its *destination group*. Note that since every server in $D(v)$ has selected supernode x uniformly at random, it holds that if v picks a server out of $D(v)$ (using a rule that does not take any server properties into account), then this corresponds to a server chosen uniformly at random from V.

Suppose now that a user v wants to send a message to user w. We assume that v can contact some currently non-blocked server $s(v)$. $s(v)$ will then forward the message to all servers in $D(v)$, which will forward it to w (if they are non-blocked) and remember v. Once w has sent its reply back to (all non-blocked servers it received the message from in) $D(v)$, $D(v)$ will send the reply back to v.

Using our insights from Section 5, we immediately get:

COROLLARY 2. *Our server system exchanges messages between users via the servers in a reliable, anonymous, and robust fashion using only $O(1)$ communication rounds for each request and just polylogarithmic communication work even if the servers are under DoS-attack by a $(1/2-\epsilon)$-bounded $\Omega(\log\log n)$-late adversary for any constant $0 < \epsilon \leq 1/2$.*

Of course, there are still various ways of attacking our system, and various more advanced defenses have already been proposed (e.g., [13]), but here we just wanted to give a basic idea of how our techniques can be used to provide a robust anonymizing system.

7.2 Robust DHTs

Another important application area of our techniques are distributed hash tables. At first glance, one may think that a continuous reconfiguration of the network also requires the data in a DHT to be continuously moved. But this is not needed when using the DHT named RoBuSt proposed in [11]. In this paper, a DHT on top of a fixed set of n reliable servers is presented that can correctly serve any set of read and write requests with at most one request per non-blocked server with just polylogarithmic work per server even if it is attacked by a 0-late adversary that can block up to $\gamma n^{1/\log\log n}$ many servers for a sufficiently small constant $\gamma > 0$. The work bound includes any movements of data needed to keep the system in a state that protects itself against the ongoing DoS-attacks. However, for RoBuSt to work, the servers have to be completely interconnected, i.e., every server knows all other servers in the system. Also, the adversary can only make a blocking decision every $O(\log^3 n)$ rounds (but whenever it can make a blocking decision, it can take the entire current state of the system into account).

First, we describe how to get rid of the completely interconnected network. It turns out that this is needed so that no matter which servers are blocked by the adversary, the remaining servers still form a connected network. In fact, they are still completely interconnected, which allows them to quickly organize themselves so that they can emulate a d-dimensional k-ary butterfly, which is needed for the routing of messages. In order to emulate a d-dimensional k-ary butterfly, the servers will establish connections forming a k-ary hypercube, which is defined as follows.

DEFINITION 1. *A d-dimensional k-ary hypercube is a graph $G = (V, E)$ with $V = \{0, \ldots, k-1\}^d$, in which two nodes $(v_1, \ldots, v_d)$, $(w_1, \ldots, w_d) \in V$ are connected by an edge if and only if they differ in only one coordinate.*

It is easy to verify that a d-dimensional k-ary hypercube consists of k^d nodes and has a degree of $(k-1)\cdot d$ and a diameter of d. For the particular case that $d = k/\log k$ (which is the case in [11]), this results in a degree of $O(\log^2 n/\log\log n)$ and a diameter of $\log n/\log\log n$, where $n = 2^k$ is the number of nodes in the hypercube.

It is straightforward to extend the reconfiguration procedure for the binary hypercube in Section 5 to the k-ary hypercube. Then we have a network that is robust against a $(1/2-\epsilon)$-bounded $\Omega(\log\log n)$-late adversary. Whenever a request in the original RoBuSt system needs to fetch some data from a node v, we simply exchange messages between the group $R(v)$ representing v in the k-ary hypercube and v. In that way, all results in [11] carry over so that we get the following result.

THEOREM 8. *With only logarithmic redundancy, the extended RoBuSt system correctly serves any set of read and write requests (at most $O(1)$ per non-blocked server) in at most $O(\log^3 n)$ communication rounds with a congestion of at most $O(\log^3 n)$ at every server for any $\Omega(\log\log n)$-late adversary that can block at most $\gamma n^{1/\log\log n}$ servers at a time.*

7.3 Robust Publish-Subscribe Systems

Finally, we also note how to construct a robust publish-subscribe system with our reconfiguration procedures. A publish-subscribe system can easily be emulated by a server-based DHT as described above: each subscriber group is identified some unique key k, and we store k together with the number of publications $m(k)$ already done for k together with k in a DHT. For any set of publications with at most one per server, first the keys of the publications are aggregated to determine the number of publications for each used key. This can be done in time $O(\log n/\log\log n)$ in the k-ary hypercube that is reconfigured as described above using, for example, Ranade's routing scheme [28]. Once this number, $m'(k)$, is known for some k, $m(k)$ can be updated to $m(k) + m'(k)$ and unique numbers from $m(k)+1$ to $m(k)+m'(k)$ can be given to the publications so that publication i for key k can be stored under the key $(k, m(k)+i)$ in the DHT. Whenever a subscriber for group k wants to access the latest publications, it simply needs to access $m(k)$ and can then request all publications with keys till $(k, m(k))$.

Overall, we can then obtain the same result for publish-subscribe systems as for DHTs above if at most $O(1)$ publish or retrieve requests are issued per non-blocked server.

