# IRIS: A Robust Information System Against Insider DoS Attacks

MARTINA EIKEL and CHRISTIAN SCHEIDELER, University of Paderborn

In this work, we present the first scalable distributed information system, that is, a system with low storage overhead, that is provably robust against denial-of-service (DoS) attacks by a current insider. We allow a current insider to have complete knowledge about the information system and to have the power to block any $\varepsilon$-fraction of its servers by a DoS attack, where $\varepsilon$ can be chosen up to a constant. The task of the system is to serve any collection of lookup requests with at most one per nonblocked server in an efficient way despite this attack. Previously, scalable solutions were only known for DoS attacks of past insiders, where a past insider only has complete knowledge about some past time point $t_0$ of the information system. Scheideler et al. [Awerbuch and Scheideler 2007; Baumgart et al. 2009] showed that in this case, it is possible to design an information system so that any information that was inserted or last updated after $t_0$ is safe against a DoS attack. But their constructions would not work at all for a current insider. The key idea behind our IRIS system is to make extensive use of coding. More precisely, we present two alternative distributed coding strategies with an at most logarithmic storage overhead that can handle up to a constant fraction of blocked servers.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

General Terms: Theory

Additional Key Words and Phrases: Distributed systems, denial-of-service attacks, DHT

**ACM Reference Format:**
Martina Eikel and Christian Scheideler. 2015. IRIS: A robust information system against insider DoS attacks. ACM Trans. Parallel Comput. 2, 3, Article 18 (October 2015), 33 pages.
DOI: http://dx.doi.org/10.1145/2809806



## 1. INTRODUCTION

Distributed denial-of-service (DoS) attacks are one of the biggest threats in the Internet. The basic idea behind a DoS attack is to make a service unavailable to its intended users. There are various ways of achieving that, like causing computationally expensive operations [Kandula et al. 2005], downloading large files [Ratliff 2005], exploiting protocol bugs, or just overloading servers with junk. Information services like Google and Akamai are frequently under attack, and the Domain Name System also has been involved in many attacks, either as a victim itself or as a means to raise reflected or DNS amplification attacks [Wikipedia 2013].

The predominant approaches in information systems to deal with the threat of DoS attacks are to use *redundancy* and *information hiding*: information that is replicated on multiple servers is more likely to remain accessible during a DoS attack, in particular,

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901).
Authors' addresses: M. Eikel and C. Scheideler, University of Paderborn, Department of Computer Science, Fürstenallee 11, 33102 Paderborn, Germany; emails: {martinah, scheideler}@upb.de.

DOI: http://dx.doi.org/10.1145/2809806

if the attacker does not know the servers or how the data items are distributed among the servers. For example, if $\Theta(\log n)$ copies of a data item are placed randomly among $n$ servers and these random positions are not known to the attacker, then it is easy to show that any strategy of the attacker to block half of the servers will not block all of the copies with high probability.[1] The situation is completely different, however, when considering an insider, that is, someone who has complete knowledge about the system. Since information cannot be hidden any more in this case, it seems unavoidable to replicate a data item across more than $t$ servers to remain accessible under a DoS attack that can block up to $t$ servers, which creates a huge storage overhead. This is why in our previous work we just focused on past insiders [Awerbuch and Scheideler 2007; Baumgart et al. 2009]. However, it turns out that this dilemma can be circumvented when using coding, which is the key idea of the IRIS system presented in this article ("IRIS" is a short form of "Insider-Resistant Information System").

### 1.1. Model

To keep the presentation of our ideas clean and simple, we will use a simple model. We assume that the information system consists of a static set $V$ of $n$ reliable servers of identical type. (We deliberately use the word "server" here since assuming a static set of nodes is unrealistic for peer-to-peer solutions; we focus instead on solutions based on dedicated equipment provided by one or more institutions or companies.) The servers are responsible for storing the data as well as handling the user requests. We assume that all data items are of the same size, and a data item $x$ is uniquely identified by a key $key(x)$. The universe of all possible keys is denoted by $U$, and we set $m = |U|$. The only type of user requests that we consider are lookup($k$) requests, where $k$ is any key in $U$ (i.e., the purpose of the system is just to deliver information, not to let users update it). Given a lookup($k$) request, the system is supposed to either return the data item $x$ with $key(x) = k$ or return $NULL$ if no such data item exists.

Every server knows about all other servers and can therefore directly communicate with any one of them. This does not endanger scalability since we assume the set of servers to be static and we do not expect the servers to maintain an open connection to each other server. Instead, we only expect the servers to hold the IP addresses of all other servers. This does not cause a problem either, since millions of IP addresses can easily be stored in main memory in any reasonable computer today. We will use the standard synchronous message passing model for the communication between the servers. That is, time proceeds in synchronized *communication rounds*, or simply *rounds*, and in each round each server first receives all messages sent to it in the previous round, processes all of them, and then sends out all messages that it wants to send out in this round. A message will never get lost unless it is sent to a blocked server. We assume that the time needed for internal computations is negligible (the IRIS protocols are simple enough to satisfy this property). Note, however, that using local synchronizers [Peleg 2000], our algorithms also work in asynchronous settings. All we need is a bounded transmission time between two nonattacked servers.

The competition between the information system and the attacker works as follows. Initially, the attacker can inspect the entire system and selects, based on that, an arbitrary $\varepsilon$-fraction of the servers to be blocked (where $\varepsilon$ depends on the limitations of the given system). A server that is blocked will not react to messages from the other servers. We assume that the servers have a failure detector that allows them to determine whether a server is blocked so that statements like "if server $i$ is blocked then ..." are allowed in the protocol. In this work, we assume the adversary to block

[1] "With high probability," or for short, "w.h.p.," means a probability of at least $1 - 1/n^c$, where the constant $c$ can be made arbitrarily large.

a fraction of the servers by DoS attacks. Instead, we could also allow the adversary to simply crash a fraction of the servers. The latter attacks are called crash failures, where DoS attacks represent one reason for crash failures.

The attacker may then select an arbitrary collection of lookup requests, one per nonblocked server (the most simple case of an even request distribution among servers). That is, the keys selected by the attacker may or may not be associated with data items stored in the system, and the attacker is also allowed to issue multiple lookup requests for the same key. The task of the system is to correctly serve *all* of these requests.

In order to measure the quality of the information system, we introduce the following notation. A storage strategy is said to have a *redundancy* of $r$ if $r$ times more storage (including any control storage) is used for the data than storing the plain data. We call an information system

—*scalable* if its redundancy is at most polylog($n$),
—*efficient* if any collection of lookup requests specified by the attacker can be processed correctly in at most polylog($n$) many communication rounds in which every server sends and receives at most polylog($n$) many messages of at most polylog($n$) size, and
—*robust* if any collection of lookup requests specified by the attacker can be processed correctly even if up to an $\varepsilon$-fraction of the servers is blocked by an insider.

Our goal is to design an information system that is scalable, efficient, and robust for an $\varepsilon$ that is as large as possible. As we will see, the IRIS system satisfies all of these properties.

### 1.2. Related work

Due to their importance, DoS attacks are a well-studied problem (e.g., see Dittrich et al. [2005] and Mirkovic and Reiher [2004] for an overview). Unfortunately, it is often difficult to distinguish DoS traffic from legitimate traffic, which limits the effectiveness of network-layer and transport-layer DoS prevention tools [Walfish et al. 2005], such as filtering out anomalies [Mazu Networks Inc. 2008], blacklisting particular IP addresses, using TCP SYN cookies [Bernstein 2008], and pushback [Ioannidis and Bellovin 2002]. This has led some researchers to follow alternative means such as letting legitimate clients "speak up" [Walfish et al. 2005, 2006].

In this article, we do not seek to prevent DoS attacks but rather focus on how to maintain good availability and performance during the attack. Our system is based on the distributed hash table (DHT) paradigm (e.g., Bhargava et al. [2004], Druschel and Rowstron [2001], Harvey et al. [2003], Ratnasamy et al. [2001], and Stoica et al. [2002b]), with the new twist of using coding. Various DoS-resistant systems based on DHTs have already been proposed [Kargl et al. 2001; Keromytis et al. 2002; Morein et al. 2003]. For instance, the Secure Overlay Services approach [Keromytis et al. 2002] uses proxies on Chord to defend against DoS attacks. A Chord overlay is also used by the Internet Indirection Infrastructure *i3* [Stoica et al. 2002a] to achieve resilience to DoS attacks. Other DoS-limiting architectures have been proposed in Oikonomou et al. [2006] and Yang et al. [2005]. Many of these systems are based on traffic analysis or some indirection approach.

Nonmalicious DoS attacks like flash crowds have also been studied in the context of DHTs. Examples in the systems community include CoopNet [Padmanabhan and Sripanidkulchai 2002], Backslash [Stading et al. 2002], and PROOFS [Stavrou et al. 2002], and there is also theoretical work [Naor and Wieder 2003]. However, these works only consider scenarios where many requests are targeted to the same data item, but there are harder instances like many requests to different items *at the same location* (which can be set up by an attacker when the hash functions are known). These instances can still be handled in DHTs using techniques originally proposed for

CRCW PRAMs [Awerbuch and Scheideler 2006], but these techniques cannot protect the system against DoS attacks that can block specific servers.

The first DHTs that are robust against past-insider DoS attacks were proposed in Awerbuch and Scheideler [2007] and Baumgart et al. [2009]. A past insider only has complete knowledge of the information system up to some *past* time point $t_0$. For this kind of insider, it is possible to design an information system so that any information that was inserted or last updated *after* $t_0$ is safe against a DoS attack [Awerbuch and Scheideler 2007; Baumgart et al. 2009]. But the constructions proposed in these papers would not work at all for a current insider because they are heavily based on randomization to ensure unpredictability.

For a system to be able to tolerate a current insider, as considered in this work, the key idea is to make massive use of distributed coding. In particular, we make use of erasure codes. An important class of these codes is $(n, k)$ maximum distance separable (MDS) codes. An $(n, k)$ MDS code stores data in $n$ storage nodes such that any failure of $(n - k)$ storage nodes can be tolerated. Hence, $(n, k)$ MDS codes achieve the optimal lower bound on the storage overhead, which is $n/(n - k)$. Examples of MDS array codes (where the input is organized into columns and rows) are: Reed-Solomon codes [Reed and Solomon 1960], EVENODD [Blaum et al. 1994], RDP [Corbett et al. 2004], B-code [Xu et al. 2006], X-code [Xu and Bruck 1999], and STAR-code [Huang and Xu 2008].

While MDS codes are optimal in terms of storage overhead, they may have a significant overhead in the *repair bandwidth*. The repair bandwidth was initially introduced by Dimakis et al. [2010] and denotes the amount of information to be communicated during the repair of node failures. Specifically, for the case of repairing only a single node failure, the overhead on the repair bandwidth may become too large. This problem motivated the development of so-called *regenerating codes,* which were first introduced by Dimakis et al. [2010]. The codes presented in this work match the lower bound on the storage cost as well as MDS codes, while additionally significantly reducing the repair bandwidth. Dimakis et al. [2010] additionally showed that in case of a single failure, the repair bandwidth is lower bounded by $l(n - 1)/(n - k)$, where $l$ denotes the capacity of the nodes. Besides the codes in Dimakis et al. [2010], many further regenerating codes have been proposed that achieve this lower bound (e.g., Tamo et al. [2013], Papailiopoulos et al. [2011], Cadambe et al. [2011], and Suh and Ramchandran [2010, 2011]).

Distributed coding has proved to be useful not only in distributed storage systems but also in the field of information dissemination and gossiping. For the multicasting problem, Ahlswede et al. [2000] showed that it is not sufficient to regard data to transfer just as an unsplittable entity but that the optimal throughput from the source to targets in a network can be achieved if and only if the intermediate nodes code messages. In fact, it suffices for an intermediate node to compute linear combinations of the received data for coding it [Li et al. 2003]. In order to create those linear combinations, Ho et al. [2006] showed that randomly chosen coefficients work for any network, w.h.p. Haeupler [2011] introduced a new technique (denoted as projection analysis) for analyzing the runtime of gossip algorithms that are based on random linear network coding.

In previous coding schemes, data was encoded separately from each other; that is, outputs of the encoding of tuples of data items are not encoded with each other once more. This yields a storage overhead that is linear in the maximum number of node failures allowed. Since the information system presented in this work is supposed to have at most a logarithmic redundancy, we needed to come up with a new distributed coding scheme that decreases the redundancy by hierarchically interlacing encoded data items with each other.

### 1.3. Our Contribution

Consider a distributed information system that holds that for each data item $d$, it is supposed to store $c$ copies at $c$ servers. If the adversary blocks these $c$ servers, then the system would not be able to correctly serve a lookup request for $d$. That is, if we allow the adversary to block $c$ servers, then the distributed information system needs to store $c + 1$ copies of each data item in order to be able to correctly answer each lookup request. Hence, when considering a current insider who knows *everything* about the information system, standard replication and information hiding techniques are useless to efficiently protect the system against DoS attacks. In order to circumvent this problem, we present a distributed information system that makes massive use of a (very simple) distributed coding strategy. For this purpose, we partition each data item into $c$ pieces using a conventional coding strategy (e.g., Reed-Solomon codes) such that $c/4$ of these pieces suffice to recover the complete data item. On top of this, we use a very simple distributed coding strategy that only consists of some parity computations and guarantees the recovery of the data on a blocked server in case of a single failure. In order to guarantee the recovery of all data items if there is a failure of many servers, tuples of data blocks are encoded with each other in a blockwise fashion. By intelligently reapplying and interlacing this block-based coding strategy, our system will allow a huge fraction of the servers to be blocked. This distributed coding strategy will ensure that data is not only replicated onto a few servers but also that each server holds some encoding information for every data item in the system.

The development of a lookup protocol, which enables us to efficiently serve any set of lookup requests despite a massive DoS attack by an insider, implied several challenges we had to deal with, for instance, the problem of answering the requests such that no additional congestion at the servers is caused that would block the servers. All in all, we developed a distributed information system IRIS that serves any set of lookup requests (with at most a constant number of requests at each nonblocked server) in polylogarithmic time, with at most polylogarithmic congestion at each server in each round and an at most logarithmic redundancy.

More precisely, we present two variants of IRIS, Basic IRIS and Enhanced IRIS, with the following properties given that $m$ is at most polynomial in $n$.

THEOREM 1.1. *Basic IRIS and Enhanced IRIS are both scalable and efficient. Whereas the Basic IRIS just needs a constant redundancy to protect itself against insider DoS attacks blocking up to* $\gamma \cdot n^{1/\log\log n}$ *servers for a constant* $0 < \gamma < 1/24$*, Enhanced IRIS needs* $O(\log n)$ *redundancy but can protect itself against insider DoS attacks blocking up to a constant fraction of the servers.*

## 2. BASIC IRIS

The key idea of Basic IRIS is to make massive use of distributed coding. Hence, we start by describing the coding and storage strategy of Basic IRIS in Section 2.1. Afterward, in Section 2.2, we present the lookup protocol of our system. The presentation of Basic IRIS finishes in Section 2.3 with a correctness proof of the lookup protocol.

### 2.1. Storage and Coding Strategy

In the following, we first introduce the coding strategy Basic IRIS uses (Section 2.1.1) followed by the presentation of the complete storage strategy of our system (Section 2.1.2).

