# Near-Optimal Resilient Labeling Schemes

**Keren Censor-Hillel**
Department of Computer Science, Technion, Haifa, Israel

**Einav Huberman**
Department of Computer Science, Technion, Haifa, Israel

## Abstract

Labeling schemes are a prevalent paradigm in various computing settings. In such schemes, an oracle is given an input graph and produces a label for each of its nodes, enabling the labels to be used for various tasks. Fundamental examples in distributed settings include distance labeling schemes, proof labeling schemes, advice schemes, and more. This paper addresses the question of what happens in a labeling scheme if some labels are erased, e.g., due to communication loss with the oracle or hardware errors. We adapt the notion of resilient proof-labeling schemes of Fischer, Oshman, Shamir [OPODIS 2021] and consider resiliency in general labeling schemes. A resilient labeling scheme consists of two parts – a transformation of any given labeling to a new one, executed by the oracle, and a distributed algorithm in which the nodes can restore their original labels given the new ones, despite some label erasures.

Our contribution is a resilient labeling scheme that can handle $F$ such erasures. Given a labeling of $\ell$ bits per node, it produces new labels with multiplicative and additive overheads of $O(1)$ and $O(\log(F))$, respectively. The running time of the distributed reconstruction algorithm is $O(F + (\ell \cdot F)/\log n)$ in the Congest model.

This improves upon what can be deduced from the work of Bick, Kol, and Oshman [SODA 2022], for non-constant values of $F$. It is not hard to show that the running time of our distributed algorithm is optimal, making our construction near-optimal, up to the additive overhead in the label size.

**2012 ACM Subject Classification** Theory of computation $\rightarrow$ Distributed algorithms; Mathematics of computing $\rightarrow$ Graph algorithms

**Keywords and phrases** Labeling schemes, Erasures

**Digital Object Identifier** 10.4230/LIPIcs.OPODIS.2024.35

**Funding** This research is supported in part by the Israel Science Foundation (grant 529/23).
*Keren Censor-Hillel*: The research is supported in part by the Israel Science Foundation (grant 529/23).

**Acknowledgements** We thank Alkida Balliu and Dennis Olivetti for useful discussions about the state of the art for computing ruling sets.

## 1 Introduction

Assigning labels to nodes of a graph is a part of many fundamental computing paradigms. In distributed computing, cornerstone examples include distance and connectivity labeling schemes, as well as labeling schemes for nearest common ancestor [1, 2, 5, 9, 21–23, 27, 28, 30, 32, 38, 39, 41], proof labeling schemes and distributed proofs [8, 10, 11, 13, 15, 25, 26, 31, 33–35], advice schemes [17–20], and more, where the above are only samples of the extensive literature on these topics. The common theme in all of the above is that an oracle assigns labels to nodes in a graph, which are then used by the nodes in a distributed algorithm. Since distributed networks are prone to various types of failures, Fischer, Oshman and Shamir [16] posed the natural question of how to cope with nodes whose labels have been erased and defined the notion of *resilient proof labeling schemes*.

We will use the following definition, which applies to a labeling scheme for *any* purpose but captures the same essence. A resilient labeling scheme consists of two parts. The first part is an oracle transformation of an input labeling $\varphi : V \to \{0,1\}^\ell$ into an output labeling $\psi : V \to \{0,1\}^{\ell'}$, where $\ell' = A\ell + B$. The second part is a distributed Congest [1] algorithm, in which each node $v$ receives $\psi(v)$ as its input, except for at most $F$ nodes where $\psi(v)$ is erased, and each node must recover $\varphi(v)$. The parameters of interest are (i) the number $F$ of allowed label erasures, (ii) the multiplicative overhead $A$ and the additive overhead $B$ in the label sizes, and (iii) the round complexity of the distributed algorithm.

The aforementioned work of Fischer et al. [16] provides a resilient labeling scheme for the case of *uniform* labels, i.e., when the labels of all nodes are the same. For a general case, the work by Bick, Kol, and Oshman [7] implies a labeling scheme that is resilient to $F$ failures, with a constant multiplicative overhead in the label size, within $O(F^3 \log F + (\ell \cdot F^3)/\log n)$ rounds. In this paper, we ask how far we can reduce the overhead costs in the label size and the round complexity of the distributed algorithm.

▶ **Question.** *What are the costs of resilient labeling schemes?*

When $F$ is constant, the work of Bick et al. [7] achieves constant overheads in the label size and algorithm complexity. We show that for larger values of $F$, one can do better. The following states our contribution.

▶ **Theorem 1.** *There exists a resilient labeling scheme that tolerates $F$ label erasures, has $O(1)$ and $O(\log F)$ multiplicative and additive overheads to the size of the labels, respectively, and whose distributed* Congest *algorithm for recovering the original labels has a complexity of at most $O(F + (\ell \cdot F)/\log n)$ rounds.*

Notice that $\Omega(F)$ is a straightforward lower bound for the complexity of such a distributed algorithm. To see this, suppose the network is a path. If the labels of $F$ consecutive nodes along the path get erased, then for the middle node $v$ to obtain any information about its label, it must receive information from some node whose label is not erased. A message from such a node can only reach $v$ after $\Omega(F)$ rounds. Further, this example shows that $\Omega((\ell \cdot F)/\log n)$ is also a lower bound for the number of rounds, due to an information-theoretic argument: the nodes on the subpath of $P$ consisting of its middle $F/2$ nodes must obtain $O(\ell \cdot F)$ bits of information. This number of bits must pass over the two single edges that connect this subpath to the rest of $P$. Since each of the two edges can carry at most $O(\log n)$ bits, we get that $\Omega((\ell \cdot F)/\log n)$ rounds are required. This implies that our scheme is nearly optimal, leaving only the $O(\log F)$ additive overhead in the size of the label as an open question.

## 1.1 Technical Overview

To use a labeling scheme despite erasures, it must be strengthened by backing up information in case it is lost. Erasure-correcting codes are designed for this purpose, but they apply in a different setting – the entire codeword is available to the computing device, except for some bounded number of erasures. Yet, to handle label erasures, we need to encode the information and split the codeword among several nodes. Thus, when labels in a labeling scheme are erased, restoring them requires faulty nodes to obtain information from other nodes. This means that the oracle that assigns labels must encode its original labels into new, longer ones, and the nodes must communicate to restore their possibly erased labels.

[1] In the Congest model, $n$ nodes of a graph communicate in synchronous rounds by exchanging $O(\log n)$-bit messages along the edges of the graph.

**Partitioning.** By the above discussion on using error-correction codes, it is clear that if we partition the nodes of the graph into groups of size $\Theta(F)$, we can encode the labels within each such subgraph, as follows. By ensuring that each group forms a connected subgraph, the nodes can quickly collect all non-erased labels within a group in $O(F + (\ell \cdot F)/\log n)$ rounds, and thus decode and reconstruct all original labels.

When $F$ is large, simple methods can construct such a partition with negligible complexity overhead compared to $O(F)$. However, when $F$ is small, such overheads become significant. While this motivation is clear for the complexity of the distributed algorithm, it also applies to the overhead in the label size: much work is invested in providing labels of small size, e.g., the MST advice construction of Fraigniaud, Korman, and Lebhar [19].

The caveat is that if $F$ is small, there may not exist a distributed algorithm for such a partitioning that completes within $O(F)$ rounds. To see why, consider a simple reduction from 3-coloring a ring, which is known to have a lower bound of $\Omega(\log^* n)$ by Linial [36]. Given a partition into groups (paths) of size at least 3, we can color each group node-by-node in parallel, except for its two endpoints. This takes at most $O(F)$ rounds, as this is the size of each group. In a constant number of additional rounds, we can in parallel stitch all groups by coloring every pair of adjacent endpoints in a manner consistent with their other two neighbors, which are already colored. Thus, an $O(F)$-round algorithm for obtaining such a partition contradicts the lower bound for small values of $F$.

We stress that it is essential that the partition obtained by such a partitioning algorithm is exactly the same one used by the oracle for encoding. In particular, this rules out randomized partitioning algorithms or algorithms whose obtained partition depends on the identity of nodes with label erasures.

Our approach to address this issue is to have the oracle include partitioning information in the new labeling $\psi$. This way, even if $O(F)$ rounds are insufficient to *construct the partition from scratch*, the nodes can still *reconstruct the partition using the labels* within $O(F)$ rounds, despite any $F$ label erasures. A straightforward method would be to include all the node IDs of a group in each label. However, this would increase the size $\ell'$ of the new labeling $\psi$ to at least $O(F \log n)$, which for small values of $F$ is an overhead we aim to avoid. Thus, we need a way to hint at the partition without explicitly listing it.

**Ruling sets.** To this end, we employ ruling sets. An $(\alpha, \beta)$-ruling set is a set $S \subseteq V$ such that the distance between every two nodes in $S$ is at least $\alpha$, and every node is within distance at most $\beta$ to the set $S$. Given a ruling set with values of $\alpha, \beta$ that are $O(F)$, we prove that $O(F)$ rounds are sufficient for obtaining a good partition.

Interestingly, the properties of the partition we obtain are slightly weaker but still sufficient for our needs (and still rule out an $O(F)$-round algorithm for constructing them from scratch, by a similar reduction from 3-coloring a ring). The partition we arrive at is a partition into groups of $\Theta(F)$ nodes, where each group may not be connected but has low-congestion shortcuts of diameter $O(F)$ and congestion 2. This will allow us to collect all labels within a group in $O(F + (\ell \cdot F)/\log n)$ rounds.

More concretely, we aim to construct a $(2f + 2, 2f + 1)$-ruling set $S$, for some parameter $f \geq F$. For intuition, assume $f = F$. We create groups by constructing a BFS tree $T_v$ rooted at each node $v \in S$, capped at a distance of $2f + 1$ (with ties broken according to the smaller root ID). We then communicate only over tree edges to divide the nodes of $T_v$ into groups. We choose $\alpha = 2f + 2$ as a lower bound for the distance between two nodes in $S$, ensuring that each root $v$ has at least $f + 1$ nodes in its tree. These nodes are within distance $f + 1$ from $v$ and cannot be within that distance from any other node in the ruling set due to our choice of $\alpha = 2f + 2$. Additionally, to ensure that the round complexity does not exceed $O(f)$, we need to bound the diameter of each tree. Any $\beta = O(f)$ would suffice. However, as we will argue, computing the ruling set in the distributed algorithm of our resilient labeling scheme is too expensive. Thus, the oracle must provide information in its new labels to help nodes reconstruct the ruling set $S$. We choose $\beta = 2f + 1$ so that a node $u$ at distance $2f + 2$ from a vertex $v \in S$ can determine it must be in $S$ even if its label is erased.