8. CONCLUSION AND FUTURE WORK

We demonstrated how efficient network reconfiguration can be used to construct overlay networks that maintain connectivity under large-scale adversarial churn and adversarial DoS-attacks. Possible future work includes the extension of the churn- and DoS-resistant network presented in Section 6 to handle constant churn rates. In less immediate future work, it would be interesting to investigate the efficient reconfiguration of other topologies and further applications of our rapid node sampling primitives.

9. ACKNOWLEDGMENTS

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901) and by the EU within FET project MULTIPLEX under contract no. 317532.

10. REFERENCES

[1] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *Proc. of IPDPS*, page 40, 2003.

[2] J. Aspnes and U. Wieder. The expansion and mixing time of skip graphs with applications. *Distributed Computing*, 21(6):385–393, 2009.

[3] J. Augustine, G. Pandurangan, P. Robinson, S. Roche, and E. Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In *Proc. of FOCS*, pages 350–369, 2015.

[4] B. Awerbuch and C. Scheideler. A denial-of-service resistant DHT. In *Proc. of DISC*, pages 33–47, 2007.

[5] B. Awerbuch and C. Scheideler. Towards scalable and robust overlay networks. In *Proc. of IPTPS*, 2007.

[6] A. Berns, S. Ghosh, and S. V. Pemmaraju. Building self-stabilizing overlay networks with the transitive closure framework. *TCS*, 512, 2013.

[7] C. Cooper, M. Dyer, and A. J. Handley. The flip markov chain and a randomising p2p protocol. In *Proc. of PODC*, pages 141–150, 2009.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proc. SSYM*, 2004.

[9] J. R. Douceur. The sybil attack. In *Proc. of IPTPS*, pages 251–260, 2002.

[10] M. Eikel and C. Scheideler. IRIS: a robust information system against insider dos-attacks. In *Proc. of SPAA*, pages 119–129, 2013.

[11] M. Eikel, C. Scheideler, and A. Setzer. RoBuSt: A crash-failure-resistant distributed storage system. In *Proc. of OPODIS*, pages 107–122, 2014.

[12] T. Feder, A. Guetz, M. Mihail, and A. Saberi. A local switch markov chain on given degree graphs with application in connectivity of peer-to-peer networks. In *Proc. of FOCS*, pages 69–76, 2006.

[13] J. Feigenbaum, A. Johnson, and P. Syverson. Preventing active timing attacks in low-latency anonymous communication. In *Prof. of PETS*, pages 166–183, 2010.

[14] D. Foreback, A. Koutsopoulos, M. Nesterenko, C. Scheideler, and T. Strothmann. On stabilizing departures in overlay networks. In *Proc. of SSS*, 2014.

[15] J. Friedman. *A Proof of Alon's Second Eigenvalue Conjecture and Related Problems*, volume 195 of *Memoirs of the AMS*. 2008.

[16] T. P. Hayes, J. Saia, and A. Trehan. The forgiving graph: a distributed data structure for low stretch under adversarial attack. In *Proc. of PODC*, 2009.

[17] R. Jacob, A. W. Richa, C. Scheideler, S. Schmid, and H. Täubig. A distributed polylogarithmic time algorithm for self-stabilizing skip graphs. In *Proc. of PODC*, pages 131–140, 2009.

[18] T. Jacobs and G. Pandurangan. Stochastic analysis of a churn-tolerant structured peer-to-peer scheme. *Peer-to-Peer Networking and Applications*, 6(1), 2013.

[19] J. JaJa. *An Introduction to Parallel Algorithms*. Addison Wesley, 1992.

[20] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proc. of the ACM SIGCOMM*, pages 61–72, 2002.

[21] F. Kuhn, S. Schmid, and R. Wattenhofer. A self-repairing peer-to-peer system resilient to dynamic adversarial churn. In *Proc. of IPTPS*, 2005.

[22] C. Law and K.-Y. Siu. Distributed construction of random expander networks. In *Proc. of INFOCOM*, volume 3, pages 2133–2143 vol.3, March 2003.

[23] L. Lovasz. Random walks on graphs: A survey, 1993.

[24] P. Mahlmann and C. Schindelhauer. Distributed random digraph transformations for peer-to-peer networks. In *Proc. of SPAA*, pages 308–317, 2006.

[25] D. Nanongkai, A. D. Sarma, and G. Pandurangan. A tight unconditional lower bound on distributed randomwalk computation. In *Proc. of PODC*, 2011.

[26] M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *Proc. of SPAA*, pages 50–59, 2003.

[27] G. Pandurangan, P. Robinson, and A. Trehan. Dex: Self-healing expanders. In *Proc. of IPDPS*, 2014.

[28] A. Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42(3):307–326, 1991.

[29] J. Saia and A. Trehan. Picking up the pieces: Self-healing in reconfigurable networks. In *Proc. of IPDPS*, pages 1–12, 2008.

[30] A. D. Sarma, D. Nanongkai, and G. Pandurangan. Fast distributed random walks. In *Proc. of PODC*, 2009.

[31] A. D. Sarma, D. Nanongkai, G. Pandurangan, and P. Tetali. Distributed random walks. *Journal of the ACM*, 60:2:1–2:31, 2013.

[32] A. Singh, T. Ngan, P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *Proc. of the INFOCOM*, 2006.

[33] K. Suto, H. Nishiyama, N. Kato, T. Nakachi, T. Fujii, and A. Takahara. THUP: A P2P network robust to churn and DoS attack based on bimodal degree distribution. *IEEE Journal on Selected Areas of Communication*, 31(9):247–256, 2013.