*2.1.1. Coding Strategy.* This distributed coding strategy is a very simple error-correcting code that is able to recover one out of $k \in \mathbb{N}$ symbols. In other words, assume $k$ servers holding one data item each while exactly one of these servers is blocked. Then,

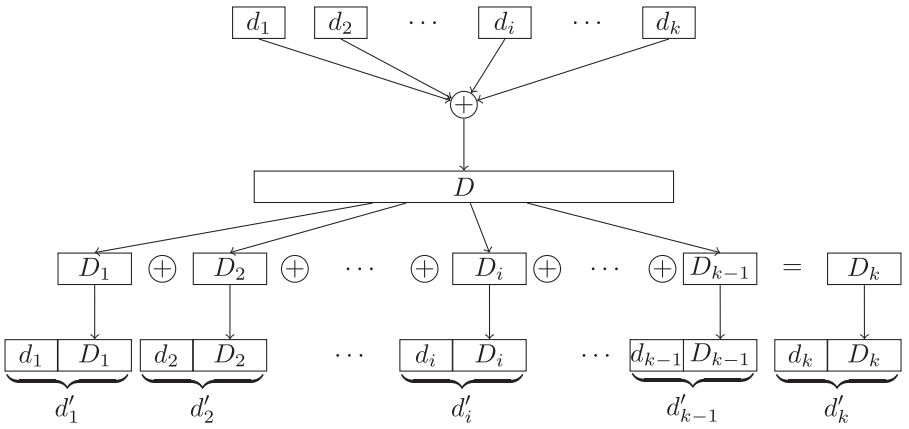


Fig. 1. Visualization of the encoding of $k$ data items $d_1, \ldots, d_k$.

our coding strategy guarantees that the data item held by the blocked server can be recovered using only the (parity) information the remaining $k-1$ servers hold. To be more precise, the encoding of $k$ data items $d_1, \ldots, d_k$ works as follows: We first compute $D = d_1 \oplus d_2 \oplus \ldots \oplus d_k$, where $\oplus$ is the bitwise parity operation. Then we cut $D$ into $k-1$ pieces $D_1, \ldots, D_{k-1}$ of equal size (up to an additive 1) and set $d'_i = d_i \circ D_i$, $i \in \{0, \ldots, k\}$, where $\circ$ is the concatenation operator. Finally, we compute $D_k = D_1 \oplus D_2 \oplus \cdots \oplus D_{k-1}$ and set $d'_k = d_k \circ D_k$. See Figure 1 for a visualization of the encoding of $k$ data items.

Our coding strategy satisfies the following lemma.

LEMMA 2.1. *Let the data items $d_1, \ldots, d_k$ be encoded with each other using the previously described coding strategy resulting in $d'_1, \ldots, d'_k$. Then, if one $d'_j$, $j \in \{1, \ldots, k\}$, is missing, the information in $d'_1, \ldots, d'_{j-1}, d'_{j+1}, \ldots, d'_k$ suffices to recover $d_1, \ldots, d_k$.*

PROOF. Suppose the data block $d'_j$ is missing. Since each data block $d'_i$, $i \in \{1, \ldots, k\}\backslash\{j\}$ contains $d_i$, the information in any corresponding $d_i$ can be recovered directly. Furthermore, the parity coding strategy allows us to recover $D_j = \bigoplus_{i \neq j} D_i$. This allows us to recover $D$ by computing $D = D_1 \circ \ldots \circ D_{k-1}$, which then allows us to recover $d_j$ by computing $d_j = D \oplus \bigoplus_{i \neq j} d_i$. □

Up to this point, our coding strategy only guarantees the recovery of one out of $k$ symbols, but we want Basic IRIS to be able to correctly answer each lookup request despite the attack of an insider that blocks up to $\gamma n^{1/\log\log n}$ servers ($0 < \gamma < 1/24$). That is, using just one encoding routine for all data items at the servers is not sufficient. Instead, we hierarchically encode data items with each other. That is, we first encode different blocks of $k$ data items with each other. Next, we select new blocks each consisting of $k$ results of the first encoding and encode these blocks. This procedure is repeated until each server stores some encoding information of each data item that is stored in the system. In order to construct the blocks of $k$ data items each, we make use of a $k$-ary butterfly.

*Definition* 2.2 (*$k$-ary Butterfly*). For any $d, k \in \mathbb{N}$, the *$d$-dimensional $k$-ary butterfly* $BF(k, d)$ is a graph $G = (V_k, E)$ with node set $V_k = [d+1] \times [k]^d$ and edge set $E$ with

$$E = \{\{(i, x), (i+1, (x_1, \ldots, x_i, b, x_{i+2}, \ldots, x_d))\} \mid x = (x_1, \ldots, x_d) \in [k]^d,\ i \in [d],\ \text{and } b \in [k]\}.$$

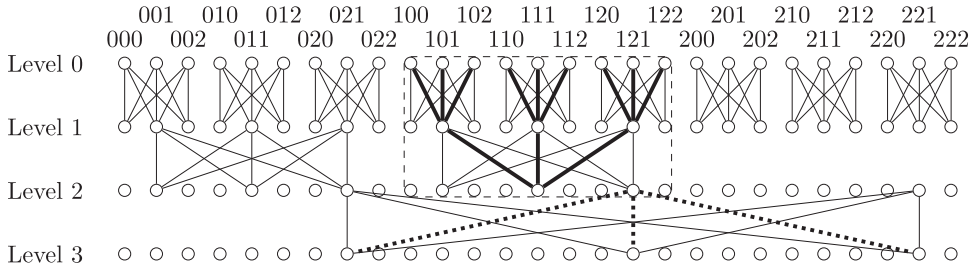


Fig. 2. Visualization of a $k$-ary butterfly $BF(k, d)$ for $k = d = 3$. For a better readability, most of the edges from level 2 and 3 are omitted. The edges shown between levels 1, 2, and 3 visualize three $k$-blocks at levels 1 and 2. The dashed box denotes the subbutterfly $BF((2, 111))$. The thick solid lines in the dashed box denote the edges of $UT((2, 111))$. The thick dotted lines denote the edges of $LT((2, 121))$.

A node $u$ of the form $(\ell, x)$ is said to *be on level* $\ell$ of $G$. For a node $u = (\ell, x)$, $LT(u)$ is the unique $k$-ary tree of nodes reached from $u$ when the butterfly is going downward (i.e., to nodes on levels $\ell' > \ell$) and $UT(u)$ is the unique $k$-ary tree of nodes reached from $u$ when the butterfly is going upward. Moreover, let $BF(u)$ be the unique $k$-ary subbutterfly of dimension $\ell$ ranging from level 0 to $\ell$ in $BF(k, d)$ that contains $u$. Finally, $B(u)$ is the unique $k$-ary subbutterfly of dimension 1 (which is just a bipartite graph of $k$ nodes on each side) ranging from level $\ell$ to $\ell + 1$ in $BF(k, d)$ that contains $u$. We also call $B(u)$ a *$k$-block at level* $\ell$.

See Figure 2 for a visualization.

We are now ready to describe how to encode any set of $n$ data items with each other using the $k$-ary butterfly as the underlying topology. Consider any $BF(k, d)$, and let $n = k^d$. The encoding of a set of data items $d_0, \ldots, d_{n-1}$ of uniform size works as follows: Initially, $d_i$ is placed in node $(0, i)$ for every $i \in \{0, \ldots, n - 1\}$. Given that in level $\ell$ we have already assigned data items $d(\ell, x)$ to the nodes $(\ell, x)$, we use the previously described coding strategy to assign data items $d(\ell + 1, x)$ to the nodes at level $\ell + 1$: that is, for each $k$-block $B$ at level $\ell$ with nodes $(\ell, x_1), \ldots, (\ell, x_k)$, we encode the data items $d_1 = d(\ell, x_1), \ldots, d_k = d(\ell, x_k)$ with each other resulting in $d'_1, \ldots, d'_k$. Finally, we set $d(\ell + 1, x_i) = d'_i$ for all $i \in \{1, \ldots, k\}$.

For a node $v$ in $BF(k, d)$, let $|d(v)|$ denote the size of the data stored in node $v$. Since to each node $(\ell + 1, x_i)$ a bit string of size $|d(\ell, x_i)|/(k - 1) \pm 1$ is appended, the following lemma holds.

LEMMA 2.3. *For any $k$-block $B$ with node sets $(\ell, x_1), \ldots, (\ell, x_k)$ and $(\ell + 1, x_1), \ldots, (\ell + 1, x_k)$, it holds that $|d(\ell + 1, x_i)| \leq (1 + 1/(k - 1))|d(\ell, x_i)|$ up to an additive 1.*

For simplicity, we will ignore in the following the additive 1 because $|d(\ell, x_i)|$ may not be perfectly divisible by $k - 1$. This will only cause a constant factor deviation from the bounds below as long as the original data items have a size of $z$ with $z \geq k$.

The decoding of the data items in $k$-ary butterfly follows a bottom-up approach. That is, beginning with the last level, level $\log_k n$, the nodes first encode all $k$-blocks from level $\log_k n - 1$. The decoding of a single $k$-block proceeds as described in Lemma 2.1. Using this information, the nodes continue to decode all $k$-blocks from level $\log_k n - 2$. The nodes proceed in this manner until all $k$-blocks from level 0 have been decoded. Note that by Lemma 2.1, a $k$-block can only be decoded if from the set of $k$ servers that hold the data encoded in the block of at most one server is blocked.

*2.1.2. Storage Strategy.* In the previous section, we showed how a set of $n$ data items is encoded with each other, but we did not describe how these $n$ data items are chosen or

at which servers the data resulting from the encoding is stored. In the following, let $n$ be the number of servers with $n = k^d$ for some $d \in \mathbb{N}$. For each server $s_i$, we introduce $d$ virtual nodes $(0, i), \ldots, (d, i)$ that server $s_i$ will *emulate*. We connect all virtual nodes with each other as defined by the $k$-ary butterfly $BF(k, d)$. Whenever a virtual butterfly node $(\ell, i)$ is supposed to perform an action (like forwarding a message or storing some data), the server $s_i$ will perform this action instead. We say a server $s$ is connected to a server $s'$ (via the $k$-ary butterfly) if $s$ emulates a butterfly node that is in $BF(k, d)$ connected to a butterfly node that is emulated by $s'$.

In order to present how a set of $m' < m \in \mathbb{N}$ data items can be stored in Basic IRIS, we distinguish between the following three cases depending on $m'$:

*Case 1*. If $m' = n$, that is, Basic IRIS has to store exactly $n$ data items $d_0, \ldots, d_{n-1}$. In this case the encoding of these data items works just as described above. According to Lemma 2.3, node $(\ell, i)$ in $BF(k, d)$ has the property that $d(\ell, i) = d(\ell - 1, i) \circ p_\ell(i)$ for some parity information $p_\ell(i)$ with $|p_\ell(i)| \leq |d(\ell - 1, i)|/(k - 1)$. Hence, it suffices for server $s_i$ to store $d_i, p_1(i), \ldots, p_d(i)$ in order to be able to recover every $d(\ell, i)$.

*Case 2*. If Basic IRIS has to store $m' < n$ data items, we may just add dummy data items (i.e., all bits are 0 until we have $n$ data items) such that case 1 is applicable.

*Case 3*. If Basic IRIS has to store $m' > n$ data items, we introduce for tuples of $n$ data items a layer and for each layer we reuse the strategy applied in the previous cases. To be more specific, we proceed as follows: Let $\mathcal{K} \subseteq U$ be the set of all keys that have a data item in the system. The idea is to encode each $n$ data items with each other as described in case 1. That is, we partition the $|\mathcal{K}|$ data items into $O(|\mathcal{K}|/n)$ *layers* of $n$ data items each and encode the data items of each layer separately with each other using the encoding strategy described previously. Hence, we need to map each data item not only to one server but also to one of these layers. To be more precise, we use the following hashing strategy to distribute the data among the servers and layers: we use a hash function $f : U \to V$ to assign each data item to a server and a hash function $g : U \to \{0, \ldots, \gamma \cdot |\mathcal{K}|/n\}$ for some constant $\gamma$ to assign each data item to a layer. The goal is to choose these two hash functions such that the function $h : U \to V \times \{0, \ldots, \gamma \cdot |\mathcal{K}|/n\}, x \mapsto (f(x), g(x))$ is injective; that is, for every layer $i$ at most one key is assigned to each server. Given that this is the case, each layer will define a set of $n$ data items (when padded with dummies) with one per server, so we can apply the coding strategy described in case 1 to each of these layers.

The simplest way of realizing an injective $h$ with low storage overhead for $h$ (in fact, $\gamma = 2$ suffices) is to use cuckoo hashing [Pagh and Rodler 2001]: each data item has two optional positions, and they are distributed among these optional positions so that there is no collision. Of course, in this case, a lookup request for some data item $d$ would involve looking at both optional positions, but this would just double the work spent for the lookup operation described next, so in the following we just assume that $h$ is an injective hash function that can be directly evaluated to determine the unique server and layer of a datum. Furthermore, for simplicity, we assume in the following that $m \geq n$.

In the following, we examine the redundancy needed for the previously described encoding strategy. First, we consider the redundancy that occurs if at most $n$ data items need to be stored in the system. Let $d(s_i) = d_i \circ p_1(i) \circ \ldots \circ p_d(i)$. It is easy to prove by induction that $|p_\ell(i)| \leq (1 + 1/(k - 1))^{\ell - 1}|d_i|/(k - 1)$ for all $\ell \in \{1, \ldots, \log_k n\}$, which implies the following lemma under the assumption that $|d_i| = z$ for all $i$.

LEMMA 2.4. *For any $k > d$ and $z \geq k$, it holds that $|d(s_i)| \leq (1 + e)z$ for every server $s_i$.*

PROOF. By definition of $d(s_i)$, for each server $s_i$, it holds that

$$|d(s_i)| = |d_i| + \sum_{j=1}^{d} |p_j(i)| \leq |d_i| + \sum_{\ell=1}^{d} \left(1 + \frac{1}{k-1}\right)^{\ell-1} \cdot \frac{|d_i|}{k-1}.$$

Since $(1+x) \leq e^x$ for all $x \geq 0$, we get

$$|d(s_i)| \leq |d_i| + \sum_{\ell=1}^{d} e^{(\ell-1)/(k-1)} \cdot \frac{|d_i|}{k-1} \leq |d_i| \left(1 + \frac{d \cdot e^{(d-1)/(k-1)}}{k-1}\right).$$

Since $k > d$, this term is upper bounded by $(1+e)z$, which proves the lemma. □

Notice that $d = \log_k n = \log n / \log\log n$. Hence, in order to ensure $k > d$, we must choose $k > \log n / \log\log n$. For $n > 4$, it holds that $\log n > \log n / \log\log n$. Thus, for the rest of this section, we can fix $k$ to $\log n$. Then, for exactly $n$ data items of size at least $k$, we have a storage strategy with a constant redundancy.

Now assume more than $n$ data items have to be stored in the system. Then, if $\gamma$ is constant, Lemma 2.4 implies that the overall redundancy for this case is still constant.

Up to this point, we assumed in each case that each data item that is supposed to be stored in the system is mapped via a single hash function $h$ to a single server. But this is not yet enough for our lookup protocol to work. In our lookup protocol, we assume that each data item is cut into $c = \Theta(\log m)$ pieces, which are mapped to different servers. For this mapping, we will make use of $c$ hash functions $h_1, \ldots, h_c$ with the same properties as $h$ that together satisfy certain expansion properties. To avoid the redundancy to go up to $\Theta(c)$, we will assume that $z$ (the size of the data items) is at least $kc$.

Furthermore, our lookup protocol requires that $c/4$ of these pieces of a data item $d$ are sufficient to completely recover $d$. This approach can efficiently be realized by using Reed-Solomon codes [Reed and Solomon 1960]. That is, with Reed-Solomon codes, we can encode each data item $d$ into $c$ pieces such that $\sum_{i=1}^{c} |d_i| = O(|d|)$ and any $c/4$ of these pieces suffice to recover $d$. In order to minimize the overall redundancy to $O(1)$ (or $O(\log n)$ in case of Extended IRIS), we assume data items to have a size of $\Omega(\log n \log m)$. Here, the $\log n$ factor is needed due to our parity-based encoding scheme and $\log m$ is needed due to the Reed-Solomon codes we use. In summary, we obtain the following result.