As promised, we note that due to the lower bound of Balliu, Brandt, Kuhn, and Olivetti [4], no distributed algorithm can compute even the simpler task of a $(2, \beta)$-ruling set in fewer than $\Omega(\log n/\beta \log\log n)$ rounds. Thus, in an algorithm that takes $O(F)$ rounds to construct a ruling set with $\beta = \Theta(F)$, $F$ must be at least $\Omega((\log n/\log\log n)^{1/2})$. This leaves the range $F = O((\log n/\log\log n)^{1/2})$ as an intriguing area. We emphasize that such label sizes are addressed by substantial research on labeling schemes, e.g.: Korman, Kutten, and Peleg [35] assert that for every value of $F$, there exists a proof-labeling scheme with this label length. Baruch, Fraigniaud, and Patt-Shamir [6] prove that a proof-labeling scheme of a given length can be converted into a randomized version with logarithmic length. Patt-Shamir and Perry [40] provide additional proof-labeling schemes with small non-constant labels, e.g., they prove the existence of a proof-labeling scheme for the $k$-maximum flow problem with a label size of $O(\log k)$.[2]

The existence of this lower bound could be overcome by adding a single bit to each node's new label to indicate whether the node is in the ruling set. However, another significant obstacle is that the nodes need to reconstruct the ruling set given $F$ erasures. This turns out to be a major issue, as follows.

Suppose we have a path $(w_0, \cdots, w_{f+1}, \cdots, w_{2f+1})$, with two nodes $u, v$ connected to $w_{2f+1}$, and the rest of the graph is connected through $w_0$. Suppose a $(2f + 2, 2f + 1)$-ruling set $S$ contains $w_0$ and one of $u$ or $v$, and that $ID(v) < ID(w_0) < ID(u)$. If the labels of $u$ and $v$ both get erased, then the node $w_{f+1}$ cannot determine whether it should be part of the tree $T_{w_0}$ or not. Its distance from $v, u$ and $w_0$ is $f + 1$, but the ID comparisons for $w_0, u$ and $w_0, v$ yield different results: if $u \in S$, then $w_{f+1}$ should join $T_{w_0}$, but if $v \in S$, then $w_{f+1}$ should join $T_v$. Notice that all nodes with non-erased labels have the same distance to $u$ as to $v$, making it impossible to distinguish between the two possibilities.

This seems to bring us to a dead-end, as the simple workaround of providing each node with the identity of its closest node in the ruling set $S$ is too costly. It would require adding $\Theta(\log n)$ bits to the new labeling $\psi$, which we wish to avoid.

**Greedy ruling sets.** Our main technical contribution overcomes this challenge by demonstrating that a greedy ruling set *does* allow the nodes to recover it given $F$ label erasures. Such a ruling set is constructed by iterating over the nodes in increasing order of IDs. When a node $v$ is reached, it is added to the ruling set, and all nodes within distance $2f + 1$ from $v$ are excluded from the ruling set and ignored in later iterations. Intuitively, in our above example, this ensures that $S$ contains $v$ and not $u$. Thus, when their labels are erased, the nodes can deduce that $v$ should be in $S$ because it has a smaller ID.

While this approach solves the previous example, it is still insufficient. Suppose there is a sequence of nodes $v_1, \ldots, v_f$ with decreasing IDs whose labels get erased and are not *covered* by any other node in $S$ (a node $v$ is covered by a node in $S$ if their distance is at

[2] To our knowledge, the fastest deterministic construction in Congest takes $\tilde{O}(\log^3 n)$ rounds, due to Faour, Ghaffari, Grunau, Kuhn, and Václav [12]. Therefore, the actual complexity may be even larger, which would further widen the range of $F$ in which our construction outperforms other methods.

most $2f + 1$). Assume that the distance between every two consecutive nodes in the sequence is $\Theta(f)$. This means that each node $v_i$ knows it needs to be in the ruling set if and only if $v_{i+1}$ is not, essentially unveiling the decisions of the greedy algorithm. However, this implies that $v_1$ needs to receive information about $v_f$, which may be at distance of $\Theta(f^2)$, implying more rounds than our target running time.

To cope with the latter example, the oracle will provide each node with its distance from $S$. This incurs an additive $O(\log F)$ bits in the label size. This information is useful for deciding, for some of the nodes with erased labels, whether they are in $S$ or not and, in particular, it breaks the long chain of the latter example. Yet, notice that in the former example, this information does not help us, since all nodes apart from $u$ and $v$ have the same distance to $u$ as to $v$. The reason this helps in one example but not in the other is as follows. For any two nodes $u$ and $v$ with erased labels that need to decide which of them is in $S$, as long as we have at least $f - 1$ additional nodes with a *different* distance from $u$ and $v$, we can easily choose between $u$ and $v$, because in total there can be at most $f$ label erasures, so one of these $f - 1$ reveals which node needs to be chosen.

Thus, to reconstruct a greedy $(2f + 2, 2f + 1)$-ruling set $S$ using the above information in the non-erased labels, the nodes do the following. First, a node that sees a non-erased label in $S$ within distance $2f + 1$ from it can deduce that it is not in $S$. Second, a node for which all labels within distance $2f + 1$ are not erased and are not in $S$ can deduce that it is in $S$. For a remaining node $u$ with an erased label, the distance from $S$ of the non-erased labels in its $2f + 1$ neighborhood helps in deciding whether $u$ is in $S$ or not: if it has a node within this distance whose distance from $u$ differs from its distance from $S$, then $u$ is not in $S$. At this point in the reconstruction algorithm, the only remaining case is a node $v$ which is not known to be covered by nodes in $S$ at this stage and has a non-empty set of nodes $X(v)$ within distance $f$ from it whose labels are erased. We show that such nodes cannot create distant node sequences as in the latter example, which allows $v$ to simply check if it has the smallest ID in $X(v)$, indicating whether it should be in $S$.

Formally, we refer to such nodes as *alternative nodes*, defined as follows. Let $dist(v, u)$ be the distance between nodes $v$ and $u$, and let $B_t(v)$ be the set of nodes at distance at most $t$ from $v$. For a node $v$, a node $u \neq v$ is called an *alternative node* if $dist(u, v) \leq f$ and there are at most $f - 2$ nodes $q \in B_{f+1}(v) \cup B_{f+1}(u)$ with $dist(q, v) \neq dist(q, u)$. We emphasize that proving that the only remaining case of alternative nodes is technical and non-trivial.

Given any $f \geq F$, we prove that providing each node with a single-bit indication of whether it is in the greedy $(2f+2, 2f+1)$-ruling set $S$ and an additional $O(\log f)$ bits holding its distance from $S$ is sufficient for the nodes to reconstruct $S$ in $O(f)$ rounds, despite $F$ erasures. Due to the codes we choose, we will need $f$ to be slightly larger than $F$. However, it is sufficient to choose $f = \Theta(F)$, so to reconstruct the greedy ruling set, the label size needs an additive overhead of $O(\log F)$ and the number of rounds remains $O(F)$.

**Wrap-up.** We can now describe our resilient labeling scheme in full.

In Section 2, we use a greedy ruling set with $f = \Theta(F)$ to construct groups and apply an error-correcting code to the labels of their nodes. This provides the entire algorithm of the oracle for transforming an $\ell$-labeling $\varphi$ into an $\ell'$-labeling $\psi$, such that $\ell'$ has a constant multiplicative overhead for coding and an additive overhead of $O(\log F)$ for restoring the greedy $(2f + 2, 2f + 1)$-ruling set.

In Section 3, we provide a distributed algorithm for restoring the greedy $(2f + 2, 2f + 1)$-ruling set given the labeling $\psi$ despite $F$ label erasures, within $O(F)$ rounds. In Section 4, the nodes partition themselves into the same groups as the oracle, within $O(F)$ rounds.

Finally, Section 5 presents our complete resilient labeling scheme, which requires an additional number of $O(F + (\ell \cdot F)/\log n)$ rounds for communicating the non-erased labels within each group and decode to obtain the original labeling $\varphi$.

## 1.2 Preliminaries

Throughout the paper, let $G = (V, E)$ be the network graph, which without loss of generality is considered to be connected. Let $n$ denote the number of nodes in $G$. After labels are assigned to nodes by the oracle, there is a parameter $F$ of the number of erasures of labels that may occur. For each node $v \in V$, we denote by $ID(v)$ the unique identifier of $v$.

We denote by $dist(v, u)$ the distance between two nodes $v$ and $u$, and by $P_{v,u}$ some shortest path between them. The ball of radius $t$ centered at a node $v$ consists of all nodes within distance $t$ from $v$ and is denoted by $B_t(v) = \{u \in V \mid dist(u, v) \leq t\}$.

Throughout the paper, we use $a \circ b$ to refer to the concatenation of two strings $a$ and $b$.

## 1.3 Related Work

As mentioned earlier, the work of Bick, Kol, and Oshman [7] implicitly suggests a resilient labeling scheme with an $O(1)$ multiplicative overhead to the label size in $O(F^3 \log F + (\ell \cdot F^3)/\log n)$ rounds. This is achieved by constructing a so-called $(k, d, c, h)$-helper assignment, which assigns $k$ helper nodes to each node $v$ in the graph, with paths of length at most $d$ from each $v$ to all of its helper nodes, such that each edge belongs to at most $c$ paths, and a node is assigned as a helper node to at most $h$ nodes $v$. The paper computes a $(k, O(k), O(k^2), O(k))$-helper assignment in $O(k \log k)$ rounds, using messages of $O(k^2 \log n)$ bits. Plugging in $k = \Theta(F)$ gives the aforementioned resilient labeling scheme. For $F = \omega(1)$, the parameters we obtain improve upon the above.

Ostrovsky, Perry, and Rosenbaum [37] introduced $t$-proof labeling schemes, which generalize the classic definition by allowing $t$ rounds for verification, and study the tradeoff between $t$ and the label size. This line of work was followed up by Feuilloley, Fraigniaud, Horvonen, Paz, and Perry [15], and Fischer, Oshman, and Shamir [16]. All of these papers provide constructions of groups of nodes that allow sharing label information to reduce the label size in this context. However, our construction is necessary for handling: (i) general graphs, (ii) in the Congest model, (iii) with arbitrary labels, and (iv) while achieving our goal complexity. It is likely that our algorithm can be applied to this task as well. Error-correcting codes have already been embedded in the aforementioned approaches of Bick et al. [7] and Fischer et al. [16], and we do not consider their usage a novelty of our result.

Note that Feuilloley and Fraigniaud [14] design error-sensitive proof-labeling schemes, whose name may seem to hint at a similar task, but are unrelated to our work: these are proof-labeling schemes where a proof that is far from being correct needs to be detected by many nodes.

## 2 Assigning Labels

To allow the nodes in the distributed algorithm to decode their original labels despite $F$ label erasures, we employ an error-correcting code in our resilient labeling scheme. Known codes can correct a certain number of errors at the cost of only a constant overhead in the number of bits. Therefore, we can split the nodes into groups of size $\Theta(F)$ and encode the labels within each group. We include in the new labels information about the group partitioning to allow the distributed algorithm to quickly reconstruct the same groups. This information comes in the form of a ruling set, on which we base the group partitioning. To ensure that this reconstruction can be done despite label erasures, we ensure that the oracle uses a greedy ruling set. We present the oracle's algorithm and then prove its guarantees in Theorem 3.

**Overview.** In Algorithm 1, the oracle first constructs a greedy $(2f+2, 2f+1)$-ruling set $S$. For every node $v$, let $b(v)$ be a single bit indicating whether $v \in S$, and let $distS(v)$ be an $O(\log f)$-bit value which is the distance between $v$ and $S$.

The oracle then partitions the nodes into groups of size $\Theta(f)$. To later allow the distributed algorithm to reconstruct the same groups, the oracle uses the same algorithm the nodes will later use, i.e., Algorithm 3. However, for the description in this section, any partitioning with the same guarantees would give the desired result in Theorem 3 below.

Finally, the oracle applies an error-correcting code to each index of the labels of all nodes in each group. The new label it assigns to each node $v$ is a concatenation of its $b(v)$ and $distS(v)$ values, along with the encoded label.

**Algorithm 1** Oracle's algorithm for assigning new labels.

```
Input: A graph $G$, a parameter $F$, and an $\ell$-labeling $\varphi : V \to \{0,1\}^\ell$.
Output: An $\ell'$-labeling $\psi : V \to \{0,1\}^{\ell'}$.
1 Construct a greedy $(2f+2, 2f+1)$-ruling set $S$. For every node $v$, denote by
    $distS(v)$ its distance to $S$ .
2 Simulate an execution of Algorithm 3 with $b(v) = 1$ for every node $v \in S$ and
    $b(v) = 0$ for every node $v \notin S$. At the end of the algorithm, every node $v$ is
    associated with a group identifier $Group(v)$ .
3 For every group $Group$ obtained in the previous step, let $(v_1, \dots, v_s)$ be the IDs of its
    nodes in increasing order. For every $1 \le j \le \ell$, concatenate the $j$-th bit of the labels
    $\varphi(v)$ of the nodes according to their increasing to obtain a string $\mathtt{r}^j$, and encode this
    string into a codeword $\mathtt{w}^j$ according to the code $\mathtt{C}$. Split the bits of $\mathtt{w}^j$ into $s$
    consecutive blocks $(\mathtt{w}^j_1, \dots, \mathtt{w}^j_s)$ of sizes $\lceil |\mathtt{w}^j|/s \rceil$ or $\lfloor |\mathtt{w}^j|/s \rfloor$. For every $1 \le i \le s$, let
    $\mathtt{w}_i = \mathtt{w}^1_i, \dots, \mathtt{w}^\ell_i$ and set $\psi(v_i) = b(v_i) \circ distS(v_i) \circ \mathtt{w}_i$ .
```

**Choice of code.** Different codes $\mathtt{C}$ result in different overheads of $\ell'$ compared to $\ell$. A simple repetition code takes the string `info` of length $s$ and repeats it for $s$ blocks to create the resulting codeword $\mathtt{w} = \mathtt{C}(\mathtt{info})$, which is trivially decodable given any $s-1$ blocks erasures. Splitting $\mathtt{w}$ back to $s$ pieces implies that $\mathtt{w}_i$ equals `info` for every node $v_i$ in the group. Taking $f = F$, this results in $\ell' = 1 + O(\log f) + s \cdot \ell = O(\log F) + s \cdot \ell$. Since the group size $s$ is always $\Theta(f) = \Theta(F)$, we get a multiplicative overhead of $O(F)$ and an additive overhead of $O(\log F)$. To reduce the multiplicative overhead to $O(1)$, we use a code with better parameters. Formally, a binary code $\mathtt{C}$ maps strings of $k$ bits into strings of $N$ bits called *codewords*. The ratio $\rho = N/k$ is the code's *rate*. The *relative distance*, $\delta$, ensures the Hamming distance (number of indices where the strings differ) between any two codewords is at most $\delta N$. Such a code can correct $\delta N - 1$ erasures.

▶ **Lemma 2** (Justesen Codes [29], phrasing adopted from [3]). *For any given rate $\rho < \frac{1}{2}$ and for any $M > 0$, there exists a binary code $\mathtt{C}_M : \{0,1\}^{k_M = \rho_M \cdot N_M} \to \{0,1\}^{N_M}$ with $N_M = 2M(2M-1)$ such that the code $\mathtt{C}_M$ has rate $\rho_M \ge \rho$ and relative distance $\delta_M > (1-2\rho)H^{-1}(1/2)$, where $H(x) = x\log(1/x) + (1-x)\log(1/(1-x))$ is the binary entropy function. Encoding and decoding can be done in polynomial time.*

Given the bound $F$ on the number of label erasures, we need to choose a value of $f$ to work with. Then, given a group of size $s$ such that $f+1 \leq s \leq 3f+1$, we need to choose values for $\rho$ and $M$ that determine the code $\mathtt{C}_M$. We set these parameters as follows. We let $f = 80F$, and for every such $s$ we choose $\rho = 1/4$. To choose $M$, notice that Lemma 2 gives that for any $M > 0$, the number of bits we can encode is $k_M = \rho_M N_M$. We choose the smallest $M$ such that $k_M$ is at least $s$.

The next theorem proves that the overhead for the number of bits held by each node per coded bit is constant and that, given the coded blocks,it is possible to decode the original string of bits if up to $F$ blocks are erased. The proof is presented in Appendix A.

▶ **Theorem 3.** *Let $f = 80F$ and let $\rho = 1/4$. Let $(v_1, \ldots, v_s)$ be the sequence of nodes in a group of size $s$, such that $f+1 \leq s \leq 3f+1$, ordered by increasing IDs. Let $\mathtt{r}$ be a string obtained by concatenating a single bit associated with each $v_i$ according to their order, and let $\mathtt{w}$ be a codeword obtained in Step 3 of Algorithm 1, by using a code $\mathtt{C}_M$ from Lemma 2, with a value of $M$ that is the smallest such that $k_M \geq s$. Then,*

1. *it holds that $\lceil |\mathtt{w}|/s \rceil \leq 80$, and*
2. *the string $\mathtt{r}$ is decodable given $\mathtt{w} = (\mathtt{w}_1, \ldots, \mathtt{w}_s)$ with up to $F$ erasures of blocks $\mathtt{w}_i$.*

For later use by the distributed algorithm in our resilient labeling scheme, we denote by $decode(v, \{(u, \mathtt{w}(u)) \mid u \in Group(v)\})$ the function that takes as input a node $v$ and the values $\mathtt{w}(u)$ for all the nodes $u$ in $Group(v)$ (along with their IDs so that they can be ordered correctly) and produces the bit of $v$ in the string $\mathtt{r}$ which was encoded into $\mathtt{w}$, by Theorem 3.

## 3 Restoring a Greedy Ruling Set Despite Erasures

In this section we present a distributed algorithm that restores a greedy ruling set despite label erasures. Nodes are *faulty* if their label are erased, and *non-faulty* otherwise. Each node $v$ aims to restore its value $b(v)$ so that the set $\{v \mid b(v) = 1\}$ is exactly the given ruling set $S$. Once a faulty node $v$ sets its value $b(v)$, it is no longer considered faulty for the remainder of the algorithm. The number of label erasures our algorithm can tolerate is denoted by $f$, and in a setting with at most $F$ label erasures, the algorithm can be executed correctly for any value $f \geq F$.

▶ **Theorem 4.** *Let $G = (V, E)$ be a graph and let $S$ be a greedy $(2f+2, 2f+1)$-ruling set $S$. Suppose that, apart from up to at most $f$ faulty nodes, each node $v$ is provided with a label $\zeta(v) = b(v) \circ distS(v)$, where $b(v) = 1$ if $v \in S$ and $b(v) = 0$ otherwise, and $distS(v) = \min_{u \in S}\{dist(v, u)\}$. Then, there exists a deterministic* Congest *algorithm at the end of which every node $v$ restores its value $b(v)$. The algorithm runs in $O(f)$ rounds.*

Before presenting and analyzing our algorithm for Theorem 4, we recall the definition of alternative nodes.

▶ **Definition 5** (Alternative Nodes)**.** *For a node $v$, a node $u \neq v$ is called an* alternative node*, if $dist(u, v) \leq f$ and there are at most $f-2$ nodes $q \in B_{f+1}(v) \cup B_{f+1}(u)$ with $dist(q, v) \neq dist(q, u)$. Denote by $M(v)$ the set of alternative nodes for $v$.*

A key tool in our analysis is the proof that a greedy ruling set $S$ excludes nodes $v$ with alternative nodes $u \notin S$ having smaller ID. In Appendix A, we prove the following.

▶ **Lemma 6.** *Let $S$ be a greedy $(2f+2, 2f+1)$-ruling set. Then, there exists no node $v \in S$ for which there is an alternative node $u \notin S$ with $ID(u) < ID(v)$.*

We are now ready to prove Theorem 4. We begin by presenting the algorithm.

**Overview.** In Algorithm 2, nodes restore a greedy ruling set $S$ as follows. Before the algorithm starts, each non-faulty node $w$ sets its value $b(w)$ to the first bit of its label. At the beginning of the algorithm, non-faulty nodes in $S$ broadcast a bit "1" for $2f+1$ rounds. Here, every node that receives a bit "1" during these rounds forwards it to all of its neighbors. Faulty nodes $u$ that receive a bit during these rounds sets $b(u)=0$, as this indicates that $u \in B_{2f+1}(v)$ for some such $v \in S$ and thus $u \notin S$.

Next, each faulty node propagates its ID through a distance of $2f+1$. Faulty nodes $u$ that do not receive any other ID set $b(u)=1$, as this means no other faulty nodes are in $B_{2f+1}(u)$ and thus $u \in S$.