COROLLARY 2.5. *When using Reed-Solomon codes, Basic IRIS has a constant redundancy.*

### 2.2. Lookup Protocol

In the following, we assume each nonblocked server receives at most one lookup request for a data item. The lookup protocol is supposed to describe which actions the nonblocked servers perform in order to return the requested data item. Recall that for each nonblocked server with a request for some data item $d$, it suffices to receive $c/4$ pieces of $d$ in order to recover $d$ and thus correctly answer the request.

The naïve approach in which each server with a request for a data item $d$ simply asks the servers $s_1, \ldots, s_c$ that hold the $c$ pieces of $d$ for these pieces does not work for the following reasons: First of all, all the servers $s_1, \ldots, s_c$ could be blocked, which disables them to answer any requests. In another scenario, the adversary could have sent all nonblocked servers a lookup request for the same data item $d$. In this case, each nonblocked server would contact the same servers $s_1, \ldots, s_c$, causing these servers to become congested and hence not be able to answer requests.

Hence, we need a more clever strategy to serve the requests, which is described in the following. This strategy will only be followed by the nonblocked servers; that is, the blocked servers do not send or receive anything. Let $s$ be a nonblocked server that received a lookup request for some data item $d$. The lookup protocol is divided into three stages: a *preprocessing stage*, a *probing stage*, and a *decoding stage*. In the preprocessing stage, the nonblocked servers determine a unique representative for each blocked server so that we can route in the $k$-ary butterfly as if all servers are still nonblocked (but, of course, the data in the blocked servers is lost). Also, information is collected that allows us to bound the work of decoding specific pieces of data items. In the probing stage, we issue read requests to the $c$ pieces of each $d_i$ and select $c/2$ of them to be decoded in the decoding stage.

*2.2.1. Preprocessing Stage.* The preprocessing stage consists of two further substages: the *butterfly completion stage* and the *decoding depth computation stage*.

*Butterfly Completion*. The goal of the butterfly completion stage is to make sure that for any DoS attack, the servers are reorganized such that we again have a complete $k$-ary butterfly but only over the nonblocked servers. This is done by determining for each blocked server $s$ a unique nonblocked server $s'$ that becomes the *representative* of $s$. That is, $s'$ will in the rest of the protocol take over the role of $s$ in all actions the server $s$ is supposed to perform. By this, $s'$ can only act as $s$, but it does not have access to the data stored at $s$. Once a server $s'$ becomes the representative of a blocked server $s$, it still has to be ensured that each other nonblocked server that is connected to $s$ via the underlying $k$-ary butterfly knows the representative $s'$ of $s$. Hence, in the rest of the lookup protocol, whenever a server $s$ is supposed to contact a blocked server $s'$, $s$ contacts the representative of $s'$ instead.

In short, the butterfly completion works as follows: First, a tree of depth $O(\log n/\log\log n)$ is built over all nonblocked servers. Afterward, the constructed tree is transformed into a doubly linked list $L$ of $n$ nonblocked servers. The created list $L$ will then be rearranged such that each nonblocked server with identifier $i$ is at position $i$ in $L$ and for each blocked server with identifier $j$ there is a nonblocked server at position $j$ in $L$ that is declared the *representative* of the blocked server. Finally, the resulting list is transformed into a $k$-ary butterfly such that at the end, each nonblocked server $s$ that is connected to a blocked server $s'$ in the initial $k$-ary butterfly is now connected to the representative of $s'$.

Since the implementation of the previous steps is rather straightforward and not of too big interest for the actual lookup, a detailed description of the butterfly completion stage is moved to the appendix.

Nevertheless, at the end of the butterfly completion stage, the following lemma holds.

LEMMA 2.6. *The butterfly completion stage guarantees that after* $(2+o(1))\log n$ *rounds and with a congestion of at most* $O(\log n)$ *at each nonblocked server, the following holds:*

—*For each blocked server, a unique nonblocked server as its representative is determined.*
—*Each server that is connected to a blocked server s via the k-ary butterfly knows the representative of s.*
—*Each nonblocked server is the representative of at most one blocked server.*

*Decoding Depth Computation*. Once the $k$-ary butterfly has been re-established, we can go ahead with collecting additional information. In particular, we are interested in the decoding work for specific data items. This is determined with the help of the following recursively defined function:

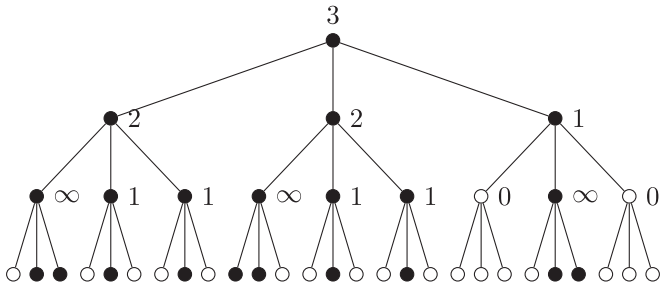


Fig. 3. Visualization of decoding depth computation with some nodes and edges of the $k$-ary butterfly omitted. Black/white colored nodes represent blocked/nonblocked nodes. The labels next to the nodes denote their decoding depth at the corresponding level.

*Definition* 2.7 (*Decoding Depth*). For a node $u = (\ell, x)$ of $BF(k, d)$, the *decoding depth* $dd(u)$ is defined as:

$$dd(u) = \begin{cases} 0 & \text{if } u \text{ is not blocked} \\ \infty & \text{if } \ell = d \text{ and } u \text{ is blocked} \\ \max\limits_{v \in C(u)} \{dd(v)\} + 1 & \text{if } \ell < d \text{ and } u \text{ is blocked,} \end{cases}$$

where $C(u)$ denotes the set of children of $u$ in $LT(u)$ excluding one child with the biggest decoding depth among these children. The *decoding depth of a server* $s_i$ is defined as $dd(s_i) = dd((0, i))$, and the *decoding depth of a subbutterfly* $BF(u)$ is defined as $dd(BF(u)) = \max_{(0,x) \in BF(u)} dd((0, x))$.

See Figure 3 for a visualization.

The decoding depth $d(u)$ of a butterfly node $u$ immediately implies an asymptotical upper bound on the time needed for restoring the data of a blocked server.

LEMMA 2.8. *If $dd(s_i) = \delta$ for a blocked server $s_i$, then any data item that has been assigned to $s_i$ can be restored in time $O(\delta)$ by the nodes in $BF((\delta, i))$.*

In a distributed fashion, the decoding depth is computed as follows: starting from level $\log_k n$, the servers compute the $dd(u)$-values of the butterfly nodes level by level and disseminate them among their neighbors in the next lower level until the $dd$-values of all nodes have been computed. This can certainly be done in $O(\log_k n)$ communication rounds with congestion $O(k)$ in each round. At the end, every server $s_i$ knows $dd(s_i)$. Then the servers compute the $dd(BF(\cdot))$-values level by level in a way that, starting in level 0, each node $u$ sends its $dd(BF(u))$-value to all of its neighbors $v$ in the next higher level, which will then be able to determine their $dd(BF(v))$-value by taking the maximum of the received values. Hence, at the end, every node $u$ (the server owning it, respectively) knows $dd(BF(u))$. This process also takes $O(\log_k n)$ communication rounds with congestion $O(k)$ in each round.

With Lemma 2.6 it follows:

LEMMA 2.9. *The preprocessing stage takes at most $(2 + o(1)) \log n$ communication rounds with at most $O(\log^2 n)$ congestion at every nonblocked server at each round.*

*2.2.2. Probing Stage.* With the probing stage, the actual lookup for the requested data items begin. Before we go into details of the probing stage, we will provide a short overview.

*Overview*. The idea of the probing stage is to forward a lookup request for each piece $d_i$ of a requested data item $d$ along $c$ paths from level $\log_k n$ to level 0 in the $k$-ary

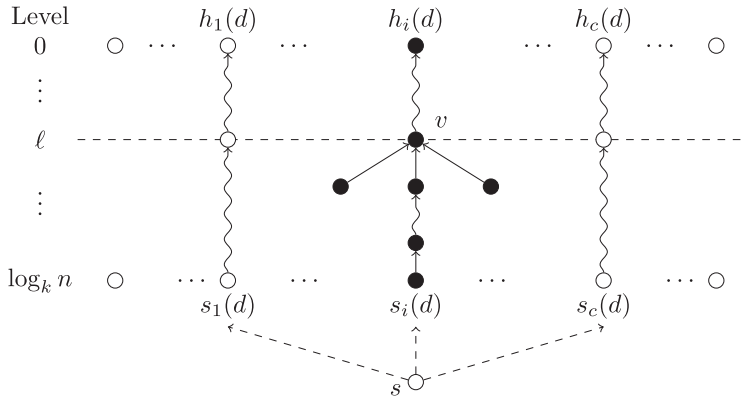


Fig. 4. Visualization of the probing stage. The curved lines denote the paths along which the probe messages are sent.

butterfly. The goal of this probing is to determine up to which level the $c$ requests (also denoted as probes) can be routed without a node on the paths becoming congested and without exceeding the decoding depth of one of the nodes. If enough probes reach level 0, the corresponding nodes can return the requested pieces to the server that issued the requests. This server can in turn recover the requested data item using the returned pieces. Otherwise, the request for $d$ will be assigned to a level $\ell \in \{1, \ldots, \log_k n\}$ at which not too many probes failed. Such a request will further be handled in the decoding stage.

*Details*. At the beginning of the probing stage, each nonblocked server $s$ that received a lookup request for some data item $d$ chooses $c$ nonblocked servers $s_1(d), \ldots, s_c(d) \in V$ uniformly and independently at random. This can simply be realized by selecting $c$ random servers in each round until $c$ nonblocked servers have been found (which takes $O(1)$ communication rounds w.h.p.).

The server $s$ then asks each server $s_i(d)$ to route a probe$(d, i)$ message along the unique path from the butterfly node on level $\log_k n$ emulated by $s_i(d)$ to the butterfly node on level 0 emulated by the server that holds $h_i(d)$. See Figure 4 for a visualization.

The actual probing takes place in synchronized rounds. The first $\log_k n + 1$ rounds work as follows. In round 0, all probe messages are active, and their *origin* is declared to be the server $s$ that initiated that probe. In round $r$, all probe messages that remain to be active are currently in a butterfly node $v$ at level $\log_k n - r$. First of all, $v$ checks the following rules:

—If the number of different $(d, i)$-pairs with a probe is more than $\alpha c$ (for a sufficiently large constant $\alpha$ defined later), then $v$ deactivates all probes and informs their origins about the level in which that happened. Such a node $v$ is called *congested*.
—If $dd(BF(v)) > \log_k n - r$, then $v$ deactivates all probes and informs their origins about the level in which that happened. Such a node $v$ is called *blocked*.

If none of the two rules apply, then $v$ distinguishes between two cases. If $\log_k n - r > 0$, then $v$ first combines, for those pairs $(d, i)$ with multiple probes, all of these probes into a single probe and declares itself as the new origin of that probe. Then $v$ forwards all probes to the next node on level $\log_k n - r - 1$ along their paths. If $\log_k n - r = 0$, that is, the probes have reached their destination at level 0 in the butterfly, $v$ delivers the requested data pieces of its probes to their origins (by using splitting if needed), who then can decode the data item using RS codes. These probes have been successful.

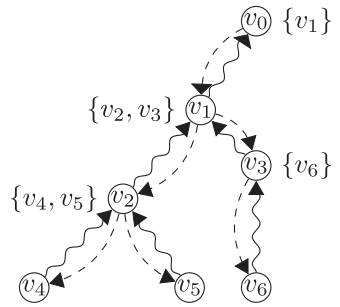


Fig. 5. Visualization of combining and splitting of messages. In this example the nodes $v_4, v_5, v_6$ send a probe message for the same pair $(d, i)$. The curved lines illustrate the combining of the probes, and the dashed lines illustrate the splitting of the responses. The sets next to some nodes denote the origins of the probes.

The splitting of the messages works as follows: Let $u$ be a node that is the origin of a probe $(d, i)$, but not the initial origin of that probe. Whenever $u$ receives a message concerning the probe $(d, i)$, $u$ forwards the message to all nodes it has previously (during the splitting process) stored as former origins of probe $(d, i)$. By this, within $O(\log_k n)$ communication rounds all servers get informed about which of their $c$ probes were successful or got deactivated at a level. See Figure 5 for a visualization of the combining of probes and the splitting of their responses.

If a server $s$ that is responsible for a lookup request for $d$ receives at least $c/2$ success messages, it can recover $d$ from the collected pieces (or it discovers that no data item exists in the system for the given search key in case there is a key mismatch) and is done, so it does not participate in the decoding stage any more. Otherwise, $s$ declares $d$ to *belong to level* $\ell$, where $\ell \in \{1, \ldots, \log_k n\}$ is the smallest level that contains at least $c/2$ active $(d, i)$ probes (i.e., $(d, i)$ probes that were not deactivated at level $\ell$ or earlier for $d$).

Notice that actually $c/4$ pieces of a data item suffice to decode the whole data item, but for the following reasons we use a bound of $c/2$ here: In the decoding stage, a server $s$ that receives a lookup request for a data item $d$ does not initiate a request for all $c$ pieces of $d$ but only for a subset $S$ of these pieces that have not been deactivated at the currently considered level in the probing stage. In the decoding stage, the size of this subset $S$ has to be chosen such that any half of the pieces suffice to recover $d$. This is given if we choose $|S| = c/2$.

It is easy to see that the probing stage satisfies the following property.

LEMMA 2.10. *The probing stage takes at most* $O(\log_k n)$ *communication rounds with at most* $O(\log^2 n)$ *congestion in every node at each round, w.h.p.*

Furthermore, one can show that if the adversary can block at most $(1/24) \cdot 2^{\log_k n}$ servers, then the number of data items with requests belonging to a level exponentially decreases in $k$, such that there is at most a logarithmic number of data items with requests belonging to the last level, level $\log_k n$. A detailed analysis of this fact is given in Section 2.3.1.

*2.2.3. Decoding Stage.* In the probing stage, each request has either been served or been assigned to a level $\ell \in \{1, \ldots, \log_k n\}$. In the decoding stage, the latter requests will be served by encoding appropriate subbutterflies. Again, we start with a short overview of this stage followed by a more detailed description.

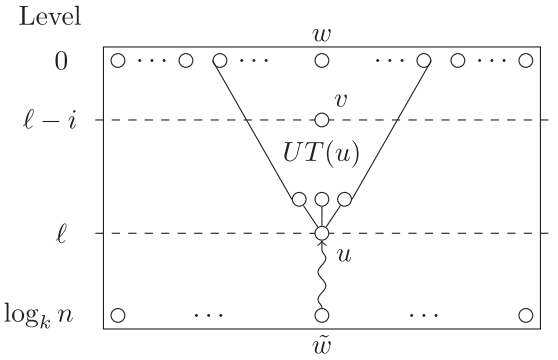


Fig. 6. Visualization of (round $i$ of) phase $r$ of the decoding stage.

*Overview*. The decoding stage proceeds in $\log_k n$ phases. Each phase $\ell \in \{1, \ldots, \log_k n\}$ is dedicated to the handling of all requests belonging to level $\ell$. That is, for each request for a data item $d$ belonging to level $\ell$, we first determine whether enough subbutterflies, which contain those $d_i$s that belong to level $\ell$, can be decoded. In that case, these subbutterflies will be decoded such that the server responsible for the request for $d$ receives enough pieces of $d$ and it can recover $d$ using Reed-Solomon codes. Otherwise, the request for $d$ is determined to belong to level $\ell + 1$, which will be handled in the next phase of the decoding stage.

*Details*. Each single phase $\ell \in \{1, \ldots, \log_k n\}$ is dedicated to decode the requests belonging to level $\ell$ and is divided into $O(\log_k n)$ rounds. In round 0 of phase $\ell$, each server $s$ that is responsible for a lookup request for some data item $d$ that belongs to level $\ell$ chooses a set $\mathcal{A}(s) \subseteq [c]$ of $c/2$ indices that were active at level $\ell$ in the probing stage. For such a server $s$ and $i \in \mathcal{A}(s)$, let $w$ be the butterfly node in level 0 responsible for $h_i(d)$ and let $\tilde{w}$ be the butterfly node in level $\log_k n$ emulated by $s_i(d)$. See Figure 6 for a visualization. Each such server $s$ then sends for each $i \in \mathcal{A}(s)$ a decode$(d, i)$ message to $s_i(d)$, which will then be forwarded (and possibly combined with other decode$(d, i)$ requests) along the unique path in the butterfly from $\tilde{w}$ to $w$ until it reaches a node $u$ on level $\ell$. From this point on, the decode$(d, i)$ request will be spread (and possibly combined with other requests to the same $(d, i)$ item on the way) to all nodes in $UT(u)$. This takes $\ell$ further rounds. The aim of this process is to determine whether $BF(u)$ can be decoded without causing nodes to become congested. For this purpose, in round $i$ of this spreading, each node $v$ at level $\ell - i$ determines whether it is congested. $v$ is *congested* if one of the following conditions is satisfied:

—The number of different $(d, i)$-pairs for which $v$ received a decode message at the beginning of this round is more than $\beta ck$ for a sufficiently large constant $\beta$.
—$v$ received a decode(cong) message at the beginning of this round.

If $v$ is not at level 0, it forwards a message to its children in $UT(v)$ depending on the following two cases:

(1) If $v$ is congested, then $v$ sends a decode(cong) message to its children in $UT(v)$.
(2) If $v$ is not congested, then $v$ first combines, for the (remaining) pairs $(d, i)$ with multiple decoding messages, all of those decoding messages to one message. Subsequently, $v$ forwards for all remaining $(d, i)$ pairs the message decode$(d, i)$ to its children in $UT(v)$.

In the following, we denote the subbutterfly $BF(v)$ of a node $v$ as *congested* if at least one node from $BF(v)$ receives more than $\beta ck$ decode messages for different $(d, i)$-pairs. By using the symmetry of the $k$-ary butterfly and the way messages are forwarded, it is easy to show the following lemma (Lemma 2.11).

LEMMA 2.11. *Let $u$ be a node at level $\ell$ that received a decode request in phase $\ell$. If $BF(u)$ is congested, then each node on level 0 of $BF(u)$ receives a* decode(cong) *message after at most $\ell$ rounds.*

Thus, by Lemma 2.11, after $\ell$ rounds the server that simulates $u$ knows whether $BF(u)$ is congested. If $BF(u)$ is congested, then $u$ informs the origins of the decode $(d, i)$ requests it received about that (by using splitting if needed). If $BF(u)$ is not congested, then $u$ initiates the decoding of $BF(u)$, which will recover the data pieces of all of the (at most $\beta ck$) remaining requests that arrived at $u$ within $O(\ell)$ communication rounds with a congestion of at most $\beta ck^2$ per node (by using the distributed decoding described in Section 2.1). The recovered pieces are then delivered to their origins (by using splitting if needed).

At the end of phase $\ell$, every server $s$ with a request belonging to level $\ell$ has received responses for all $i \in \mathcal{A}(s)$. If at least $c/4$ of these requests deliver decoded pieces, the server can recover its requested data item and is done. Otherwise, it changes its request to belong to level $\ell + 1$ so that it continues to be processed in the next phase. It is easy to see that the decoding stage satisfies the following property.

LEMMA 2.12. *The decoding stage takes at most $O(\log_k^2 n)$ communication rounds with at most $O(\log^3 n)$ congestion in every node at each round, w.h.p.*

Analogously to the probing stage, in the decoding stage, the number of data items with a request belonging to some level $\ell$ exponentially decreases in $k$. Hence, in the last phase of the decoding state, at most a logarithmic number of requests has to be handled. Obviously, this does not cause any congestion at any server such that at the end of the decoding stage, each lookup request has been served. A detailed analysis can be found in Section 2.3.2.

### 2.3. Analysis of Basic IRIS

In this section, we show that at the end of the lookup protocol, each lookup request has been served. The analysis is split into two parts: the analysis of the probing stage and the analysis of the decoding stage.

*2.3.1. Analysis of the Probing Stage.* In the following, we show that in the probing stage, not "too many requests" are declared to belong to the same level. To be more precise, we show that if the adversary can block at most $(1/24) \cdot 2^{\log_k n}$ servers, then the number of data items with requests belonging to a level exponentially decreases such that there is at most a logarithmic number of data items with requests belonging to the last level, level $\log_k n$.

LEMMA 2.13. *If the adversary can block at most $(1/24) \cdot 2^{\log_k n}$ servers, then for every $\ell \in \{1, \ldots, \log_k n\}$, the number of data items with requests belonging to level $\ell$ is at most $2\gamma n/k^\ell$ with $\gamma < 1/24$.*

Lemma 2.13 can be shown by adapting the analysis in Awerbuch and Scheideler [2007] (see Lemmas 4 and 5). Before we give a proof, we point out the main differences to the analysis in Awerbuch and Scheideler [2007] and introduce some required definitions and claims.

The new aspect that we need to exploit in the analysis is the fact that if $dd(BF(v)) > \ell$ for a node $v$ on level $\ell$, then at least $2^\ell$ nodes in $BF(v)$ must be blocked. This is because for every node $(i,x)$ in $BF(k,d)$ with $dd((i,x)) > \ell$ for some $\ell \leq \log_k n - i$, $LT((i,x))$ must contain a complete binary tree of blocked nodes of depth $\ell$ rooted at $(i,x)$ (Claim 2.14). This tree is also called a *witness tree* as it witnesses a high decoding depth of a node. Notice that due to the structure of the $BF(k,d)$, the leaves of such a witness tree must be distinct. Hence, if $dd(BF(v)) > \ell$ for a node $v$ on level $\ell$, then there must be a node $(0,x)$ in $BF(v)$ with a witness tree of depth $\ell$, which implies the lower bound on $2^\ell$ blocked nodes in $BF(v)$.

CLAIM 2.14. *Let $(i,x) \in BF(k,d)$ with $dd((i,x)) > \ell$ for $\ell \leq \log_k n - i$. Then, $LT((i,x))$ contains a witness tree of depth $\ell$.*

PROOF. The proof is by induction on $\ell$. For $\ell = 1$, let $u = (i,x) \in BF(k,d)$ with $dd(u) > 1$. Assume $LT(u)$ does not contain a complete binary tree of blocked nodes of depth 1. Then, at most one child of $u$ in $LT(u)$ is blocked, and therefore by definition of $dd(u)$, it holds that $dd(v) = 0$ for all $v \in C(u)$. Hence, $dd(u) = 1$, which contradicts $dd(u) > 1$. Suppose that for each node $u = (i,x) \in BF(k,d)$ with $dd(u) > \ell$ for $\ell < \log_k n - i$, $LT(u)$ contains a complete binary tree of blocked nodes of depth $\ell$. We show that the claim also holds for $\ell + 1$. Let $u = (i,x) \in BF(k,d)$ with $dd(u) > \ell + 1$. By definition of $dd(u)$, there exist at least two children $v$ and $w$ of $u$ in $LT(u)$ with $dd(v), dd(w) \geq dd(u) - 1 > \ell$. Then, by the induction hypothesis, $LT(v)$ and $LT(w)$ contain a complete binary tree $T_v$ and $T_w$ of blocked nodes of depth $\ell$. Notice that $LT(v)$ and $LT(w)$ are subtrees of $LT(u)$. Since $dd(u) \geq 1$, $u$ is also blocked, and the tree induced by connecting $u$ to the roots of $T_v$ and $T_w$ is a complete binary subtree in $LT(u)$ of blocked nodes of depth $\ell + 1$. □

In the following, we denote $BF(v)$ as *blocked* if the adversary blocks at least $2^\ell$ servers from $BF(v)$. $BF(v)$ is denoted as *congested* if the servers from $BF(v)$ receive in total more than $k^\ell \alpha c/2$ probes for different $(d,i)$ pairs in round $\ell$. For a server $s$ that received a lookup request for some data item $d$, we define $s_i^{(\ell)}(d)$ as the node at level $\ell$ on the unique path of length $\log_k n$ from $v$ to $w$, with $v$ being the butterfly node on level $\log_k n$ emulated by $s_i(d)$ and $w$ being the butterfly node in level 0 responsible for $h_i(d)$. A data item $d$ is called *blocked/congested* at level $\ell$ if there are blocked/congested $BF(s_{i_1}^{(\ell_1)}(d)), \ldots, BF(s_{i_r}^{(\ell_r)}(d))$ with $\ell_i \geq \ell$, $r = c/4$, and $i_1, \ldots, i_r$ being pairwise different.

We can now give an overview of the proof of Lemma 2.13. The idea of this proof is as follows: First, we show that whenever a probe$(d,i)$ is deactivated on a level $\ell$, then $BF(s_i^{(\ell)}(d))$ is blocked or congested (Claim 2.15). Moreover, if a data item $d$ is declared to belong to level $\ell$, then at least $c/2$ of its $(d,i)$-probes have been deactivated either because of blocked subbutterflies or because of congested subbutterflies at level $\ell$ or higher. Many data items that belong to level $\ell$ therefore imply many blocked or congested subbutterflies. But since only a limited fraction of them can be blocked or congested, we show that only a limited fraction of the data items can belong to level $\ell$. As a crucial ingredient for the proof, we require the hash functions $h_1, \ldots, h_c$ to satisfy a certain expansion property, which holds w.h.p. if the hash functions are chosen uniformly and independently at random (Claim 2.16).

CLAIM 2.15. *Whenever a $(d,i)$ pair is deactivated in a level $\ell \geq 0$ by a node $v$, then $BF(v)$ is blocked or congested, w.h.p.*

PROOF. If $(d,i)$ was deactivated due to $dd(BF(v)) > \ell$, then by Claim 2.14, $BF(v)$ contains at least $2^\ell$ blocked servers. Now assume $(d,i)$ was deactivated due to a too high congestion at $v$. Then, $v$ received in round $\ell$ probe messages for more than $\alpha c$ different $(d,i)$ pairs. Since the starting points for the lookup requests are chosen uniformly at

random, it holds that $E[|\mathcal{M}_\ell(w)|] = E[|\mathcal{M}_\ell(w')|]$ for all $w, w'$ at level $\ell$ in $BF(v)$ with $\mathcal{M}_\ell(w)$ being the set of $(d,i)$-pairs with probes received by node $w$. The Chernoff bounds [Chernoff 1952] can be applied, implying

$$Pr[|\mathcal{M}_\ell(w)| \geq (1+\delta)E[|\mathcal{M}_\ell(w)|]] \leq e^{-\min\{\delta,\delta^2\}E[|\mathcal{M}_\ell(w)|]/3}$$

for all $\delta \geq 0$ and all $w \in BF(v)$. Setting $\delta = 1/2$ gives $E[|\mathcal{M}_\ell(w)|] \geq 2\alpha c/3$ for all $w \in BF(v)$, w.h.p. Hence, the expected number of $(d,i)$-pairs with probes sent to $BF(v)$ is at least $2\alpha k^\ell c/3$, w.h.p. Furthermore, with $M$ being the number of $(d,i)$-pairs with probes sent to $BF(v)$, the Chernoff bounds imply that

$$Pr\left[M \leq \frac{2(1-\delta)}{3}\alpha c k^\ell\right] \leq e^{-\delta^2\alpha c k^\ell/3} \quad \text{for all } \delta \in [0,1].$$

With $\delta = 1/4$, we get that there are more than $\alpha c k^\ell/2$ $(d,i)$-pairs with probes sent to $BF(v)$, w.h.p. □

Hence, if a lookup request belongs to level $\ell$, then more than $c/2$ nodes that received a probe for this lookup request are congested or blocked at level $\ell - 1$, which together with Claim 2.15 implies that if a lookup request for some data item $d$ belongs to level $\ell$, then $d$ must be blocked or congested at level $\ell - 1$. In order to show that there cannot be too many of these data items, we introduce an expander property for collections of hash functions.

Recall that $U$ is the key universe and $m = |U|$. For any subbutterfly $B$, let $V(B)$ be the set of servers emulating the nodes of $B$. Let $\mathcal{H}$ be the collection of hash functions $h_1, \ldots, h_c$. Given a set $S \subset U$ of keys and a $k \in \mathbb{N}$, we call $F \subseteq S \times \{1, \ldots, c\}$ a *b-bundle* of $S$ if every $d \in S$ has exactly $b$ many pairs $(d,i)$ in $F$. Given $h_1, \ldots, h_c$ and a level $\ell \in \{0, \ldots, \log_k n\}$, let $\Gamma_{F,\ell}(S)$ be the union of the servers involved in these pairs at level $\ell$, that is, $\Gamma_{F,\ell}(S) = \bigcup_{(d,i)\in F} V(BF(s_i^{(\ell)}(d)))$. Given a $0 < \sigma < 1$, we call $\mathcal{H}$ a $(b,\sigma)$-*expander* if for any $\ell \leq \log_k n$, any $S \subseteq U$ with $|S| \leq \sigma n/k^\ell$, and any $b$-bundle $F$ of $S$, it holds that $|\Gamma_{F,\ell}(S)| \geq k^\ell |S|$.

In the following, we make use of the following claim, as similarly stated in Awerbuch and Scheideler [2007].

Claim 2.16. *If the hash functions $\mathcal{H} = \{h_1, \ldots, h_c\}$ are chosen uniformly and independently at random and $c \geq 12 \log m$, then $\mathcal{H}$ is a $(c/4, 1/24)$-expander, w.h.p.*

Although Awerbuch and Scheideler [2007] proved a similar version of Claim 2.16, for the sake of completeness we add the (adapted) proof here.

Proof. Suppose that, for randomly chosen functions $h_1, \ldots, h_c$, $\mathcal{H}$ is not a $(c/4, \sigma)$-expander. Then there exists an $i \leq \log_k n$, a set $S \subseteq U$ with $|S| \leq \sigma n/k^i$, and a $c/4$-bundle $F$ of $S$ with $|\Gamma_{F,i}(S)| < k^i|S|$. We claim that the probability $p_{s,i}$ that such a set $S$ of size $s$ exists is at most

$$\binom{m}{s}\binom{cs}{cs/4}\binom{n/k^i}{s}\left(\frac{s}{n/k^i}\right)^{cs/4}. \tag{1}$$

For the following reasons, Equation (1) holds: There are $\binom{m}{s}$ ways of choosing a subset $S \subset U$. Furthermore, there are $\binom{cs}{cs/4}$ ways of choosing $cs/4$ pairs $(d,j)$ for $F$ and at most $\binom{n/k^i}{s}$ ways of choosing a set $W$ of $s$ butterflies of dimension $i$ witnessing a bad expansion of the pairs in $F$. The fraction of collections $\mathcal{H}$ for which the selected pairs $(d,j)$ indeed have the property that $BF(s_j^{(i)}(d)) \subseteq W$ is equal to $(\frac{s}{n/k^i})^{cs/4}$ because the hash functions $h_1, \ldots, h_c$ are chosen independently and uniformly at random. Next we

simplify $p_{s,i}$. By using the conditions on $c$ and $\sigma$ in the lemma and with $c \geq 12 \log m$ and $m$ is sufficiently large, it holds that

$$\begin{aligned} p_{s,i} &\leq \binom{m}{s}\binom{cs}{cs/4}\binom{n/k^i}{s}\left(\frac{s}{n/k^i}\right)^{cs/4} \\ &\leq \left(\frac{em}{s}\right)^s (4e)^{cs/4}\left(\frac{en}{sk^i}\right)^s\left(\frac{sk^i}{n}\right)^{cs/4} = \left[\frac{em}{s}\cdot\left(4e^{1+4/c}\cdot\left(\frac{sk^i}{n}\right)^{1-4/c}\right)^{c/4}\right]^s \\ &\overset{(*)}{\leq} \left[m\cdot\left(4e^{1+4/c}\cdot\sigma^{1-4/c}\right)^{c/4}\right]^s \leq \left[m\cdot\left(\frac{1}{2}\right)^{c/4}\right]^s \leq \frac{1}{m^s}. \end{aligned}$$

In $(*)$, we used $\sigma \geq k^i s/n$. Hence, summing up over all possible values of $s$ and $i$, we obtain a probability of having a bad $c/4$-bundle of at most $(2\log_k n)/m$, which proves the lemma. □

We remark that the hash functions have to form a $(c/4, \sigma)$-expander for some constant $\sigma$ for our lookup protocol to work, but they do not have to be chosen at random. The previous proof just illustrates that if they are chosen at random, they will form a $(c/4, \sigma)$-expander w.h.p.