Now, every remaining faulty node $u$ propagate a message of the form $ID(u) \circ dist$ through a distance of $f+1$, where $dist$ starts at 0 and is increased by 1 by each node along the way, corresponding to the length of the path. For each node $w$, let $dist_u(w)$ be the smallest $dist$ value received for $u$, representing the actual distance between $w$ and $u$.

Each node $w \in B_{f+1}(u)$ receives the message and checks if it is possible that $u \in S$ by verifying whether $distS(w) = dist_u(w)$. If this equality does not hold, $w$ propagates a message with $ID(u)$ through a distance of $f+1$. Since $dist(w,u) \leq f+1$, if $u \in S$, then $distS(w) = dist_u(w)$. If $w$ propagates $ID(u)$, then this is not the case, so when $u$ receives its own ID, it can conclude that it is not in $S$ and sets $b(u)=0$.

Now, we are left with faulty nodes $v \in S$ and faulty nodes $u \notin S$. We will prove that $u$ is an alternative node for a faulty node $v \in S$. Thus, every faulty node $v \in S$ must have the lowest ID among faulty nodes in $B_{f+1}(v)$ and can deduce that it is in $S$ and set $b(v)=1$. Similarly, every faulty node $u \notin S$ must have a faulty node $v \in S$ within $B_{f+1}(u)$ such that $ID(v) < ID(u)$, allowing $u$ to deduce it is not in $S$ and set $b(u)=0$.

Following is the formal algorithm and proof.

**Proof of Theorem 4.** To prove that the algorithm restores the set $S$, we prove that every node $w \in V$ holds $b(w)$ at the end of Algorithm 2, such that $\{w \mid b(w)=1\} = S$. By Step 1, this holds for all non-faulty nodes, so it remains to prove it for faulty nodes. In what follows, we go over each step in the algorithm where nodes may set their output (Steps 3,5,8, and 10), and we show that any node which sets its output during that step does so correctly.

**Step 3.** In Step 2, every non-faulty node $v \in S$ broadcasts a bit "1" for $2f+1$ rounds. By the definition of a $(2f+2, 2f+1)$-ruling set, every two nodes $v', v'' \in S$ satisfy that $dist(v', v'') \geq 2f+2$. Thus, if a faulty node $u$ receives a bit "1" from $v \in S$ during these $2f+1$ rounds, it implies that $u \in B_{f+1}(v)$. Hence, $u \notin S$ since $v \in S$. So, for any node $u$ that sets $b(u)=0$ in Step 3, its output is correct.

**Step 5.** Afterwards, in Step 4, each faulty node propagates a message with its ID through a distance of $2f+1$. Every faulty node $u$ that does not receive any other ID during these rounds and did not receive a "1" during Step 2 sets $b(u)=1$, because this implies that there is no other faulty node in $B_{2f+1}(u)$ and hence it must hold that $u \in S$. So, it is correct that $u$ sets $b(u)=1$ in Step 5.

**Step 8.** Then, in Step 6, every node $u$ that is still faulty propagates a message of the form $ID(u) \circ dist$ through a distance of $f+1$, where $dist$ starts at 0 and is increased by 1 before the message is propagated further. This corresponds to the length of the path that this message traverses. Every node $w \in B_{f+1}(u)$ receives the above message and checks if it is possible that $u \in S$ by verifying whether $distS(w) = dist_u(w)$. We denote by $dist(u)$ the value of $dist$ from the message $ID(u) \circ dist$ that $w$ receives. If equality does not hold, then $w$ propagates a message with $ID(u)$ through a distance of $f+1$. Since $dist(u,w) \leq f+1$, if $u \in S$, then for every other node $v' \in S$ such that $dist(w,v') \leq f+1$, it holds that

**Algorithm 2** Restoring a greedy $(2f+2, 2f+1)$-ruling set $S$ despite $f$ label erasures.

```
Input: A graph $G$ with a greedy ruling set $S$, where each node $w$ is provided with
       $\zeta(w) = b(w) \circ distS(w)$, except at most $f$ faulty nodes which are provided
       with $\zeta(w) = empty$.
Output: Every node $w$ outputs $b(w)$ such that $\{w \mid b(w) = 1\} = S$.
1  Every non-faulty node $w \in V$ sets $b(w)$ to the first bit in $\zeta(w)$ .
2  Every node $v \in S$ with $\zeta(v) \neq empty$ broadcasts a bit "1" for $2f+1$ rounds .
3  Every faulty node $u$ that receives a bit "1" during Step 2 sets $b(u) = 0$ .
4  Every faulty node $u$ propagates a message with $ID(u)$ through a distance of $2f+1$.
     Nodes that have already broadcast $ID(u)$ do not repeat the broadcast .
5  Every faulty node $u$ which does not receive any other ID during Step 4 sets $b(u) = 1$ .
6  Every faulty node $u$ propagates a message of the form $ID(u) \circ dist$ through a
     distance of $f+1$, where $dist$ starts at 0 and is increased by 1 by any node along the
     way. A node does not propagate a message with some ID if it has already done so
     with a smaller value of $dist$. For each node $w$, let $dist_u(w)$ be the smallest $dist$
     value it receives for node $u$ .
7  Every node $w \in B_{f+1}(u)$ with $\zeta(w) \neq empty$ receives a message during Step 6 and
     checks if $distS(w) = dist_u(w)$. If the equality does not hold, then $w$ propagates a
     message with $ID(u)$ through a distance of $f+1$. Nodes that have already broadcast
     $ID(u)$ do not repeat the broadcast .
8  Every faulty node $u$ that receives $ID(u)$ during Step 7 sets $b(u) = 0$.
9  Every faulty node $u$ propagates a message with $ID(u)$ through a distance of $f$.
10 Let $X(u)$ be the set of nodes whose IDs are received by $u$ during Step 9. If
     $ID(u) = \min_{u' \in X(u)}\{ID(u')\}$, then $u$ sets $b(u) = 1$. Otherwise, it sets $b(u) = 0$ .
```

$2f+2 \leq dist(u,v') \leq dist(u,w) + dist(w,v') \leq f+1+dist(w,v')$. Thus, $dist(w,v') \geq f+1$, and since $distS(w) = \min_{v \in S}\{dist(q,v)\}$ we have $distS(w) = dist_u(w)$. But if $w$ propagates $ID(u)$, then this is not the case, so $u \notin S$. Thus, it is correct that $u$ sets $b(u) = 0$ in Step 8.

**Step 10.** First we prove that every node $u$ that is still faulty at the beginning of this step and every node $v \in X(u)$ are alternative nodes. By definition of alternative nodes, we need to prove that $dist(u,v) \leq f$ and that $v$ and $u$ have at most $f-2$ nodes $w \in B_{f+1}(v) \cup B_{f+1}(u)$ with $dist(u,w) \neq dist(v,w)$.

First, since $v \in X(u)$, it must be that $dist(u,v) \leq f$. Second, we prove that $v$ and $u$ have at most $f-2$ nodes $t \in B_{f+1}(v) \cup B_{f+1}(u)$ with $dist(u,t) \neq dist(v,t)$. Assume, towards a contradiction, that there are at least $f-1$ nodes $t \in B_{f+1}(v) \cup B_{f+1}(u)$ with $dist(v,t) \neq dist(u,t)$. Since $u$ is still faulty in Step 9, there are at most $f-2$ nodes $q \neq u, v$ in $B_{f+1}(u)$ with $dist(u,q) \neq dist(v,q)$, because otherwise $u$ would receive a message with $ID(u)$ from such a non-faulty node during Step 7 and would set $b(u) = 0$ in Step 8. Therefore, there exists a node $s \in B_{f+1}(v) \setminus B_{f+1}(u)$ that is still faulty in Step 9. We consider two cases, obtaining a contradiction in both, and conclude that the assumption is incorrect. Thus, $v$ and $u$ have at most $f-2$ nodes $t \in B_{f+1}(v) \cup B_{f+1}(u)$ with $dist(u,t) \neq dist(v,t)$.

**Case 1: No node $r \neq v$ exists on $P_{v,s} \cap P_{v,u}$.** An illustration of this case is presented in Figure 1. In this case, every node $t \neq u, v$ on $P_{v,u}$ satisfies that $dist(u,t) \neq dist(v,t)$, except at most one node $a$ (such a node $a$ exists if $dist(u,v)$ is even and $a$ is exactly in the middle of $P_{v,u}$). Additionally, every node $t$ on $P_{v,s}$ satisfies that $dist(t,v) \neq dist(t,u)$, as otherwise $dist(u,s) \leq dist(u,t) + dist(t,s) = dist(v,t) + dist(t,s) = dist(v,s) \leq f+1$,

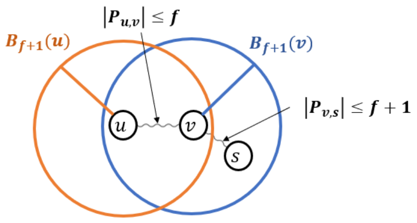

**Figure 1** Illustration for Case 1. No node $r \neq v$ exists on $P_{v,s} \cap P_{v,u}$. Therefore, $P_{v,u}$ and $P_{v,s}$ are disjoint except at the node $v$. The orange circle represents $B_{f+1}(u)$ and the blue circle represents $B_{f+1}(v)$. We assume that $dist(u, v) \leq f$, so $|P_{u,v}| \leq f$. Additionally, $s \in B_{f+1}(v) \setminus B_{f+1}(u)$.

which contradicts the assumption that $dist(u, s) \geq f + 2$. Denote by $P'$ the path $P_{u,v} \circ P_{v,s}$ at a distance of at most $f + 1$ from $u$. There are $f$ nodes on $P'$ that are neither $u$ nor $v$. Additionally, excluding the node $a$, we have $f - 1$ nodes $q$ on $P'$ with $dist(v, q) \neq dist(u, q)$, and they are all in $B_{f+1}(u)$. This contradicts the assumption that there are at most $f - 2$ such nodes. Therefore, this case in not possible.

**Case 2: There exists a node $r \neq u$ on $P_{v,s} \cap P_{v,u}$.** An illustration of this case is presented in Figure 2.