We are now ready to upper bound the number of congested and blocked data items at level $\ell$, which proves Lemma 2.13.

Proof of Lemma 2.13. We start with upper bounding the number of blocked data items. Let $S$ be a set of data items that are blocked at level $\ell$. We will show that $|S| < \gamma n/k^\ell$. Recall that a data item $d$ is blocked at level $\ell$ if at least $c/4$ of the subbutterflies $BF(s_i^{(\ell_i)}(d))$ are blocked with $\ell_i \geq \ell$; that is, each of these contains at least $2^{\ell_i}$ blocked servers. Adding the corresponding pairs $(d, i)$ to $F$, we obtain a $c/4$-bundle $F$ of $S$. Since a subbutterfly of level $\ell'$ contains $k^{\ell'}$ servers in total, by Claim 2.14, a $2^{\ell'}/k^{\ell'}$ fraction of the servers of a blocked subbutterfly of level $\ell'$ are blocked, which is at least $2^{\log_k n}/n$ for any $\ell' \leq \log_k n$. Therefore, if the adversary can only block up to $\varepsilon n$ servers with $\varepsilon < \gamma \cdot 2^{\log_k n}/n$, then the number of servers covered by all $BF(s_i^{(\ell_i)}(d))$ with $(d, i) \in F$ must be less than $\gamma \cdot n$. Since $\Gamma_{F_\ell(S)}$ is exactly the set of these servers, it holds that $|\Gamma_{F,\ell}(S)| < \gamma \cdot n$. On the other hand, we know from Claim 2.16 that for any $c/4$-bundle $F$ of $S$ with $|S| \leq (1/24)n/k^\ell$, $|\Gamma_{F,\ell}(S)| \geq k^\ell |S|$. Assume there is a set $S$ of blocked data items of size $\gamma n/k^\ell$. Then, according to Claim 2.16, $|\Gamma_{F,\ell}(S)| \geq \gamma n$, which is not possible. Hence, the number of blocked data items at level $\ell$ is less than $\gamma n/k^\ell$.

Similarly, the number of congested data items can be upper bounded. Let $S$ be a set of data items that are congested at level $\ell$. Analogously to the case of blocked data items, we can construct a $c/4$ bundle $F$ of $S$. First, we show that for a sufficiently large $\alpha$, there exists less than a $\gamma$-fraction of congested subbutterflies on level $\ell$ for all $\ell \in \{0, \ldots, \log_k n\}$. Recall that a subbutterfly on level $\ell$ is congested if it receives more than $\alpha c k^\ell/2$ probes for different $(d, i)$ pairs. Since there are at most $(1-\varepsilon)n$ lookup requests in total, at most $c(1-\varepsilon)n$ probes arrive at level $\ell$. Thus, at most $c(1-\varepsilon)n/(\alpha c k^\ell/2) = 2(1-\varepsilon)n/(\alpha k^\ell)$ subbutterflies can be congested at level $\ell$. Since there are exactly $n/k^\ell$ disjoint subbutterflies at level $\ell$, the fraction of congested subbutterflies at level $\ell$ is upper bounded by $\frac{2(1-\varepsilon)n/(k^\ell\alpha)}{n/k^\ell} = 2(1-\varepsilon)/\alpha$. Hence, for $\alpha > 2(1-\varepsilon)/\gamma$, at most a $\gamma$-fraction of the subbutterflies on level $\ell$ is congested for all $\ell \in \{0, 1 \ldots, \log_k n\}$. That is, all of the congested subbutterflies $BF(s_i^{(l_i)}(d))$ with $(d, i) \in F$ together contain at most a $\gamma$-fraction of the subbutterflies on level $\ell$. Again, with Claim 2.16, we can deduce that $|S| < \gamma n/k^\ell$ for $\gamma < 1/24$. Hence, all in all, only less than $(2\gamma)n/k^\ell$ data

items can be blocked or congested at level $\ell$, implying an upper bound of $n/k^\ell$ for the number of data items with lookup requests that belong to level $\ell$. □

*2.3.2. Analysis of the Decoding Stage.* Analogously to Lemma 2.13, for the decoding stage, the following lemma holds:

LEMMA 2.17. *For every phase $r$, the number of data items with requests belonging to level $r$ is at most $\varphi n/k^r$ with $\varphi = \Theta(k)$.*

PROOF. We show the lemma by induction on $r$. For $r = 1$, the claim obviously holds. In the following, let $\gamma = 1/24$ and $\varphi = \gamma(k+2)$. The proof of the induction step is similar to the proof of Lemma 2.13. We start with determining the number of congested subbutterflies of dimension $r$. If the subbutterfly $BF(u)$ of a node $u$ is congested, then by definition, there exists a node in $BF(u)$ that receives more than $\beta ck$ decode messages for different $(d,i)$-pairs. Since $\varphi/(2\gamma k) < 3 = \beta$, it holds that $\beta ck > \varphi c/(2\gamma)$, implying that a congested subbutterfly $BF(u)$ of a node $u$ receives more than $\varphi c/(2\gamma)$ decode messages for different $(d,i)$-pairs. By the induction hypothesis, there are at most $\varphi n/k^r$ messages with requests for different data items in level $r$. For each lookup request that belongs to level $r$, $c/2$ decode messages are sent. Hence, in total, there are less than $\varphi n/k^r \cdot c/2 \cdot 2\gamma/\varphi c = \gamma n/k^r$ congested subbutterflies of dimension $r$. Let $S$ be a set of congested data items at level $r$. Similarly to the proof of Lemma 2.13, there exists a $c/4$-bundle $F$ for $S$. Since there are less than $\gamma n/k^r$ congested subbutterflies of dimension $r$, it holds that $|S| < (1/24)n/k^r$. And since each subbutterfly of dimension $r$ contains $k^r$ nodes, less than $\gamma n$ servers simulate a node of a congested subbutterfly of dimension $r$, that is, $|\Gamma_{F,r}(S)| < \gamma n$. On the other hand, Claim 2.16 can be applied, giving $|\Gamma_{F,r}(S)| \geq k^r|S|$, and it follows that $|S| \leq \gamma n/k^r$. Even if all requests for congested data items do not finish in round $r$, together with the number of requests belonging to level $r+1$ from the probing stage, by Lemma 2.13 there are at most $2\gamma n/k^{r+1} + \gamma n/k^r$ data items with requests that participate in phase $r+1$ of the decoding stage. For $\varphi = \gamma(k+2)$, this term is upper bounded by $\varphi n/k^{r+1}$, which proves the lemma. □

Hence, less than $\Theta(k)$ data items with lookup requests participate in the last phase, phase $\log_k n$, of the decoding stage and therefore each node receives in this phase decoding requests for less than $\Theta(k)$ different data items. Thus, there cannot be a congested subbutterfly any more. This, together with the fact that the decoding depth of $BF(k,d)$ must be less than $\log_k n$ when blocking at most $\gamma 2^{\log_k n} = \gamma n^{1/\log\log n}$ nodes with $\gamma < 1$, implies that all remaining data items can be decoded at the end.

Lemma 2.9, Lemma 2.10, Lemma 2.12, and Lemma 2.17 imply the following theorem.

THEOREM 2.18. *If the adversary blocks less than $1/24 \cdot \log n/\log\log n$ many servers, then, by using only a constant redundancy, any set of lookup requests (one per non-blocked server) is served correctly after at most $O(\log^2 n)$ communication rounds with a congestion of at most $O(\log^3 n)$ at every node in each round, w.h.p.*

## 3. ENHANCED IRIS

In the following, we extend the previously presented Basic IRIS system to the Enhanced IRIS, which is able to handle up to a constant fraction of the servers with a redundancy of $O(\log n)$. Before we describe the encoding strategy of Enhanced IRIS (Section 3.2) and the lookup protocol (Section 3.3), we need to introduce some further preliminaries (Section 3.1).

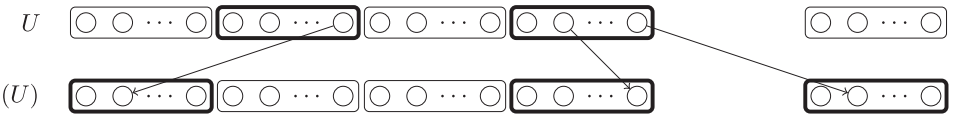


Fig. 7. Visualization of a permutation $\pi$ with expansion 2/3. The upper nodes denote the order of the nodes in $U$; the lower nodes denote the order of the nodes in $\pi(U)$. The rounded rectangles around the nodes denote the groups. The thick rounded rectangles around the upper nodes denote the groups in $S$.

### 3.1. Preliminaries

Instead of using a simple parity coding strategy to recover from any blocked server within a $k$-block, we need a more complex coding strategy that can recover from any two blocked servers within a $k$-block. Here, we can use the EVENODD scheme proposed in Blaum et al. [1994]. EVENODD is a 2-erasure correcting code that only uses exclusive OR operations and is optimal in terms of redundancy.

When using this scheme, we obtain the following results.

LEMMA 3.1. *For any $k$-block $B$ with node sets $(\ell, x_1), \ldots, (\ell, x_k)$ and $(\ell+1, x_1), \ldots, (\ell+1, x_k)$ in which at most two $(\ell+1, x_j)$ are blocked, the information in the remaining nodes $(\ell+1, x_i)$ suffices to recover $d(\ell, x_1), \ldots, d(\ell, x_k)$.*

In order to encode $k$ data items of equal size $z$ with each other, EVENODD adds in total $2z$ parity bits. That is, $2z/(k-2)$ parity bits (up to an additive 1) are assigned to each server that holds one of the $d$ data items, which implies the following lemma (Lemma 3.2).

LEMMA 3.2. *For any $k$-block $B$ with node sets $(\ell, x_1), \ldots, (\ell, x_k)$ and $(\ell+1, x_1), \ldots, (\ell+1, x_k)$, it holds that $|d(\ell+1, x_i)| \leq (1+2/(k-2))|d(\ell, x_i)|$ up to an additive 1.*

Another aspect in which the Enhanced IRIS system deviates from the Basic IRIS system is that the $k$-blocks are no longer organized in a $k$-ary butterfly. Instead, we make use of permutations with certain expansion properties.

*Definition* 3.3. Let $U$ be a set of $N$ nodes that are organized into $N/K$ groups of $K$ consecutive nodes. A permutation $\pi : U \to U$ is said to *have an expansion of $\gamma$* if for any subset $S$ of at most $N/[12K^5]$ groups and any subset $W$ of nodes with exactly three nodes in each group in $S$ it holds that $\pi(W)$ contains nodes from at least $\gamma|S|$ many groups from $\pi(U)$.

In other words, a permutation $\pi$ with expansion $\gamma$ on a set $U$ guarantees that the number of blocked $k$-blocks in $\pi(U)$ decreases by a factor of at least $\gamma$ in comparison to the number of $k$-blocks in $U$. See Figure 7 for a visualization.

One can show that for sufficiently large $N$, there always exists a permutation on $U$ with expansion at least $5/4$.

LEMMA 3.4. *Define $U$ as in Definition 3.3. For any $N$ and $K$ with $N \geq 12K^5$, there is a permutation on $U$ with expansion at least $(1+\delta)$ for a constant $\delta \geq 1/4$.*

PROOF. Let the permutation $\pi$ be chosen uniformly at random from all permutations on $U$. Let $p(s)$ be the probability that there exists a set $S$ of groups with $|S| = s$ and a set of triples $W$ from these groups such that $\pi(W)$ contains at most $(1+\delta)|S|$ many groups. We will show that $p(s) < 1$, which proves the lemma. $p(s)$ can be upper bounded by

$$p(s) \leq \binom{N/K}{s}\binom{K}{3}^s \binom{N/K}{(1+\delta)s}\left(\frac{(1+\delta)s}{N/K}\right)^{3s}, \tag{2}$$

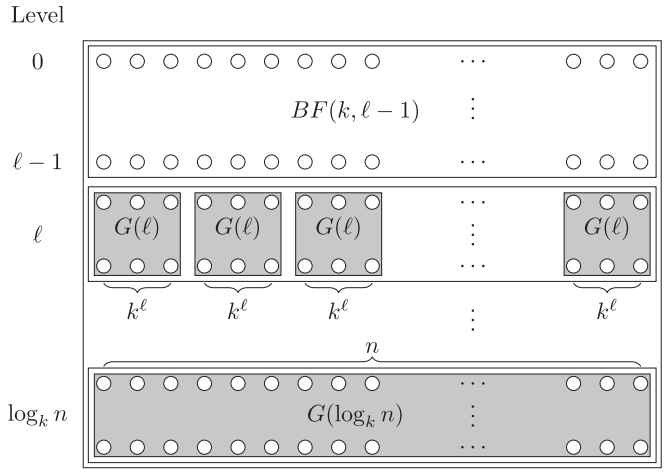


Fig. 8. Visualization of the underlying topology used in the Enhanced IRIS system, where $\ell$ denotes the first level with $k^\ell \geq 12(\log k)^5$.

where $\binom{N/K}{s}$ is the number of possibilities for choosing $s$ groups, $\binom{K}{3}^s$ is the number of possibilities of choosing a triple in each of the selected groups, $\binom{N/K}{(1+\delta)s}$ is the number of possibilities for choosing $(1+\delta)s$ groups that the triples have to map to, and $\left(\frac{(1+\delta)s}{N/K}\right)^{3s}$ is an upper bound on the probability that all of the triples are indeed mapped to the $(1+\delta)$ groups. When choosing $s = \gamma N/K$, Equation (2) is at most

$$\left(\frac{e}{\gamma}\right)^s \left(\frac{K^3}{6}\right)^s \left(\frac{e}{(1+\delta)\gamma}\right)^{(1+\delta)s} ((1+\delta)\gamma)^{3s} = \left(\frac{e^{2+\delta}(1+\delta)^{2-\delta}}{6}\right)^s (K^4\gamma)^{(1-\delta)s}.$$

When choosing $\delta = 1/4$ and $\gamma \leq 1/(12K^4)$, the first term is at most $4^{(1-\delta)s}$ and the second term is at most $(1/12)^{(1-\delta)s}$, so altogether, $p(s) \leq (1/3)^{(1-\delta)s}$. When summing up over all $s \geq 1$, this gives an overall probability of less than 0.8 that the expansion of $\pi$ is at most $(1+\delta)$, which completes the proof. □

### 3.2. Encoding

Similarly to the encoding in Basic IRIS, we introduce an underlying topology consisting of $\log_k n$ levels each containing $n$ nodes. The description of this topology is divided into three parts depending on $\ell$.

*Part 1*. $k^\ell < 12(\log k)^5$
*Part 2*. $k^\ell \geq 12(\log k)^5$ and $\ell < 6$
*Part 3*. $k^\ell \geq 12(\log k)^5$ and $\ell \geq 6$

Figure 8 provides a high-level overview of the encoding of the different levels.

*Part 1*. For the encoding of the data for all levels $\ell$ with $k^\ell < 12(\log k)^5$, we just use the encoding via $k$-ary $\ell$-dimensional subbutterflies as in Basic IRIS. Since for these levels $k^\ell$ is a constant, these levels can tolerate a constant fraction of blocked nodes in each subbutterfly while still being able to decode all data.