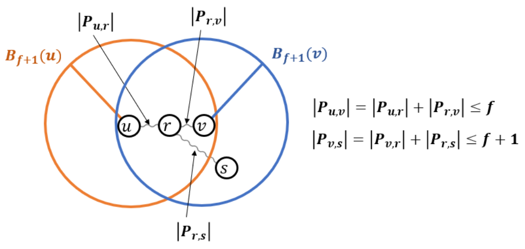
$|P_{u,r}|$
$|P_{r,v}|$
$B_{f+1}(u)$
$B_{f+1}(v)$
$|P_{u,v}| = |P_{u,r}| + |P_{r,v}| \leq f$
$|P_{v,s}| = |P_{v,r}| + |P_{r,s}| \leq f+1$
$u$
$r$
$v$
$s$
$|P_{r,s}|$

**Figure 2** Illustration for Case 2. There exists a node $r \neq v$ on $P_{v,s} \cap P_{v,u}$. We consider $r$ to be the closest node to $u$ in $P_{v,u} \cap P_{v,s}$. The orange circle represents $B_{f+1}(u)$ and the blue circle represents $B_{f+1}(v)$. We assume that $dist(u, v) \leq f$, and since $r$ is on $P_{u,v}$, we have $|P_{u,r}|+|P_{r,v}| \leq f$. Additionally, $s \in B_{f+1}(v) \setminus B_{f+1}(u)$. Since $r$ is on $P_{v,s}$, we have $|P_{v,r}| + |P_{r,s}| \leq f + 1$.

We consider $r$ to be the closest node to $u$ in $P_{v,u} \cap P_{v,s}$. In this case, every node $t \neq u, v$ on $P_{v,u}$ satisfies that $dist(u, t) \neq dist(v, t)$, except at most one node $a$, as we proved in the previous case. Therefore, it is also correct for $P_{u,r}$. Additionally, every node $t$ on $P_{r,s}$ satisfies that $dist(t, v) \neq dist(t, u)$, as otherwise $dist(u, s) \leq dist(u, t) + dist(t, s) = dist(v, t) + dist(t, s) = dist(v, s) \leq f + 1$, contradicting the assumption that $dist(u, s) \geq f + 2$.

Denote by $P'$ the path $P_{u,r} \circ P_{r,s}$ at a distance of at most $f + 1$ from $u$. There are $f + 1$ nodes on $P'$ that are not $u$. Additionally, excluding the node $a$, we have $f$ nodes $q$ on $P'$ with $dist(v, q) \neq dist(u, q)$, and they are all in $B_{f+1}(u)$. This contradicts the assumption that there are at most $f - 2$ such nodes. Therefore, this case in not possible.

This completes the proof that every node $u$ that is still faulty at the beginning of this step and every node $v \in X(u)$ are alternative nodes.

Now, we prove that if $u$ sets $b(u) = 0$ in this step, then it satisfies that $u \notin S$. In this case, since $u$ sets $b(u) = 0$, there exists a node $v \in X(u)$ with $ID(v) < ID(u)$. Since $v$ is an alternative node for $u$ with $ID(v) < ID(u)$, it follows that $u \notin S$ by Lemma 6.

Finally, we prove that every faulty node $u$ that sets $b(u) = 1$ satisfies that $u \in S$. By the definition of a $(2f + 2, 2f + 1)$-ruling set, there exists a node $v \in S$ such that $dist(u, v) \leq 2f + 1$ (which may be $u$ itself). We claim that $v \in B_f(u)$. Assume otherwise, i.e., that

$dist(u,v) \geq f+1$. Notice that $v,u$ have at least $|P_{v,u}|-2$ nodes $j$ with $dist(v,j) \neq dist(u,j)$ on $P_{v,u}$, since there are $|P_{v,u}|-1$ nodes $j \neq u,v$ on $P_{v,u}$, and at most one of them has $dist(v,j) = dist(u,j)$ (such a node $j$ exists if $dist(u,v)$ is even and $j$ is exactly in the middle of $P_{v,u}$). Therefore, if $dist(u,v) \geq f+1$, consider nodes on the path $P_{u,v}$ that are at a distance of at most $f+1$ from $u$. There are at least $f-1$ such nodes $j \neq u,v$ with $dist(u,j) \neq dist(v,j)$, and so at least one of them is not faulty, which would lead to $u$ setting $b(u) = 0$ in Step 8. But, $u$ is faulty at the beginning of Step 9, which leads to a contradiction.

Thus, $v \in B_f(u)$. Additionally, $v \in X(u)$ because in Step 9, $u$ is still faulty, and therefore, in Step 3, $u$ does not set $b(u) = 0$, so $v$ must also still be faulty in Step 3. Since $dist(u,v) \leq 2f+1$ and $v$ is still faulty in Step 4, $u$ receives a message with $ID(v)$ in Step 4. Note that if $v$ is not faulty in Step 9, then either it is non-faulty to begin with, in which case $u$ would have already set $b(u) = 0$ in Step 3, or $v$ sets $b(v) = 1$ in Step 5. The latter is impossible because $dist(u,v) \leq 2f+1$ and $u$ is still faulty in Step 9. Thus, $v$ is still faulty in Step 4, which means that $v$ would receive a message from $u$ in Step 4 and not set $b(v) = 1$ in Step 5. Thus, $v$ is also still faulty in Step 9, and since $dist(u,v) \leq f$, then $v \in X(u)$.

Thus, since $u$ sets $b(u) = 1$, every node $w \in X(u)$ satisfies $ID(u) \leq ID(w)$. Since, $w \in X(u)$, then as we proved, $w$ is an alternative node for $u$. By Lemma 6, this implies that $w$ cannot be in $S$. Since $v \in X(u) \cap S$, it must hold that $v = u$. Thus $u \in S$, as needed.

This completes the proof that, at the end of Algorithm 2, we set $b(w)$ for every node $w \in V$ correctly, satisfying $\{w | b(w) = 1\} = S$. Finally, we sum the number of rounds the algorithm takes. In every broadcast in the algorithm, we have at most $f$ different messages (each of which fits in $O(\log n)$ bits). Moreover, a node passes such a message only once. Thus, each broadcast takes $O(f)$ rounds. Since there are only a constant number of steps with such broadcasts, the algorithm completes in $O(f)$ rounds. ◀

## 4 Partitioning

The nodes use the restored ruling set to partition themselves into groups, with each group containing an error correction code to recover the erased labels. The algorithm they follow is the same as that of the oracle, ensuring determinism and independence from faulty nodes. This guarantees consistent partitioning and correct decoding.

To ensure fast communication within each group, our goal is to partition the graph nodes into groups of size $\Theta(f)$. Our algorithm achieves guarantees that are slightly weaker, in the sense that a group may be a disconnected component in the graph, but its weak diameter (its diameter in the original graph) is still $O(f)$. In general, such a guarantee may still be insufficient for fast communication within each group, as some edges may be used by multiple groups, causing congestion. However, our algorithm will make sure to provide low-congestion shortcuts. That is, it constructs groups of size $\Theta(f)$ with a weak diameter of $O(f)$, such that each edge $e \in E$ is associated with a constant number of groups that communicate over it. A formal definition of this concept is as follows.

▶ **Definition 7** (Low-Congestion Shortcuts [24]). *Given a graph $G = (V,E)$ and a partition of $V$ into disjoint subsets $Q_1, \dots, Q_K \subseteq V$, a set of subgraphs $H_1, \dots H_K \subseteq G$ is called a set of* $(c,d)$-shortcuts*, if:*

1. *The subgraph $G[Q_i] \cup H_i$ is connected.*
2. *For each $i$, the diameter of the subgraph $G[Q_i] \cup H_i$ is at most $d$.*
3. *For each edge $e \in E$, the number of subgraphs $G[Q_i] \cup H_i$ containing $e$ is at most $c$.*

With the above definition, we can now state the properties of our partitioning algorithm.

▶ **Theorem 8.** *Let $G = (V, E)$ be a graph, and let $S$ be a $(2f + 2, 2f + 1)$-ruling set where each node $v$ knows whether it is in $S$. There exists a deterministic* Congest *algorithm that partitions $V$ into disjoint groups $Q_1, \dots Q_K \subseteq V$ with sizes ranging from $f + 1$ to $3f + 1$, and provides a set of $(2, 4f)$-shortcuts for these groups. The algorithm completes in $\Theta(f)$ rounds.*

**Overview.** Algorithm 3 constructs the required partition as follows. Initially, each node $v \in S$ computes a BFS tree to depth $2f + 1$ by propagating its ID. Nodes that receive multiple root IDs choose the smallest and use that tree only. The rest of the algorithm operates within each tree independently.

Within each tree, each node $u$ divides the nodes in its subtree into groups of size $f + 1 \leq s \leq 3f + 1$, with a remainder of up to $f$ nodes. This remaining number is sent to its parent $parent(u)$, who handles dividing the remainders of its children similarly. The `ComputeGroups`$(u)$ procedure sorts $u$'s children by increasing IDs, appends itself, and then forms groups by summing the remaining values of its children. Once a group reaches the desired size, a new group starts from the next child. Each child receives its group ID from $u$. When a node receives its group allocation, it may need to inform some of its children to ensure that remaining nodes from its subtree join this group as well. This is handled by the `RelayGroups`$(u)$ procedure, in which node $u$ relays the group ID. During this process, $u$ also sets the local variable $H_u$ which is used for constructing low-congestion shortcuts for communication among the nodes of each group. Finally, if there is a small remainder at the root $v$, the root appends these nodes to one of the groups created in `ComputeGroups`$(v)$ or, if no such group was constructed by $v$, it appends them to a group with the closest node from this group to $v$. This is done by $v$ obtaining the required group ID and relaying it to the relevant remaining nodes.

Following is the formal algorithm and proof.

**Procedure `ComputeGroups`$(u)$.** The node $u \in V$ initializes a counter $count = 0$, an empty sequence $Groups(u) = \emptyset$, an empty set $Messages(u) = \emptyset$, and a variable $remain(u) = 1$. It then loops over $1 \leq i \leq r_u + 1$ and computes $count \leftarrow count + remain(w_i)$ until $count \geq f + 1$.

When this happens for some index $i = j_1$, the node $u$ adds the item $(1, j_1, ID(w_1))$ to the sequence $Groups(u)$, sets $count \leftarrow 0$, and continues looping over the nodes in $order(u)$ from $i = j_1 + 1$ in the same manner. That is, it computes $count \leftarrow count + remain(w_i)$ until $count \geq f + 1$ for some value $j_2$, after which it adds the item $(j_1, j_2, ID(w_{j_1}))$ to $Groups(u)$, and so on. Thus, every item in $Groups(u)$ is of the form $(start, end, group)$, where $group$ is the ID of the node $w_{start}$, which serves as the group identifier for all nodes $w_i$ for $start \leq i \leq end$.

Let $(a, b, ID(w_a))$ be the last group created by $u$, so $w_b$ is the last child of $u$ whose $remain(w_b)$ value is grouped. The node $u$ computes $remain(u) = (\sum_{i=b+1}^{r_u} remain(w_i)) + 1$. If $u \in S$ and $Groups(u) \neq \emptyset$, $u$ updates $(a, b, ID(w_a))$ to $(a, r_u + 1, ID(w_a))$.

For each item $(start, end, group)$ in $Groups(u)$, $u$ inserts a tuple of $(group, i)$ into $Messages(u)$ for every child $w_i$ where $start \leq i \leq end$ and $remain(w_i) > 0$. Thus, every item is of the form: $(child, group)$. For each item $(child, group) \in Messages(u)$, the node $u$ inserts the edge $(u, w_{child})$ into the set $H_{group}(u)$.

**Procedure `RelayGroups`$(u, group)$.** Let $(a, b, ID(w_a))$ be the last group that $u$ created during its `ComputeGroups`$(u)$ procedure. The node $u$ initializes an empty set $Messages_r(u) = \emptyset$. Then, $u$ inserts a tuple $(i, group)$ into $Messages_r(u)$ for every child $w_i$ where $b+1 \leq i \leq r_u$.

**Algorithm 3** Partitioning algorithm.

```
Input: A graph G and a $(2f+2, 2f+1)$-ruling set $S$, where each node $v$ is provided
       with $b(v)$ such that $S = \{v \mid b(v) = 1\}$.
Output: Each node $v$ holds $Group(v)$ and $H_{group}(v)$ for every group $group$.
Constructing BFS trees:
1 Every node $v \in S$ starts a BFS to a distance of $2f+1$. Every node $u \notin S$ which
  receives BFS messages selects the one with the lowest ID and continues with it.
  Every node $u \notin S$ in a tree rooted at $v$, designates $parent(u)$ as one of the nodes
  from which it received $v$, and sets $leader(u) = v$. Every node $u$ sends a bit “1” to
  $parent(u)$. Let $children(u)$ denote the set of nodes from which $u$ receives such a bit.

Creating groups:
2 Every node $u \in V$ with $children(u) = \emptyset$ sends $remain(u) = 1$ to $parent(u)$. Every
  node $u \in V$ sorts its children by increasing IDs and appends itself last to create a
  sequence $order(u) = (w_1, \dots, w_{r_u}, u = w_{r_u+1})$, where $r_u = |children(u)|$.
3 Every node $u \in V$ that receives $remain(w)$ from every node $w \in children(u)$ locally
  computes $Groups(u)$, $Messages(u)$ and $remain(u)$ using the ComputeGroups$(u)$
  procedure. For each item $(child, group) \in Messages(u)$, $u$ sends $group$ to $w_{child}$.
  Additionally, if $u \notin S$ it sends $remain(u)$ to $parent(u)$.
Relaying group IDs:
4 Every node $u \notin S$ that receives $group$ from its parent computes $Messages_r(u)$ using
  the RelayGroups$(u, group)$ procedure and sets $Group(u) = group$. For each item
  $(child, group) \in Messages_r(u)$, $u$ sends $group$ to its child $w_{child}$.
Root remainder:
5 Every node $u \notin S$ sends a group ID $group$ to $parent(u)$. If $Group(u) \neq \emptyset$ then $group$
  is $Group(u)$. If $Group(u) = \emptyset$ then $group$ is the first group identifier that $u$ receives
  from some child (the smallest ID if there is more than one).
6 Every node $v \in S$ with $Group(v) = \emptyset$ sets $Group(v) = group$, where $group$ is the first
  group identifier that $v$ receives from some child (the smallest ID if there is more
  than one). Then, $v$ sends $group$ to every child $w$ with $remain(w) > 0$ and
  $Group(w) = \emptyset$ and to the node it received $group$ from in Step 5. For every such
  node $w$, the node $u$ inserts the edge $(u, w)$ into the set $H_{group}(u)$.
7 Every node $u \notin S$ with $Group(u) = \emptyset$ which receives $group$ from $parent(u)$ sets
  $Group(u) = group$ and sends $group$ to every child $w$ with $remain(w) > 0$ and
  $Group(w) = \emptyset$ and to the node it received $group$ from in Step 5 (if there exists such
  node). For every such node $w$, the node $u$ inserts the edge $(u, w)$ into $H_{group}(u)$.
Assigning shortcut edges:
8 Denote in $Q_1, \dots, Q_K$ as the groups created by all nodes and define
  $H_i = \cup_{u \in V} H_{ID(Q_i)}(u)$.
```

Thus, every item is of the form $(child, group)$. As in ComputeGroups$(u)$, for each item $(child, group) \in Messages_r(u)$, the node $u$ inserts the edge $(u, w_{child})$ into the set $H_{group}(u)$.

This concludes the description of the algorithm. Appendix A has the proof of Theorem 8, as well as for the following claim derived from it, which will be useful in the subsequent section.

▷ Claim 9. In parallel, each node can collect $Y \cdot f$ bits of information from all nodes in its group within $O(f + (Y \cdot f)/\log n)$ rounds, where each node holds $Y$ bits of information and the group sizes are $O(f)$.

## 5 Resilient Labeling Schemes

In this section, we wrap up the previous parts to obtain our resilient labeling scheme.

▶ **Theorem 1.** *There exists a resilient labeling scheme that tolerates $F$ label erasures, has $O(1)$ and $O(\log F)$ multiplicative and additive overheads to the size of the labels, respectively, and whose distributed* Congest *algorithm for recovering the original labels has a complexity of at most $O(F + (\ell \cdot F)/\log n)$ rounds.*

**Overview.** Algorithm 4 gives our full resilient labeling scheme. The oracle executes Algorithm 1 to assign new labels to each node. In the distributed algorithm, each node parses its new label, which is done correctly since the length of each item in the string is known. The node then uses this label to reconstruct the greedy $(2f + 2, 2f + 1)$-ruling set $S$ (with $f = 80F$) using Algorithm 2 and to obtain its group in the partition according to Algorithm 3. Finally, the nodes exchange all their labels within a group and decode their original labels, as guaranteed by Theorem 3.

**Algorithm 4** Resilient Labeling Scheme.

```
  Input to Oracle: A graph $G$, a parameter $F$, and a labeling $\varphi : V \to \{0,1\}^\ell$
  Output of Oracle: A labeling $\psi : V \to \{0,1\}^{\ell'}$
  Input to Distributed Algorithm: Each node $v$ gets the label $\psi(v)$
  Output of Distributed Algorithm: Each node $v$ outputs $\varphi(v)$
  Oracle:
1 Run Algorithm 1 and assign $\psi(v)$ to each node $v$ .
  The distributed algorithm:
2 For every non-faulty node $v$, parse $\psi(v)$ as $b(v) \circ distS(v) \circ \mathtt{w}(v)$ .
3 Run Algorithm 2 with $\zeta(v) = b(v) \circ distS(v)$ for each non-faulty node $v$ to obtain
   $b(v)$ for every node $v$.
4 Run Algorithm 3 with $b(v)$ for every node $v$, to obtain $Group(v)$ for every node $v$ .
5 Every node $v$ sends $(v, \mathtt{w}(v))$ to all nodes in $Group(v)$ .
6 Every node $v$ outputs $decode(v, \{(u, \mathtt{w}(u)) \mid u \in Group(v)\})$ .
```

**Proof of Theorem 1.** First, note that since $b(v)$ is a single bit and $distS(v)$ has $O(\log F)$ bits where the constant is known, the parsing of $\psi(v)$ as $b(v) \circ distS(v) \circ \mathtt{w}(v)$ in Step 2 is unique and matches Step 3 in Algorithm 1.

**Correctness:** By Theorem 3, the values $b(v)$ correspond to a greedy $(2f + 2, 2f + 1)$-ruling set $S$, meaning that $\{v \mid b(v) = 1\} = S$. Thus, according to Theorem 4, after running Algorithm 2 in Step 3, each node $v$ holds its value $b(v)$. Since Algorithm 3 is deterministic and the oracle uses the same algorithm in Algorithm 1 in Step 1, after running Algorithm 3 in Step 4, each node $v$ will have the same $Group(v)$ as determined by the oracle in Step 2 of Algorithm 1. Therefore, by Theorem 3, the operation $decode(v, \{(u, \mathtt{w}(u)) \mid u \in Group(v)\})$ in Step 6 will yield the value of $\varphi(v)$ for each node $v$.

**Round complexity:** By Theorems 4 and 8, Algorithms 2 and 3 in Steps 3 and 4, respectively, each take $O(f)$ rounds. Additionally, according to Claim 9, every node can collect $O(\ell \cdot f)$ bits of information from all nodes in its group in parallel, within $O(f + (\ell \cdot f)/\log n)$ rounds, as each node holds $O(\ell)$ bits of information and each group is of size $O(f)$. Thus, Step 5 completes in $O(f + (\ell \cdot f)/\log n)$ rounds. In total, the distributed algorithm completes in $O(f + (\ell \cdot f)/\log n)$ rounds. Since $f = \Theta(F)$, we have $O(F + (\ell \cdot F)/\log n)$ rounds.

**Label size:** According to Theorem 3, each of the original $\ell$ bits corresponds to a constant number of bits in the coded version. Additionally, there is one bit for the ruling set indication $b(v)$ and $O(\log F)$ bits for $distS(v)$, making the new label size $O(\log(F) + \ell)$. ◀

## References

1 Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1199–1218, 2012. doi:10.1145/2213977.2214084.

2 Stephen Alstrup, Cyril Gavoille, Haim Kaplan, and Theis Rauhe. Nearest common ancestors: a survey and a new distributed algorithm. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 258–264, 2002. doi:10.1145/564870.564914.

3 Yagel Ashkenazi, Ran Gelles, and Amir Leshem. Noisy beeping networks. *Inf. Comput.*, 289(Part):104925, 2022. doi:10.1016/J.IC.2022.104925.

4 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed Δ-coloring plays hide-and-seek. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 464–477. ACM, 2022. doi:10.1145/3519935.3520027.

5 Aviv Bar-Natan, Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Fault-tolerant distance labeling for planar graphs. *Theoretical Computer Science*, 918:48–59, 2022. doi:10.1016/J.TCS.2022.03.020.

6 Mor Baruch, Pierre Fraigniaud, and Boaz Patt-Shamir. Randomized proof-labeling schemes. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 315–324, 2015. doi:10.1145/2767386.2767421.

7 Aviv Bick, Gillat Kol, and Rotem Oshman. Distributed zero-knowledge proofs over networks. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2426–2458. SIAM, 2022. doi:10.1137/1.9781611977073.97.

8 Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. *Theor. Comput. Sci.*, 811:112–124, 2020. doi:10.1016/J.TCS.2018.08.020.

9 Michal Dory and Merav Parter. Fault-tolerant labeling and compact routing schemes. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 445–455, 2021. doi:10.1145/3465084.3467929.