In case of $k^\ell \geq (12 \log k)^5$ (Part 1 and Part 2), we introduce $n/k^\ell$ graphs for each level $\ell$, denoted as $G(\ell)$, which are divided into further sublevels, each consisting of $k^\ell$ nodes. We encode the data items within each $G(\ell)$ by using edge sets between the sublevels with certain expansion properties (defined later). In particular, for each level $\ell$, we do

not proceed with the encoding of the resulting data items from the last (sub)level of the graphs from level $\ell - 1$ but we restart the encoding in level $\ell$ with the original data items.

This level-based encoding approach allows us to define the decoding depth of Enhanced IRIS analogously to the decoding depth of Basic IRIS.

*Definition* 3.5 (*Decoding Depth*). The *decoding depth of a node* $u$ at a sublevel $i$ of $G(\ell)$ is now defined as follows:

$$dd(u) = \begin{cases} 0 & \text{if } u \text{ is not blocked} \\ \infty & \text{if } i = L \text{ and } u \text{ is blocked} \\ \max\limits_{v \in C(u)} \{dd(v)\} + 1 & \text{if } i < L \text{ and } u \text{ is blocked,} \end{cases}$$

where $C(u)$ denotes the neighbors of the $K$-block of $u$ in level $i + 1$ excluding any two nodes of the biggest decoding depth among these neighbors. Analogously to the Basic IRIS system, the *decoding depth of a server* $s_i$ is defined as $dd(s_i) = (d(0, i))$.

Note that if the decoding depth of a node $u$ at sublevel $i$ in $G(\ell)$ is more than $d$ with $i + d \leq L$, then it must be possible to embed a complete *ternary* tree of blocked nodes with root $u$ and depth $d$ in $G(\ell)$.

*Part 2*. Now, suppose that $\ell$ satisfies $k^\ell \geq 12(\log k)^5$ and $\ell < 6$. In order to describe the encoding of the data items on such a level $\ell$ for each consecutive $k^\ell$ nodes on level $\ell$, we introduce a graph $G(\ell)$ that consists of $L_1 = 20 \log k$ sublevels, each consisting of $k^\ell$ virtual nodes. In order to construct $G(\ell)$, choose a permutation $(\pi_1)_\ell$ that has an expansion of at least 5/4 for $N = k^\ell$ and $K_1 = \log k$. In the following, let $(i, x)$ denote the virtual node from sublevel $i$ and column $x$ in $G(\ell)$. Partition the nodes of each sublevel of $G(\ell)$ into groups of $K$ consecutive nodes. Each node $(i, x)$ in some group $B$ in sublevel $i$ of $G(\ell)$ is connected to all nodes $(i + 1, \pi_\ell(y))$ with $(i, y) \in B$. This establishes complete bipartite graphs of $K_1$ nodes on sublevel $i$ and $i + 1$, called $K_1$-*blocks* (see Figure 9). $G(\ell)$ is simulated by $N$ servers with server $s_i$ simulating the $L_1$ nodes $(0, i)$, $(1, \pi_\ell(i))$, $(1, \pi_\ell(\pi_\ell(i)))$, and so on.

We are now ready to describe the encoding of a set of $K$ data items $d_0, \ldots, d_{K-1}$. Initially, $d_j$ is placed in node $(0, j)$ of $G(\ell)$, for all $j \in \{0, \ldots, K - 1\}$. Given that in sublevel $i$, $i \in \{0, \ldots, L - 1\}$, we have already assigned data items $d(i, j)$ to the nodes $(i, j)$, $j \in \{0, \ldots, K - 1\}$, we compute, for each $K$-block $B$ of sublevel $i$, the data items for sublevel $i + 1$ using the EVENODD coding strategy and assign them to the nodes of that $K$-block in sublevel $i + 1$.

In the following, our goal is to extend $G(\ell)$ by adding more sublevels to it such that whenever a data item encoded in $G(\ell)$ cannot be recovered, then at least a constant fraction of the server emulating $G(\ell)$ must be blocked.

Suppose the data of a node $(0, x)$ in $G(\ell)$ cannot be recovered; that is, the decoding depth of $(0, x)$ is larger then $L_1$. Hence, $G(\ell)$ contains a ternary tree with root $(0, x)$ and depth $L_1$ that only consists of blocked nodes. Unfortunately, the leaves of this tree are not guaranteed to be distinct any more as they are for the binary witness trees in the $k$-ary butterfly. But due to the expansion property of $\pi_\ell$, we know that this ternary tree must cover at least $3(5/4)^{L-1}$ blocked servers at its leaves. Thus, at least $\min\{3(5/4)^{L_1-1}, N/(12(K_1)^5)\} \geq N/(12(K_1)^5)$ blocked servers are covered by the nodes at sublevel $L$ in $G(\ell)$, since $L_1 = 20 \log k \geq 4\ell \log k$ and $(5/4)^{4\ell \log k} \geq k^\ell$. That is, at least a $1/(12(K_1)^5)$-fraction of the $N$ servers simulating $G(\ell)$ is blocked. Now consider the following two cases:

*Case 1*. $(K_1)^\ell \leq 12(\log K_1)^5$. In this case, $K_1$ is a constant and therefore a constant fraction of the servers simulating $G(\ell)$ is blocked.

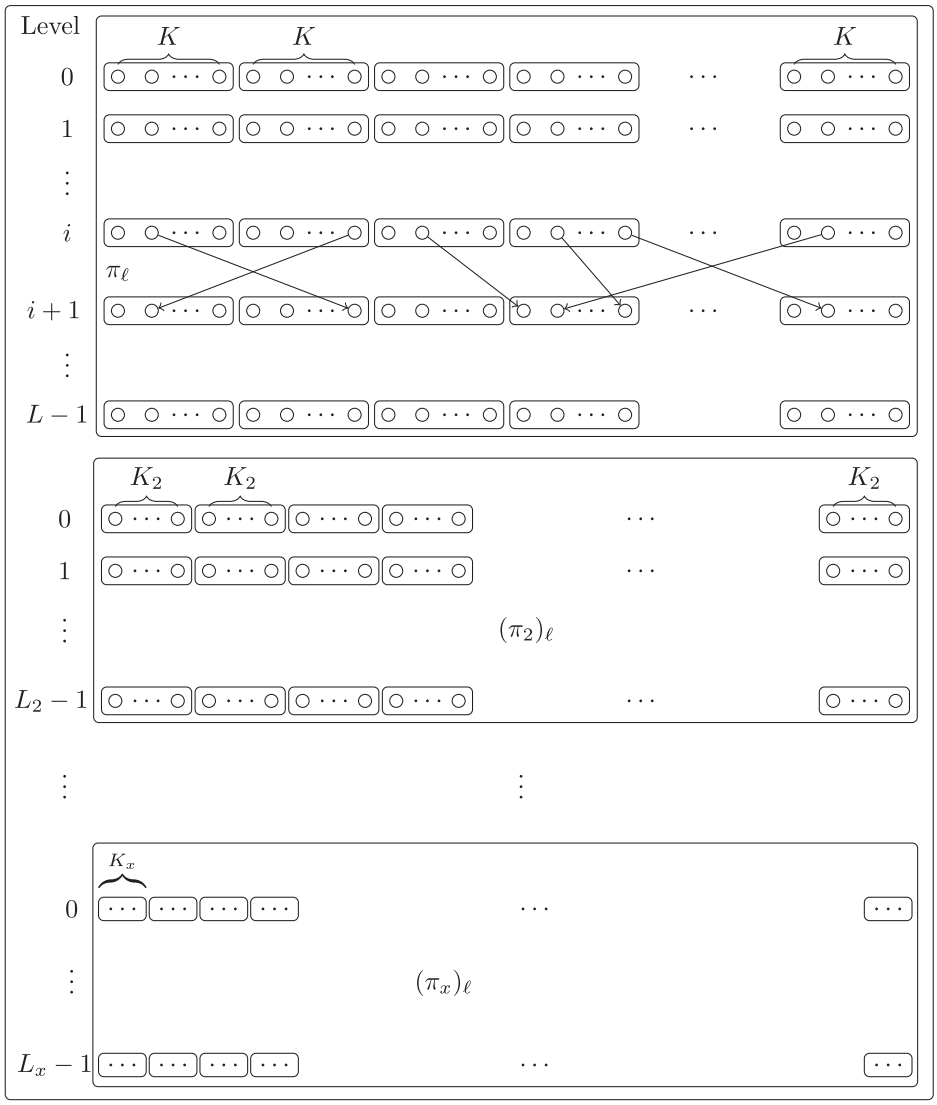


Fig. 9. Visualization of $G(\ell)$ and its $K_1$-blocks with $x \leq \log^* k$.

*Case 2*. $(K_1)^\ell > 12(\log K_1)^5$. Define $K_2 = \log K_1 = \log\log k$, $L_2 = 20 \log K_1$ and add $L_2$ additional sublevels to $G(\ell)$ using a permutation $(\pi_2)_\ell$ with expansion 5/4 for $N$ and $K_2$. With the same arguments from earlier, the number of blocked servers that are now covered by the nodes at sublevel $L_1 + L_2$ in $G(\ell)$ increases to at least $N/(12(K_2)^5)$. Again, we need to consider two cases: If $(K_2)^\ell \leq 12(\log K_2)^5$, then, analogously to case 1, $K_2$ is a constant implying that a constant fraction of the servers emulating the $G(\ell)$ is blocked.

If $(K_2)^\ell > 12(\log K_2)^5$, we continue with the extension of $G(\ell)$ as for $K$. That is, define $K_3 = \log K_2$, $L_3 = 20 \log K_2$ and add $L_3$ additional sublevels to $G(\ell)$ by using a permutation $(\pi_3)_\ell$ with expansion 5/4 for $N$ and $K_3$.

We continue with the process of extending $G(\ell)$ until after at most $\log^* k$ extensions,[2] $K_i$ is a constant, implying that a constant fraction of the servers emulating $G(\ell)$ is blocked whenever the data of some node $(0, x)$ in $G(\ell)$ cannot be recovered.

*Part 3*. It remains to describe the encoding of the data items on level $\ell$ with $k^\ell \geq 12(\log k)^5$ and $\ell \geq 6$. Similarly to the construction in part 2, we first choose a graph $G(\ell)$ using a permutation $\pi_\ell$ with $N = k^\ell$, $K = k$, and $L = 4 \log N$ levels to get the number of blocked servers in $G(\ell)$ up to $N/(12K^5)$. Then, we continue with the $\log^*$-construction as in part 2 (starting with $K = \log k$ and $L = 20 \log k$), until we get the number of blocked servers up to a constant fraction before the decoding of a data item can fail.

Lemma 3.6 provides an upper bound for the overall redundancy of the Enhanced IRIS system.

LEMMA 3.6. *The overall redundancy in the Enhanced IRIS system is* $O(\log n)$.

PROOF. In the following, let $z$ denote the size of the original data items. First, we upper bound the amount of information a nonblocked server $s$ stores for the encoding of one $G(\ell)$. Let $S_1(i)$, $i \in \{0, \ldots, L_1 - 1\}$ denote the amount of information server $s$ has stored during the encoding of the first $L_1$ sublevels of $G(\ell)$ up to sublevel $i$. By Lemma 3.2, it holds that $S(i) \leq S(i-1)(1 + 2/(k-2))$. Since $S(0) = z$, we get $S(i) \leq z(1 + 2/(k-2))^i$. Hence, the amount of data $S(L_1 - 1)$ the server $s$ stores after the encoding of the first $L_1$ sublevels of $G(\ell)$ is upper bounded by $z(1+2/(k-2))^{L_1-1} = O(z)$. By the same arguments, the amount of data the server $s$ stores after the encoding of the first $\sum_{i=1}^{x} L_i$ sublevels of $G(\ell)$ is upper bounded by $\Pi_{i=1}^{x} z(1 + 2/(k-2))^{L_i-1}$. Since there are at most $\log^* k$ level extensions in $G(\ell)$, the overall amount of data the server $s$ stores after the encoding of one complete $G(\ell)$ graph is upper bounded by $\Pi_{i=1}^{\log^* k} z(1 + 2/(k-2))^{(L_i-1)} = z \cdot 2^{O(\log^* k)}$. Hence, for all $\log_k n$ graphs $G(\ell)$, the server overall amount of data stored at $s$ after the complete encoding of the underlying topology used in Enhanced IRIS is upper bounded by $z \log_k n \cdot 2^{O(\log^* k)} = O(z \log n)$. This implies an overall redundancy of $O(\log n)$ at each server. □

### 3.3. The Lookup Protocol

Recall that the lookup protocol of Basic IRIS consists of three stages: the preprocessing stage, the probing stage, and the decoding stage. In the following, we describe how to adapt these stages in order to work for the Enhanced IRIS system.

*3.3.1. Preprocessing Stage.* Just like in the Basic IRIS system, we first recover the $k$-ary butterfly, such that at the end each nonblocked server knows the representative of each blocked server it is connected to in the $k$-ary butterfly. Next, the $G(\ell)$ graphs need to be recovered. That is, each nonblocked server needs to know the representative of each blocked server it is connected to in any $K_i$-block of a $G(\ell)$ graph. Since there is now a deterministic calculation rule for the $K_i$-blocks in the $G(\ell)$ graphs, each server additionally needs to store all permutations $(\pi_i)_\ell$. Recall that in total we only have $O(\log^* n)$ permutations, implying that each server additionally needs to store an amount of $O(n \log^* n)$ data. But this does not cause a problem, since in any case each server additionally stores information about each other server in the system (e.g., addresses of the servers). In order to recover the $G(\ell)$ graphs, each nonblocked server $s$ sends a message to each server $s'$ it is connected to in any of the $G(\ell)$ graphs by routing the message $(\mathsf{id}(s), \mathsf{id}(s'))$ along the unique path in the $k$-ary butterfly from $s$ to $s'$. Since the $k$-ary butterfly has already been recovered correctly, eventually this message reaches the server $s'$ in case $s'$ is not blocked and otherwise the representative $\mathsf{rep}(s')$ of $s'$. This

[2] $\log^*(n)$ is the number of times the logarithm has to be applied to $n$ until the result is at most 2.

initiates $s'$ (or $\mathsf{rep}(s')$) to forward the message $(\mathsf{id}(s'), ID)$ back to $s$ along the unique path from $s'$ to $s$ in the $k$-ary butterfly, where $ID = id(s')$ in case $s'$ is not blocked and $ID = \mathsf{id}(\mathsf{rep}(s'))$ otherwise. Since all nonblocked servers forward their messages in the $k$-ary butterfly, by the Borodin Hopcraft bound [Borodin and Hopcroft 1982], a congestion of $\Omega(\sqrt{n}/k)$ may occur at a nonblocked server. Using the analysis of Valiant's trick [Valiant 1982], one can show that there exist permutations with the desired expansion properties (as defined earlier) that additionally guarantee a congestion of at most $O(\log n)$ at each node in each round of the routing strategy described earlier. Hence, at the end of the preprocessing stage, each nonblocked server knows the representatives of all blocked servers it is connected to in the $k$-ary butterfly and in all $G(\ell)$ graphs.

Once the underlying topology has been recovered, the decoding depth of the nodes in $G(\ell)$, as defined in Definition 3.5, can be computed analogously to the decoding depth in Basic IRIS. Besides the decoding depth of a node at the levels in $G(\ell)$, each node from each $G(\ell)$ also needs to compute the decoding depth of $G(\ell)$, which is analogously defined to the decoding depth of a subbutterfly in Basic IRIS. That is, $dd(G(\ell)) = \max\{dd(u) \mid u \text{ is a node on level 0 in } G(\ell)\}$. Since the $k$-ary butterfly has already been recovered, we can use a bottom-up routing in the $k$-ary butterfly consisting of the servers that emulate the nodes in $G(\ell)$ in order to determine the maximum decoding depth of a node from $G(\ell)$. To be more precise, let $s_1, \ldots, s_{k^\ell}$ denote the servers that emulate $G(\ell)$. In the first round, each server $s_i$ forwards the decoding depth of the node it emulates on the last level in $G(\ell)$ to all servers it is connected to on level $\log_k n - 1$ of the $k$-ary butterfly. In round $r \in \{2, \ldots, \log_k n\}$, each server $s_i$ forwards the maximum of the decoding depths it has received at the beginning of this round and the decoding depth of the node it emulates on level $\log_k n - r + 1$ in $G(\ell)$ to all servers it is connected to on level $\log_k n - r$ of the $k$-ary butterfly.