10 Yuval Emek and Yuval Gil. Twenty-two new approximate proof labeling schemes. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.DISC.2020.20.

11 Yuval Emek, Yuval Gil, and Shay Kutten. Locally Restricted Proof Labeling Schemes. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *LIPIcs*, pages 20:1–20:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2022.20.

12 Salwa Faour, Mohsen Ghaffari, Christoph Grunau, Fabian Kuhn, and Václav Rozhoň. Local distributed rounding: Generalized to mis, matching, set cover, and beyond. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4409–4447. SIAM, 2023. doi:10.1137/1.9781611977554.CH168.

**13** Laurent Feuilloley. Introduction to local certification. *Discrete Mathematics & Theoretical Computer Science*, 23(Distributed Computing and Networking), 2021. `doi:10.46298/DMTCS.6280`.

**14** Laurent Feuilloley and Pierre Fraigniaud. Error-sensitive proof-labeling schemes. *J. Parallel Distributed Comput.*, 166:149–165, 2022. `doi:10.1016/J.JPDC.2022.04.015`.

**15** Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. Redundancy in distributed proofs. *Distributed Computing*, 34:113–132, 2021. `doi:10.1007/S00446-020-00386-Z`.

**16** Orr Fischer, Rotem Oshman, and Dana Shamir. Explicit space-time tradeoffs for proof labeling schemes in graphs with small separators. In Quentin Bramas, Vincent Gramoli, and Alessia Milani, editors, *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, LIPIcs, pages 21:1–21:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.OPODIS.2021.21`.

**17** Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas, and Andrzej Pelc. Distributed computing with advice: information sensitivity of graph coloring. *Distributed Computing*, 21:395–403, 2009. `doi:10.1007/S00446-008-0076-Y`.

**18** Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Communication algorithms with advice. *Journal of Computer and System Sciences*, 76(3-4):222–232, 2010. `doi:10.1016/J.JCSS.2009.07.002`.

**19** Pierre Fraigniaud, Amos Korman, and Emmanuelle Lebhar. Local mst computation with short advice. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 154–160, 2007. `doi:10.1145/1248377.1248402`.

**20** Emanuele G Fusco and Andrzej Pelc. Trade-offs between the size of advice and broadcasting time in trees. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 77–84, 2008. `doi:10.1145/1378533.1378545`.

**21** Cyril Gavoille and David Peleg. Compact and localized distributed data structures. *Distributed Computing*, 16:111–120, 2003. `doi:10.1007/S00446-002-0073-5`.

**22** Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of algorithms*, 53(1):85–112, 2004. `doi:10.1016/J.JALGOR.2004.05.002`.

**23** Paweł Gawrychowski, Fabian Kuhn, Jakub Łopuszański, Konstantinos Panagiotou, and Pascal Su. Labeling schemes for nearest common ancestors through minor-universal trees. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2604–2619. SIAM, 2018.

**24** Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks ii: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 202–219. SIAM, 2016. `doi:10.1137/1.9781611974331.CH16`.

**25** Mika Göös and Jukka Suomela. Locally checkable proofs. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 159–168, 2011. `doi:10.1145/1993806.1993829`.

**26** Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(1):1–33, 2016. `doi:10.4086/TOC.2016.V012A019`.

**27** Rani Izsak and Zeev Nutov. A note on labeling schemes for graph connectivity. *Information processing letters*, 112(1-2):39–43, 2012. `doi:10.1016/J.IPL.2011.10.001`.

**28** Taisuke Izumi, Yuval Emek, Tadashi Wadayama, and Toshimitsu Masuzawa. Deterministic fault-tolerant connectivity labeling scheme. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, pages 190–199, 2023. `doi:10.1145/3583668.3594584`.

**29** Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on information theory*, 18(5):652–656, 1972. `doi:10.1109/TIT.1972.1054893`.

**30** Michal Katz, Nir A Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing*, 34(1):23–40, 2004. `doi:10.1137/S0097539703433912`.

31 Gillat Kol, Rotem Oshman, and Raghuvansh R Saxena. Interactive distributed proofs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 255–264, 2018. URL: https://dl.acm.org/citation.cfm?id=3212771.

32 Amos Korman. Labeling schemes for vertex connectivity. *ACM Transactions on Algorithms (TALG)*, 6(2):1–10, 2010. doi:10.1145/1721837.1721855.

33 Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 26–34, 2006. doi:10.1145/1146381.1146389.

34 Amos Korman and Shay Kutten. On distributed verification. In *Distributed Computing and Networking: 8th International Conference, ICDCN 2006, Guwahati, India, December 27-30, 2006. Proceedings 8*, pages 100–114. Springer, 2006. doi:10.1007/11947950_12.

35 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Comput.*, 22(4):215–233, 2010. doi:10.1007/S00446-010-0095-3.

36 Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

37 Rafail Ostrovsky, Mor Perry, and Will Rosenbaum. Space-time tradeoffs for distributed verification. In Shantanu Das and Sébastien Tixeuil, editors, *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017, Porquerolles, France, June 19-22, 2017, Revised Selected Papers*, volume 10641 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2017. doi:10.1007/978-3-319-72050-0_4.

38 Merav Parter and Asaf Petruschka. Õptimal dual vertex failure connectivity labels. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPIcs*, pages 32:1–32:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.DISC.2022.32.

39 Merav Parter, Asaf Petruschka, and Seth Pettie. Connectivity labeling for multiple vertex failures. *arXiv preprint*, 2023. doi:10.48550/arXiv.2307.06276.

40 Boaz Patt-Shamir and Mor Perry. Proof-labeling schemes: Broadcast, unicast and in between. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 1–17. Springer, 2017. doi:10.1007/978-3-319-69084-1_1.

41 David Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33(3):167–176, 2000. doi:10.1002/(SICI)1097-0118(200003)33:3\%3C167::AID-JGT7\%3E3.0.CO;2-5.

## A Missing Proofs

### A.1 Proof of Theorem 3

In order to prove this theorem, we need the following claim on $N_M$.

▷ Claim 10. For every $M > 1$, it holds that $N_M \le 5N_{M-1}$.

Proof of Claim 10. For every $M > 0$, the value of $N_M$ is defined as $2M(2^M - 1)$, and so we have for $M > 1$:

$$\begin{aligned}
N_M &= 2M(2^M - 1) \\
&= 2(M-1)\cdot 2^M + 2\cdot 2^M - 2(M-1) - 2 \\
&= 2(2(M-1)\cdot 2^{M-1}) - 2(2(M-1)) + 2\cdot 2^M + 2(M-1) - 2 \\
&\le 2N_{M-1} + 2\cdot 2(M-1)(2^{M-1}) - 2\cdot 2(M-1) + 6(M-1) - 2 \\
&= 4N_{M-1} + 6(M-1) - 2 \\
&\le 5N_{M-1},
\end{aligned}$$

where the last inequality follows since $M > 1$ and thus $6(M-1) - 2 < N_{M-1}$. ◁

We now prove the aforementioned properties of the code implied by our choice of parameters using a very crude analysis of the constants, which is sufficient for our needs.

**Proof of Theorem 3.** Since $M$ is the smallest such that $k_M \geq s$, we have that $k_{M-1} < s$, or in other words, $\rho_{M-1} N_{M-1} < s$. We prove in Claim 10 above that $N_M$ is at most $5N_{M-1}$. Combining these two with the fact that $\rho_{M-1} \geq \rho = 1/4$, gives that

$$(1/4)N_{M-1} \leq \rho_{M-1}N_{M-1} \leq s \leq k_M = \rho_M N_M \leq \rho_M \cdot 5N_{M-1} \leq 5N_{M-1},$$

where the last inequality follows since $\rho_M \leq 1$. This gives that $k_M/s$ is at most 20. To encode, we take the $s$ bits and pad them with $k_M - s$ zeros concatenated at their end. We get $N_M$ bits that are split among the $s$ nodes, so now the number of bits that each node has is at most $N_M/s = k_M/(\rho_M \cdot s)$. We know that $\rho_M \geq 1/4$ and that $k_M/s$ is at most 20, so each node holds at most 80 bits. This proves Item (1).

To prove Item (2), note that the code $\mathtt{C}_M$ has a relative distance $\delta_M > (1-2\rho)H^{-1}(1/2) \geq (1 - 2 \cdot (1/4)) \cdot 0.11 \geq 1/20$. The code can fix up to $\delta_M N_M - 1$ erasures, which is more than a 1/40 fraction of $N_M$. Since $f = 80F$ and the number of blocks $s$ is at least $f + 1$ (which is at least $f$), any erasure of $F$ blocks erases at most a 1/80 fraction of the blocks. Some blocks may be of size $\lceil |\mathtt{w}|/s \rceil$ and others of size $\lfloor |\mathtt{w}|/s \rfloor$, which may differ by 1 and the larger blocks may get erased. However, even if we denote $x = \lceil |\mathtt{w}|/s \rceil$ and assume $\lfloor |\mathtt{w}|/s \rfloor = x - 1$, then in the worst case, the fraction of bit erasures caused by $F$ block erasures is at most $(sx/80)/(sx - (79s/80)) = x/(80x - 79)$, which is at most 1/40 (since $x > 1$). Thus, this may erase at most a 1/40 fraction of bits of $\mathtt{w}$. Because the code can fix up to a 1/40 fraction of erasures, we can decode $\mathtt{r}$, obtaining Item (2) as needed. ◀

## A.2 Proof of Lemma 6

**Proof of Lemma 6.** Let $v$ and $u$ be alternative nodes, and let $w \neq v, u$. We claim that $dist(v, w) \leq 2f + 1$ if and only if $dist(u, w) \leq 2f + 1$. This implies that if $v \in S$ then $(S \setminus \{v\}) \cup \{u\}$ is a $(2f + 2, 2f + 1)$-ruling set. In particular, this means that apart from $v$, the node $u$ does not have a node $w \in S$ within distance at most $2f + 1$ from it. Therefore, $ID(u)$ cannot be smaller than $ID(v)$, since otherwise the greedy algorithm would reach $u$ before $v$ when iterating over the nodes and would inserted $u$ into $S$.

To prove that $dist(v, w) \leq 2f + 1$ if and only if $dist(u, w) \leq 2f + 1$, we assume that $dist(v, w) \leq 2f + 1$ and prove that $dist(w, u) \leq 2f + 1$. The proof for the other direction is symmetric. Assume towards a contradiction that it is not the case, so $dist(u, w) \geq 2f + 2$. Thus, using the triangle inequality we have

$$2f + 2 \leq dist(u, w) \leq dist(u, v) + dist(v, w) \leq f + dist(v, w),$$

where the last inequality follows from the definition of $u, v$ as alternative nodes. Thus, $dist(v, w) \geq f + 2$. Assume towards a contradiction that there is a node $t$ on $P_{v,w}$ such that $dist(v, t) = dist(u, t)$. Since $t$ is on $P_{v,w}$ then $|P_{v,w}| = |P_{v,t}| + |P_{t,w}|$. Therefore,

$$\begin{aligned} dist(u, w) &\leq |P_{u,t}| + |P_{t,w}| \\ &= |P_{u,t}| + (|P_{v,w}| - |P_{v,t}|) \\ &= dist(u, t) + (dist(v, w) - dist(v, t)) \\ &= dist(v, t) + (dist(v, w) - dist(v, t)) \\ &= dist(v, w) \\ &\leq 2f + 1. \end{aligned}$$

This contradicts the assumption that $dist(u,w) \geq 2f+2$. Thus, every node $t$ on $P_{v,w}$ satisfies $dist(u,t) \neq dist(v,t)$.

Because $dist(v,w) \geq f+2$, there are at least $f+1$ nodes $q$ on $P_{v,w}$ such that $dist(q,u) \neq dist(q,v)$, and at least $f-1$ of them are at a distance of at most $f+1$ from $v$. Thus, $u$ is not an alternative node for $v$, contradicting the assumption. Therefore, $dist(u,w) \leq 2f+1$. ◀

## A.3 Proof of Theorem 8

**Proof of Theorem 8.** Since $S$ is a $(2f+2, 2f+1)$-ruling set, every node $u \in V$ has a node $v \in S$ within distance of at most $2f+1$ from it. Thus, $u$ belongs to exactly one tree, and in the BFS in Step 1 in the algorithm, the node $u$ chooses one node to be its leader. Let $T_v$ be the tree rooted at $v$.

**The BFS tree $T_v$ of every node $v \in S$ satisfies that $|T_v| \geq f+1$.** For every node $v \in S$, there are at least $f+1$ nodes within distance $f$ from it (including $v$ itself), since the graph is connected. These nodes cannot be within distance $f$ from any other node $v' \in S$, and thus they join the BFS tree $T_v$. This proves that $|T_v| \geq f+1$ for all trees. In particular, there exists a partition of the nodes in $T_v$ into groups of sizes at least $f+1$.

**The algorithm obtains a partitioning of the nodes in every tree into groups with sizes ranging from $f+1$ to $3f+1$.** Fix $v \in S$ and consider only nodes in $T_v$. First, we prove that all groups created until the end of Step 4 are of size ranging from $f+1$ to $2f+1$. Let $u \in T_v$. In Step 3, the node $u$ runs the procedure `ComputeGroups`$(u)$. During the procedure, when creating a group, the node $u$ starts with the first child in $order(u)$ and adds children according to $order(u)$ until the counter $count$ of their $remain$ values is at least $f+1$. For a group that starts with child $w_{a'}$ and ends with child $w_{b'}$, in Step 3 the node $u$ sends this group's identifier to every child $w_i$ for which $a' \leq i \leq b'$. Then, each $w_i$ sends in `RelayGroups`$(w_i)$ in Step 4 this group identifier to all its children for which it did not assign a group yet during the procedure `ComputeGroups`$(w_i)$, and its children relay this identifier in the same manner. Thus, the value of $count$ is the size of the group, and so this size is at least $f+1$.

Notice that if $count$ is at most $f$, when adding $remain(w') \leq f$ of some child $w'$, we get that the updated $count$ is at most $f+f=2f$ (any $remain$ value is always at most $f$, as otherwise $w'$ would form a group from these at least $f+1$ nodes). A special case is when a root of a subtree $u$ adds itself to a group it created, and then $count$ is at most $2f+1$. Therefore, since the size of a group is the value of $count$, then the size of a group ranges from $f+1$ to $2f+1$.

This proves that all groups created until the end of Step 4 are of size ranging from $f+1$ to $2f+1$, but there still could be some values that $u$ needs to handle whose sum is $remain(u)$. For every node $u \notin S$, an ancestor of $u$ handles these values. But, the algorithm for the root $v \in S$ in this case differs, as it has no ancestors. Steps 5-7 provide the root with a group ID to which it adds its remainder, so the size of that group may become at most $2f+1+f=3f+1$. Thus, all the group sizes range from $f+1$ to $3f+1$.

Notice that since every node belongs to exactly one group, then all the groups are disjoint.

**For each edge $e \in E$, the number of subgraphs $G[Q_i] \cup H_i$ containing $e$ is at most 2.** An edge $(u,w)$ is inserted into $H_{group}(u)$ if $u$ sends the group identifier $group$ to $w$ and this happens only if $w$ sends $remain(w) > 0$ to $u = parent(w)$ in Step 3. There are two cases. The first is that $u$ assigns $w$ a group during the steps of creating groups. Thus, $u$ does not assign $w$ another group in this or later steps. Therefore, in this case, the edge $(u, w)$ is inserted into only one set $H_{group}(u)$ of $u$. The same argument applies if the edge is inserted into $H_{group}(u)$ during the steps of relaying group IDs.

The second case is that $u$ receives *group* from its parent during the steps of handling the root remainder. Then, the root $v$ satisfies that $remain(v) > 0$. Thus, according to the algorithm, it assigns additional $remain(v)$ nodes to an existing group. Therefore, $(u, w)$ can be inserted into at most one additional group. Overall, there are at most two groups for which $u$ assigns $e = (u, w)$ to $H_{group}(u)$.

**The diameter of the subgraph $G[Q_i] \cup H_i$ is at most $4f$.** Consider two cases, based on whether the group contains the root $v$ or not.

Case 1: Let $Q$ be a group which does not contain the root $v$, and let $u$ be the node that builds $Q$ during `ComputeGroups`$(u)$. Since the group identifier is the ID of the first child in the group, there can be only a single $u$ that creates a group with this ID, and only nodes in the subtree of $u$ can belong to it. Let $t \in Q$. In Step 4, $parent(t)$ sends $ID(Q)$ to $t$ and inserts the edge $(parent(t), t)$ into $H_{ID(Q)}(parent(t))$. Additionally, if $u \neq parent(t)$, then $parent(t)$ also receives $ID(Q)$ from its parent and sets $Group(parent(t)) = Q$ in Step 4 of procedure `RelayGroups`$(parent(t))$. This process continues until we reach a node $w$ which is a child of $u$. Since $w$ also receives $ID(Q)$ from $u$ in Step 3, then $u$ inserts the edge $(u, w)$ into $H_{ID(Q)}(u)$, and $w$ sets $Group(w) = Q$ in Step 4 of procedure `RelayGroups`$(w)$. Thus, on the path from $u$ to $t$, all the edges belong to $H_{ID(Q)}$, and all the nodes on the path from $w$ to $t$ belong to $Q$ including $w$ and $t$. Since this holds for every node $t \in Q$, the subgraph $G[Q] \cup H_{ID(Q)}$ is connected, and the distance between $u$ and $t$ in the tree is bounded by $f$, as otherwise, $w$ has enough nodes to create a group of size $f + 1$ which includes $t$, contradicting the fact that $t$ is in the group $Q$ created by $u$. Thus, the diameter of every such group is bounded by $2f$.

Case 2: Let $Q$ be the group which contains the root $v$. There are two possibilities. If $Groups(v) \neq \emptyset$, then the remainder of $v$ is assigned by $v$ itself to one of the groups it creates, and the same argument as in the previous case applies.

Otherwise, if $Groups(v) = \emptyset$, then in the steps handling the root remainder, the node $v$ assigns its remainder of nodes to a group that one of its descendants created. We prove that the distance of $v$ from the deepest node in $Q$ is bounded by $2f$. This implies that the diameter of $G[Q] \cup H_i$ is bounded by $4f$.

Let $t$ be the node which created the group $Q$ in its `ComputeGroups`$(t)$ procedure. If $t \in Q$ before handling the root remainder, then $Group(parent(t)) = \emptyset$ at that point since otherwise, $v$ would receive $Group(parent(t))$ during Step 5 instead of $Group(t)$ and thus $v$ would not be in $Q$ that $t$ created. Thus, $parent(t)$ sets $Group(parent(t)) = ID(Q)$ and the same argument applies to every ancestor of $parent(t)$. This implies that the distance from $t$ to the root $v$ is at most $f$, since all of the nodes on that path join $Q$, and if the distance was larger then there would be enough nodes to create a group that is separate from $Q$. Moreover, the distance from $t$ to every node in its subtree that is in $Q$ is at most $f$, as otherwise the child of $t$ in that subtree would have $f + 1$ to create that group. Together, this means that the distance from $v$ of any node in $Q$ is at most $f + f = 2f$.

If $t \notin Q$ before handling the root remainder, then a similar argument applies: any child $w$ of $t$ which is in $Q$ has distance at most $f - 1$ to any other node in its subtree that is in $Q$. The node $t$ itself is again within distance at most $f$ to $v$. Thus, the distance from $v$ of any node in $Q$ is at most $(f - 1) + 1 + f = 2f$.

The above proves that the set of subgraphs $H_1, \ldots, H_K$ is a set of $(2, 4f)$-shortcuts.

**Round complexity:** Note that for every $v \in S$ and $u \in T_v \setminus \{v\}$, we have that $dist(u, v) \leq 2f + 1$. Thus, downcasting and upcasting $f$ messages in $T_v$ takes at most $O(f)$ rounds by standard pipelining. Since we only downcast or upcast a constant number of times, we have that the algorithm completes in $O(f)$ rounds. ◀

## A.4 Proof of Claim 9

Proof of Claim 9. Every node $u \in V$ holds a set $H_{group}(u)$ for every group *group* that needs to communicate through $u$. This set contain edges to children of $u$. From Theorem 8, for each edge $e \in E$, the number of subgraphs $G[Q_i] \cup H_i$ containing $e$ is at most 2. Thus, every node $u$ can send to its children which edges belong to which group so that the group can communicate over them. Thus, all groups can communicate over their edges in parallel. Additionally, the size of every group $Q_i$ is $O(f)$. Therefore, on every edge, we need to pass $O(f)$ messages, where each message is of size $Y$. Moreover, according to Theorem 8, for each group $i$, the diameter of the subgraph $G[Q_i] \cup H_i$ is at most $4f = O(f)$. Hence, by standard pipelining, for every node, it can collect $Y \cdot f$ bits of information from all nodes in its group, within $O(f + (Y \cdot f)/\log n)$ rounds. ◁