Hence, after $\log_k n$, rounds each server that emulates a node from $G(\ell)$ is aware of $dd(G(\ell))$.

The following lemma (Lemma 3.7) is easy to check.

LEMMA 3.7. *The preprocessing stage takes at most $O(\log n)$ communication rounds with at most $O(\log^2 n)$ congestion at every nonblocked server at each round. Furthermore, at the end of the preprocessing stage, the following holds:*

(1) *Each nonblocked server knows the representatives of all blocked servers it is connected to in the $k$-ary butterfly and the ones it is connected to in any of the $G(\ell)$ graphs.*
(2) *Each nonblocked server that emulates a node from $G(\ell)$ knows $dd(G(\ell))$.*

*3.3.2. Probing Stage.* The purpose of the probing stage is to determine for each lookup request the level $\ell \in \{0, \ldots, \log_k n\}$ it "belongs to" (as defined later). Analogously to the Basic IRIS system, the probing stage consists of $\log_k n$ rounds. First, each nonblocked server that received a lookup request for some data item $d$ chooses $c$ nonblocked servers $s_1(d), \ldots, s_c(d)$ just like in the probing stage of the Basic IRIS system. The following rounds are dedicated to forward for each $i \in \{1, \ldots, c\}$ a $(d, i)$ probe along the unique path in the $k$-ary butterfly from the node on level $\log_k n$ that is emulated by $s_i(d)$ to the node on level 0 that is emulated by the server responsible for $h_i(d)$. In each round $r \in \{0, \ldots, \log_k n\}$, each node $u$ that received a probe message determines whether it is *congested* or *blocked*. Analogously to the Basic IRIS system, a node $u$ that received probe messages is denoted as *congested* if it receives more than $\alpha \log n$ probes for different $(d, i)$ pairs (for a sufficiently large constant $\alpha$) in the current round. Different from the Basic IRIS system in the Enhanced IRIS system, we denote $u$ as *blocked* at level $r$ if $dd(G(u)) = \infty$, where $G(u)$ denotes the graph $G(\log_k n - r)$ that is (besides other servers) emulated by the server that emulates $u$. If $u$ is congested or blocked, $u$ deactivates all

$(d, i)$-probes it received in the current round by sending the message (fail, $d, i, \log_k n - r$) to all roots of these probes (by using the technique of splitting if necessary). Otherwise, $u$ forwards each probe to the next nodes of the according paths on level $\log_k n - r - 1$ (by using technique combining if necessary). After at most $O(\log_k n)$ rounds, each server $s$ that received a lookup request for some data item $d$ receives a fail-message from all of its probes that have been deactivated. Using this information, $s$ defines its lookup request to belong to the minimum level such that less than $c/2$ of its probes are deactivated at that level. Hence, at the end of the probing stage, each lookup request belongs to a level $\ell \in \{0, 1 \ldots, \log_k n\}$. All lookup requests that belong to level 0 can immediately be answered, while all lookup requests belonging to a level $\ell > 0$ will be handled in phase $\ell$ of the decoding stage.

It is easy to see that the probing stage of the Enhanced IRIS system satisfies the following property.

LEMMA 3.8. *The probing stage of the Enhanced IRIS system takes at most $O(\log_k n)$ communication rounds with at most $O(\log^2 n)$ congestion in every node at each round, w.h.p.*

Analogously to the Basic IRIS system (Lemma 2.13), one can show the following lemma.

LEMMA 3.9. *If the adversary can block at most $\varepsilon n$ servers for a sufficiently small constant $\varepsilon > 0$, then for every $\ell \in \{1, \ldots, \log_k n\}$, the number of data items with requests belonging to level $\ell$ is at most $\gamma n / k^\ell$, for a sufficiently small constant $\gamma$.*

*3.3.3. Decoding Stage.* Analogously to the Basic IRIS system, the decoding stage is divided into $\log_k n$ phases, where phase $\ell \in \{0, \ldots, \log_k n\}$ is dedicated to the decoding of the data items of lookup requests that belong to level $\ell$. In phase $\ell \in \{0, \ldots, \log_k n\}$, each server $s$ that received a lookup request for some data item $d$ that belongs to level $\ell$ performs or initiates the following tasks:

(1) Choose $c/2$ pairs $(d, i)$ that have not been deactivated in the probing stage for level $\ell$. For $i \in \{1 \ldots, c\}$, let $G_i(\ell)$ denote the graph from level $\ell$ that is (besides other servers) emulated by the server responsible for $h_i(d)$.
(2) For each of the $c/2$ previously chosen $(d, i)$ pairs, determine whether $G_i(\ell)$ could be decoded without nodes from $G_i(\ell)$ becoming congested, that is, receiving more than $O(ck)$ decode requests for different $(d, i)$ pairs. If any node from $G_i(\ell)$ would become congested when decoding $G_i(\ell)$, we denote $G_i(\ell)$ as *congested*. For a graph $G_i(\ell)$, let $BF(G_i(\ell))$ denote the $k$-ary butterfly that consists of the same servers as $G_i(\ell)$. Determining whether $G_i(\ell)$ is congested can be done just as in determining whether the $k$-ary subbutterfly $BF(G_i(\ell))$ is congested, as already described in the Basic IRIS system, by spreading a decode-message through $BF(G_i(\ell))$.
(3) If less than $c/4$ of the $G_i(\ell)$ graphs are congested, initiate the decoding of $c/4$ of the noncongested $G_i(\ell)$ graphs. Since the decoding depth of each $G_i(\ell)$ graph considered in level $\ell$ of the decoding stage is not exceeded, it is possible to completely decode $G_i(\ell)$ and retrieve the requested data pieces.

If at the end of phase $\ell$ server $s$ receives at least $c/4$ decoded pieces, then $s$ can recover the requested data item. Otherwise, $s$ denotes the request to belong to level $\ell + 1$ and handles it again in the next phase, phase $\ell + 1$.

It is easy to see that the probing stage of the Enhanced IRIS system satisfies the following property.

LEMMA 3.10. *The probing stage of the Enhanced IRIS system takes at most* $O(\log_k^3 n)$ *communication rounds with at most* $O(\log^3 n)$ *congestion in every node at each round, w.h.p.*

Analogously to the Basic IRIS system (Lemma 3.11), one can show the following lemma.

LEMMA 3.11. *For every phase* $r$ *of the decoding stage of the Enhanced IRIS system, the number of data items with requests belonging to level* $r$ *is at most* $\varphi n/k^r$ *with* $\varphi = \Theta(k)$.

Lemma 3.11 implies that all remaining data items can be decoded at the end.

Hence, the following theorem holds.

THEOREM 3.12. *If the adversary blocks less than* $\varepsilon n$ *many servers, with* $\varepsilon$ *being a sufficiently small constant, then using an overall redundancy of* $O(\log n)$*, any set of lookup requests (one per nonblocked server) is served correctly after at most* $O(\log^3 n)$ *communication rounds with a congestion of at most* $O(\log^3 n)$ *at every node in each round, w.h.p.*

Theorem 2.18 and Theorem 3.12 now imply our our main theorem, Theorem 1.1, stated in Section 1.3.

## 4. CONCLUSION AND FUTURE WORK

We presented the first scalable distributed information system that is provably robust against DoS attacks by a current insider that can shut down any $\varepsilon$-fraction of servers. An interesting challenge in this field of research is to enhance our system so that even write requests can be handled efficiently and correctly under a DoS attack (as was shown for a past insider in Baumgart et al. [2009]). Furthermore, notice that we did not try to optimize constants; from a practical perspective, it would certainly be interesting how small (large, respectively) they can be made.

## APPENDIX

## A. BUTTERFLY COMPLETION STAGE IN DETAIL

The goal of the butterfly completion stage is to make sure that for any DoS attack the servers are reorganized such that we again have a complete $k$-ary butterfly but only over the nonblocked servers.

The butterfly completion stage is divided into four phases, which are described in the following. For the rest of this section, we assume that the adversary blocks $\varepsilon n$ servers.

*Phase 1 (Build tree over nonblocked servers)*. In order to build a tree of depth $O(\log n/\log\log n)$, we first build a graph of degree $O(\log n)$ and diameter $O(\log n/\log\log n)$ (w.h.p.) consisting of all nonblocked servers. Afterward, this graph is transformed into a tree of depth $O(\log n/\log\log n)$ using the technique of a breadth-first search. Recall that $c = O(\log m)$ and $m \geq n$.

The graph is constructed as follows: First, each nonblocked server $s$ chooses $c$ other servers $s_1, \ldots, s_c$ uniformly at random. Afterward, each nonblocked server $s$ creates an edge to all nonblocked servers $s' \in \{s_1, \ldots, s_c\}$ and informs $s'$ about that such that $s'$ also creates an edge to $s$. With Chernoff bounds [Chernoff 1952], the following lemma is easy to prove.

LEMMA A.1. *After* $O(1)$ *rounds, the nonblocked servers have built a graph with degree* $O(\log n)$ *and diameter* $O(\log n/\log\log n)$*, w.h.p.*

In order to transform the constructed graph $G$ consisting only of nonblocked servers into a tree of depth $O(\log n/\log\log n)$, each nonblocked server initiates a breadth-first search (BFS). The idea is to let the servers create a tree that is rooted at the nonblocked server with minimum ID among all nonblocked servers. Since the nonblocked servers cannot determine this ID in advance in at most polylogarithmic time, each nonblocked server initiates a BFS. In the following, each nonblocked server $s$ holds three variables $\mathsf{minDist}(s)$, $\mathsf{minDistSource}(s)$, and $\mathsf{parent}(s)$ that are initialized with 0, ID(s), and *NIL*, respectively. $\mathsf{minDistSource}(s)$ will hold the minimum server ID from which $s$ has received a message so far. $\mathsf{minDist}(s)$ will hold the minimum distance to $\mathsf{minDistSource}(s)$ that server $s$ has stored so far. $\mathsf{parent}(s)$ will hold the server ID from which it has received the last message that initiated an update of $\mathsf{minDistSource}(s)$. In the first round, each nonblocked server $s$ sends the message $(\mathsf{id}(s), \mathsf{minDist}(s), \mathsf{minDistSource}(s))$ to each of its neighbors in $G$. Algorithm 1 describes the actions performed by each nonblocked server $s$ in each of the following rounds as soon as $s$ has received all messages from its neighbors.

**ALGORITHM 1:** GRAPHTOTREE

```
foreach message (id, minDist, minDistSource) received at the beginning of this round do
    if minDistSource < minDistSource(s) or minDistSource = minDistSource(s) and
    minDist < minDist(s) then
        ▷ update internal variables;
        minDistSource(s) ← minDistSource;
        minDist(s) ← minDist + 1;
        parent(s) ← id;
    end
end
foreach neighbor s' of s in G do
    send message (id(s), minDistSource(s), minDist(s)) to s'
end
```

Together with Lemma A.1 and for $\varepsilon < 2^{\log_k n}/24n < 1/2$ it follows:

LEMMA A.2. *After $O(\log n/\log\log n)$ rounds, the following holds, w.h.p.:*

(1) *The $\mathsf{parent}(s)$-values induce a tree $T$ of depth $O(\log n/\log\log n)$ over all $(1-\varepsilon)n$ nonblocked servers rooted at the nonblocked server with the minimum identifier.*
(2) *The degree of each node in $T$ is at most $O(\log n)$.*

*Phase 2 (Transform list to tree).* Next we show how to transform $T$ into a doubly linked list $L$ of $n$ nonblocked servers in which each nonblocked server is contained at most twice. First, using a bottom-up approach, each nonblocked server $s$ determines for each of its children $s'$ in $T$ the size $\mathsf{size}(s')$ of the subtree of $T$ rooted at $s'$ and the identifier of the rightmost server $\mathsf{rightmost}(s')$ in this subtree and reports it to its parent server. It is easy to show that this is possible in $O(\mathit{depth}(T))$ rounds. With this information, in a top-down approach, each nonblocked server then determines its position and its neighbors in a doubly linked list of $n$ nonblocked servers as follows: First, the root $r$ of $T$ (i.e., the server $r$ with $p(r) = \mathit{NULL}$) initiates a preorder walk of $T$ by performing Algorithm 2 with parameters $1, \mathit{NULL}$. Whenever a server receives a message, it also performs Algorithm 2. Clearly, after at most $\mathit{depth}(T)$ rounds, each nonblocked server knows its position and its left neighbor in $L$ (see Figure 10 for a visualization). In order to transform $L$ into a *doubly* linked list, each nonblocked server $s$ sends its ID to its left neighbor and sets $\mathsf{right}(s)$ to the ID it receives, or to *NULL* if it does not receive a message. In parallel to the transformation of $T$ into a doubly linked

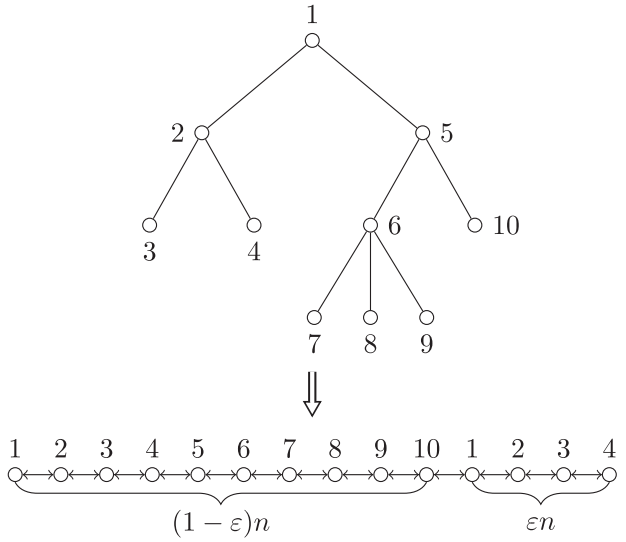


Fig. 10. Transformation of $T$ into a sorted list of $n$ nonblocked servers. The numbers next to the tree nodes denote the order of their appearance in the tree traversal.

list of the $(1-\varepsilon)n$ nonblocked servers, $r$ initiates an additional preorder traversal of $T$ by additionally performing Algorithm 2 with parameters $(1-\varepsilon)n+1$, $\mathsf{rightmost}(r)$. In contrast to the first traversal of $T$, the values $\mathsf{left}(s)$ and $\mathsf{pos}(s)$ in Algorithm 2 are now substituted by $\mathsf{left}_2(s)$ and $\mathsf{pos}_2(s)$. Also, we use the modification that as soon as a nonblocked server sets its $\mathsf{pos}_2$ value to $n$, the algorithm terminates. Then, analogously to the right values, each server sets its $\mathsf{right}_2$ value. By this additional tree traversal, the first $\varepsilon n$ servers of the traversal are appended to $L$. Notice that this preorder traversal of $T$ guarantees that the first $\varepsilon n$ servers visited form a connected subtree of $T$.

**ALGORITHM 2:** BUILDLISTFROMTREE$(x, l)$

$\mathsf{left}(s) \leftarrow l$, $\mathsf{pos}(s) \leftarrow x$;
**foreach** *child* $\mathsf{child}_i(s)$ *of* $s$, $i \in \{1, \ldots, c\}$ **do**
  **if** $i = 1$ **then**
    $\triangleright$ $\mathsf{child}_i(s)$ is the left most child of $s$;
    $\mathsf{left} \leftarrow \mathsf{child}_i(s)$;
  **else**
    $\mathsf{left} \leftarrow \mathsf{rightmost}(\mathsf{child}_{i-1}(s))$;
  **end**
  $\mathsf{pos} \leftarrow \mathsf{pos}(s) + 1 + \sum_{j=1}^{i-1} \mathsf{size}(\mathsf{child}_j(s))$;
  Send message $(\mathsf{pos}, \mathsf{left})$ to $\mathsf{child}_i(s)$;
**end**

The following lemma can easily be shown.

LEMMA A.3. *After* $2 \cdot depth(T)$ *rounds,* $T$ *is transformed into a doubly linked list* $L$ *of size* $n$ *over the* $(1-\varepsilon)n$ *nonblocked servers such that each nonblocked server is contained at most twice in* $L$.

*Phase 3 (Rearrange list).* The goal of this phase is to rearrange $L$ into a doubly linked list with the properties specified in Lemma A.4.

LEMMA A.4. *After $O(1)$ rounds, the following holds:*

(*i*) *Each nonblocked server $s_i$ is at position $i$ in $L$.*
(*ii*) *For each blocked server $s_j$, the server $s'$ with $\mathsf{pos}(s') = j$ or $\mathsf{pos}_2(s') = j$ is the unique representative of $s_j$.*

Initially, the *owner* of a position $j$ in $L$ is the server $s'$ with $\mathsf{pos}(s') = j$ or $\mathsf{pos}_2(s') = j$.

First, each nonblocked server $s'$ at position $j$ contacts server $s_j$. If $s_j$ is not blocked, then $s'$ considers $s_j$ as the new owner of $j$. $s'$ then asks its direct neighbors in $L$ for the owners of the positions $j-1$ and $j+1$ and forwards that information to $s_j$, so that $s_j$ can take over the position $j$ in $L$.

If $s_j$ is blocked, $s'$ remains to be the owner of $j$ and therefore becomes the representative of $s_j$.

*Phase 4 (Build extended k-ary butterfly).* In this phase, $L$ is transformed into a $k$-ary butterfly using an extended $k$-ary butterfly.

*Definition* A.5 (*Extended k-Ary Butterfly*). For any $d, k \in \mathbb{N}$, the *$d$-dimensional extended $k$-ary butterfly* $EBF(k,d)$ is a graph $(V,E)$ with $V = \bigcup_{\ell=0}^{d} V_\ell$ and $E = \bigcup_{\ell=1}^{d} E_\ell$, where

$$V_\ell = \{(\ell, i) \mid i \in \{1, \ldots, n\}\} \text{ and}$$
$$E_\ell = \{\{(\ell-1, i), (\ell, j)\} \in V_{\ell-1} \times V_\ell \mid \exists c \in \{1, \ldots, k-1\} : |i-j| = c \cdot k^{\ell-1}\}.$$

Furthermore, we define $G(\ell) = (V_{\ell-1} \cup V_\ell, E_\ell)$, $\ell \in \{1, \ldots, d\}$ and denote a node $(\ell, i)$ as a *level $\ell$ node*.

Recall that in the $k$-ary butterfly, the nodes from level $\ell-1$ and $\ell$, $\ell \in \{1, \ldots, \log_k n\}$ form $n/k$ disjoint complete $k$-bipartite subgraphs. In contrast to this, in the extended $k$-ary butterfly, the nodes from level $\ell-1$ and $\ell$ form $n-k$ complete $k$-bipartite subgraphs that contain the $n/k$ subgraphs from the standard $k$-ary butterfly.

The idea of this phase is to add for each nonblocked server at position $i$ in $L$ exactly $\log_k n + 1$ virtual nodes $(0,i), \ldots, (\log_k n, i)$ and to successively build the graphs $G(\ell)$, $\ell = 1, \ldots, \log_k n$, beginning with $G(1)$.

In $G(1)$, each node from level 0 needs to connect to all nodes on level 1 that are at distance at most $k-1$. For this, in the first round, each nonblocked server $s$ asks its two direct neighbors in $L$ for their neighbors in $L$, which introduces $s$ to its neighbors at distance 2 in $L$. In round $r \in \{2, \ldots, \lceil \log(k-1) \rceil\}$, each nonblocked server $s$ asks its two neighbors at distance $2^{r-1}$ for their neighbors (at distances $1, 2, \ldots, 2^{r-1}$) in $L$. This introduces $s$ to all servers at distance at most $2 \cdot 2^{r-1} = 2^r$ in $L$. Hence, after $\lceil \log(k-1) \rceil$ rounds, each nonblocked server knows all servers at distance at most $2^{\lceil \log(k-1) \rceil} = k-1$ in $L$.

The construction of $G(\ell)$, $\ell = 2, \ldots, \log_k n$ proceeds in $\lceil \log(k-1) \rceil + 1$ rounds and assumes that $G(\ell-1)$ has already been built. That is, each nonblocked server knows all servers at distance $c \cdot k^{\ell-2}$, $c \in \{1, \ldots, k-1\}$. In order to build $G(\ell)$, each nonblocked server needs to be introduced to all servers at distance $c \cdot k^{\ell-1}$, $c \in \{1, \ldots, k-1\}$. In the first round, each nonblocked server $s$ asks all servers at distance $(k-1)k^{\ell-2}$ (i.e., the servers farthest away) for their closest neighbors in $G(\ell-1)$ (i.e., their neighbors at distance $k^{\ell-2}$). By this, $s$ is introduced to all servers at distance $(k-1)k^{\ell-2} + k^{\ell-2} = k^{\ell-1}$. In round $r \in \{2, \ldots, \lceil \log(k-1) \rceil + 1\}$, each nonblocked server $s$ asks all servers at distance $2^{r-2} \cdot k^{\ell-1}$ for their neighbors (at distances $c \cdot k^{\ell-1}$, $c \in \{1, \ldots, k-1\}$) in $G(\ell)$. See Figure 11 for a visualization. This introduces each nonblocked server to the servers at

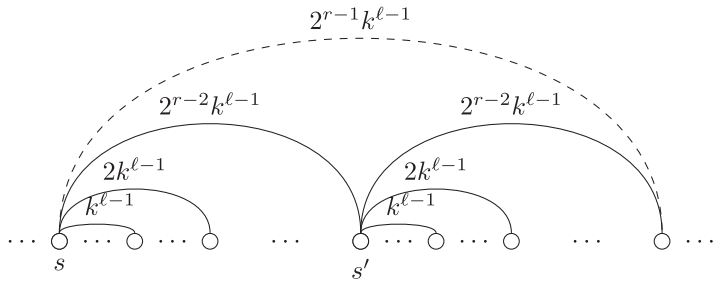


Fig. 11. Visualization of the construction of $G(\ell)$ in round $r$ in which the nonblocked server $s$ asks the server $s'$ at distance $2^{r-2}k^{\ell}$ for its neighbors. The dashed edge denotes a connection with maximum distance that $s$ builds in round $r$.

distance $(2^{r-2}+c)\cdot k^{\ell-1}$, $c \in \{1, 2, \ldots, 2^{r-2}\}$. Hence, after $\lceil \log(k-1) \rceil + 1$, each nonblocked server is introduced to all servers at distance $c \cdot k^{\ell-1}$, $c \in \{1, \ldots, (k-1)\}$.

LEMMA A.6. *In the fourth phase of the butterfly completion stage, a sorted list of $n$ nonblocked servers is correctly transformed into an extended $k$-ary butterfly in time $(2 + o(1)) \log n$ and at any time the congestion at every nonblocked server is at most $O(\log n)$.*

PROOF. By induction on $\ell$, it is easy to show that $G(\ell)$, $\ell \in \{1, \ldots, \log_k n\}$ is built correctly. Since the construction of each $G(\ell)$ takes $2(\lceil \log k \rceil + 1)$ rounds (each round described previously actually consists of two rounds), the extended $k$-ary butterfly is built after $2(\lceil \log k \rceil + 1) \log_k n$ rounds. By Lemma A.3, each nonblocked server is contained at most twice in $L$, implying that in each round each nonblocked server contacts (and is contacted by) at most four nonblocked servers and asks for their neighbors in $G(\ell - 1)$ and $G(\ell)$, respectively. Since each nonblocked server has at most $O(k)$ neighbors in $G(\ell - 1)$ and $G(\ell)$, the congestion of each nonblocked server is at most $O(k)$ in each round. □

Since the $d$-dimensional $k$-ary butterfly is a subgraph of the $d$-dimensional extended $k$-ary butterfly, Lemma 2.6 follows.

## REFERENCES

R. Ahlswede, N. Cai, S-y. R. Li, and R. W. Yeung. 2000. Network information flow. *IEEE Trans. Inf. Theory* 46, 4 (2000), 1204–1216.

B. Awerbuch and C. Scheideler. 2006. Towards a scalable and robust DHT. In *Proc. of SPAA*. 318–327.

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

M. Baumgart, C. Scheideler, and S. Schmid. 2009. A DoS-resilient information system for dynamic data management. In *Proc. of SPAA*. 300–309.

D. J. Bernstein. 2008. SYN cookies. *http://cr.yp.to/syncookies.html*.

A. Bhargava, K. Kothapalli, C. Riley, C. Scheideler, and M. Thober. 2004. Pagoda: A dynamic overlay network for routing, data management, and multicasting. In *Proc. of SPAA*. 170–179.

M. Blaum, J. Brady, J. Bruck, and J. Menon. 1994. EVENODD: An optimal scheme for tolerating double disk failures in RAID architectures. *SIGARCH Comput. Archit. News* 22, 2 (April 1994), 245–254.

A. Borodin and J. E. Hopcroft. 1982. Routing, merging and sorting on parallel models of computation. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC'82)*. ACM, New York, NY, 338–344.

V. R. Cadambe, C. Huang, and J. Li. 2011. Permutation code: Optimal exact-repair of a single failed node in MDS code based distributed storage systems. In *IEEE International Symposium on Information Theory*. 1225–1229.

H. Chernoff. 1952. A measure of asymptotic efficiency for tests of a hypothesis based on the sums of observations. *Ann. Math. Stat.* 23 (1952), 409–507.

P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. 2004. Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST'04)*. USENIX Association, Berkeley, CA, 1–14.

A. G. Dimakis, B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. 2010. Network coding for distributed storage systems. *IEEE Trans. Inf. Theory* 56, 9 (2010), 4539–4551.

D. Dittrich, J. Mirkovic, S. Dietrich, and P. Reiher. 2005. *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall PTR.

P. Druschel and A. Rowstron. 2001. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware*. 329–350.

B. Haeupler. 2011. Analyzing network coding gossip made easy. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC'11)*. ACM, New York, NY, 293–302. DOI:http://dx.doi.org/10.1145/1993636.1993676

N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. 2003. SkipNet: A scalable overlay network with practical locality properties. In *Proc. of USITS*. 9.

T. Ho, M. Medard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong. 2006. A random linear network coding approach to multicast. *IEEE Trans. Inf. Theory* 52, 10 (Oct. 2006), 4413–4430. DOI:http://dx.doi.org/10.1109/TIT.2006.881746

C. Huang and L. Xu. 2008. STAR: An efficient coding scheme for correcting triple storage node failures. *IEEE Trans. Comput.* 57, 7 (2008), 889–901. http://dblp.uni-trier.de/db/journals/tc/tc57.html#HuangX08

J. Ioannidis and S. M. Bellovin. 2002. Implementing pushback: Router-based defense against DDoS attacks. In *Proc. of NDSS*.

S. Kandula, D. Katabi, M. Jacob, and A. Berger. 2005. Botz-4-Sale: Surviving organized DDoS attacks that mimic flash crowds. In *Proc. of NSDI*. 287–300.

F. Kargl, J. Maier, and M. Weber. 2001. Protecting web servers from distributed denial of service attacks. In *Proc. of WWW*. 514–524.

A. D. Keromytis, V. Misra, and D. Rubenstein. 2002. SOS: Secure overlay services. In *Proc. of SIGCOMM*. 61–72.

S. Y. R. Li, R. W. Yeung, and N. Cai. 2003. Linear network coding. *IEEE Trans. Inf. Theory* 49, 2 (Feb. 2003), 371–381. DOI:http://dx.doi.org/10.1109/TIT.2002.807285

Mazu Networks Inc. 2008. http://mazunetworks.com.

J. Mirkovic and P. Reiher. 2004. A taxonomy of DDoS attacks and defense mechanisms. *Proc. of SIGCOMM* (2004).

W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Misra, and D. Rubenstein. 2003. Using graphic turing tests to counter automated ddos attacks against web servers. In *Proc. of CCS*. 8–19.

M. Naor and U. Wieder. 2003. Novel architectures for P2P applications: The continuous-discrete approach. In *Proc. of SPAA*. 50–59.

G. Oikonomou, J. Mirkovic, P. Reiher, and M. Robinson. 2006. A framework for collaborative DDoS defense. In *Proc. of ACSAC*. 33–42.

V. N. Padmanabhan and K. Sripanidkulchai. 2002. The case for cooperative networking. In *Proc. of IPTPS*. 178–190.

R. Pagh and F. F. Rodler. 2001. Cuckoo hashing. In *Proc of ESA*. 121–133.

D. S. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe. 2011. Repair optimal erasure codes through hadamard designs. *CoRR* abs/1106.1634 (2011). http://dblp.uni-trier.de/db/journals/corr/corr1106.html#abs-1106-1634.

D. Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics.

E. Ratliff. 2005. The zombie hunters. In *The New Yorker*.

S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. 2001. A scalable content-addressable network. In *Proc. of SIGCOMM*. 161–172.

I. Reed and G. Solomon. 1960. Polynomial codes over certain finite fields. *J. Soc. Industrial Appl. Math.* 8, 2 (1960), 300–304.

T. Stading, P. Maniatis, and M. Baker. 2002. Peer-to-peer caching schemes to address flash crowds. In *Proc. of IPTPS*. 203–213.

A. Stavrou, D. Rubenstein, and S. Sahu. 2002. A lightweight, robust P2P system to handle flash crowds. In *Proc. of ICNP*. 226–235.

I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. 2002a. Internet indirection infrastructure. In *Proc. of SIGCOMM*.

I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Kalakrishnan. 2002b. Chord: A scalable peer-to-peer lookup service for internet applications. In *Technical Report MIT*.

C. Suh and K. Ramchandran. 2010. *On the Existence of Optimal Exact-Repair MDS Codes for Distributed Storage*. Technical Report UCB/EECS-2010-46. EECS Department, University of California, Berkeley. Available at http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-46.html.

C. Suh and K. Ramchandran. 2011. Exact-repair MDS code construction using interference alignment. *IEEE Trans. Inf. Theory* 57, 3 (March 2011), 1425–1442. DOI:http://dx.doi.org/10.1109/TIT.2011.2105003

I. Tamo, Z. Wang, and J. Bruck. 2013. Zigzag codes: MDS array codes with optimal rebuilding. *IEEE Trans. Inf. Theory* 59, 3 (2013), 1597–1616.

L. G. Valiant. 1982. A scheme for fast parallel communication. *SIAM J. Comput.* 11, 2 (1982), 350–361.

M. Walfish, H. Balakrishnan, D. Karger, and S. Shenker. 2005. DoS: Fighting fire with fire. In *Workshop on Hot Topics in Networks (HotNets'05)*.

M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. 2006. DDoS defense by offense. *Proc. of SIGCOMM* 36, 4 (2006), 303–314.

Wikipedia. 2013. Denial-of-service attack. (2013). http://en.wikipedia.org/wiki/Denial-of-service_attack, accessed 12-February-2013.

L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner. 2006. Low-density MDS codes and factors of complete graphs. *IEEE Trans. Inf. Theory* 45, 6 (Sept. 2006), 1817–1826. DOI:http://dx.doi.org/10.1109/18.782102

L. Xu and J. Bruck. 1999. X-code: MDS array codes with optimal encoding. *IEEE Trans. Inf. Theory* 45 (1999), 272–276.

X. Yang, D. Wetherall, and T. Anderson. 2005. A DoS-limiting network architecture. In *Proc. of SIGCOMM*. 241–252.

Received November 2013; revised September 2014; accepted May 2015