Network Coding for Distributed Storage Systems

Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu,
Martin Wainwright and Kannan Ramchandran

***Abstract*—Distributed storage systems provide reliable access to data through redundancy spread over individually unreliable nodes. Application scenarios include data centers, peer-to-peer storage systems, and storage in wireless networks. Storing data using an erasure code, in fragments spread across nodes, requires less redundancy than simple replication for the same level of reliability. However, since fragments must be periodically replaced as nodes fail, a key question is how to generate encoded fragments in a distributed way while transferring as little data as possible across the network.**

For an erasure coded system, a common practice to repair from a single node failure is for a new node to reconstruct the whole encoded data object to generate just one encoded block. We show that this procedure is sub-optimal. We introduce the notion of regenerating codes, which allow a new node to communicate *functions* of the stored data from the surviving nodes. We show that regenerating codes can significantly reduce the repair bandwidth. Further, we show that there is a fundamental tradeoff between storage and repair bandwidth which we theoretically characterize using flow arguments on an appropriately constructed graph. By invoking constructive results in network coding, we introduce regenerating codes that can achieve any point in this optimal tradeoff.

Keywords: Distributed Storage, Network Coding, Regenerating Codes, Peer-to-peer Storage.

I. Introduction

The purpose of distributed storage systems is to store data reliably over long periods of time using a distributed collection of storage nodes which may be individually unreliable. Applications involve storage in large data centers and peer-to-peer storage systems such as OceanStore [3], Total Recall [4], and DHash++ [5], that use nodes across the Internet for distributed file storage. In wireless sensor networks, obtaining reliable storage over unreliable motes might be desirable for robust data recovery [6], especially in catastrophic scenarios [7].

In all these scenarios, ensuring reliability requires the introduction of redundancy. The simplest form of redundancy is replication, which is adopted in many practical storage systems. As a generalization of replication, erasure coding offers better storage efficiency. For instance, we can divide a file of size $\mathcal{M}$ into k pieces (to be called fragments), each of size $\mathcal{M}/k$, encode them into n encoded fragments (of the same size) using an (n, k) maximum distance separable (MDS) code, and store them at n nodes. Then, the original file can be recovered from any set of k coded fragments. This performance is optimal in terms of the redundancy–reliability tradeoff because k pieces, each of size $\mathcal{M}/k$, provide the minimum data for recovering the file, which is of size $\mathcal{M}$. Several designs [8], [4], [5] use erasure codes instead of replication. For certain cases, erasure coding can achieve orders of magnitude higher reliability for the same redundancy factor compared to replication; see, e.g., [9].

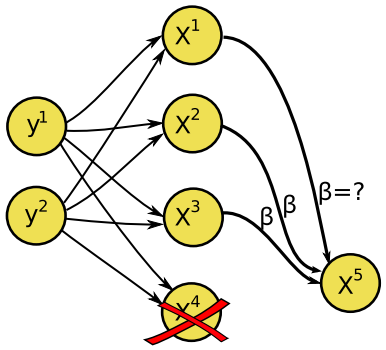


Fig. 1. The repair problem: Assume that a (4,2) MDS erasure code is used to generate 4 fragments (stored in nodes $x^1, \ldots x^4$) with the property that any 2 can be used to reconstruct the original data y^1, y^2. When node x^4 fails, and a newcomer x^5 needs to generate an erasure fragment from $x^1, \ldots x^3$, what is the minimum amount of information that needs to be communicated?

However, a complication arises: In distributed storage systems, redundancy must be continually refreshed as nodes fail or leave the system, which involves large data transfers across the network. This problem is best illustrated in the simple example of Fig. 1: a data object is divided in two fragments y^1, y^2 (say, each of size 1Mb) and these encoded into four fragments $x^1, \ldots x^4$ of same size, with the property that any two out of the four can be used to recover the original y^1, y^2. Now assume that storage node x^4 fails and a new node x^5, the newcomer, needs to communicate with existing nodes and create a new encoded packet, such that any two out of x^1, x^2, x^3, x^5 suffice to recover. Clearly, if the newcomer can receive any two encoded fragments (say from x^1, x^2), reconstruction of the whole data object is possible and then a new encoded fragment can be generated (for example by making a new linear combination that is independent from the existing ones). This, however, requires the communication of 2Mb in the network to generate an erasure encoded fragment

A.G. Dimakis is with the University of Southern California, { dimakis@usc.edu }, P.B. Godfrey is with the University of Illinois, Urbana-Champaign {pbg@illinois.edu}, Y. Wu is with Microsoft Research {yunnanwu@microsoft.com} and M. Wainwright and K. Ramchandran with UC Berkeley {wainwrig,kannanr}@eecs.berkeley.edu This work was performed while the first two authors were with the University of California, Berkeley. Results in this paper have appeared in part in [1] and [2]. This research was supported by NSF under grants CCR-0219722, CCR- 0330514 and a Microsoft Research Fellowship.

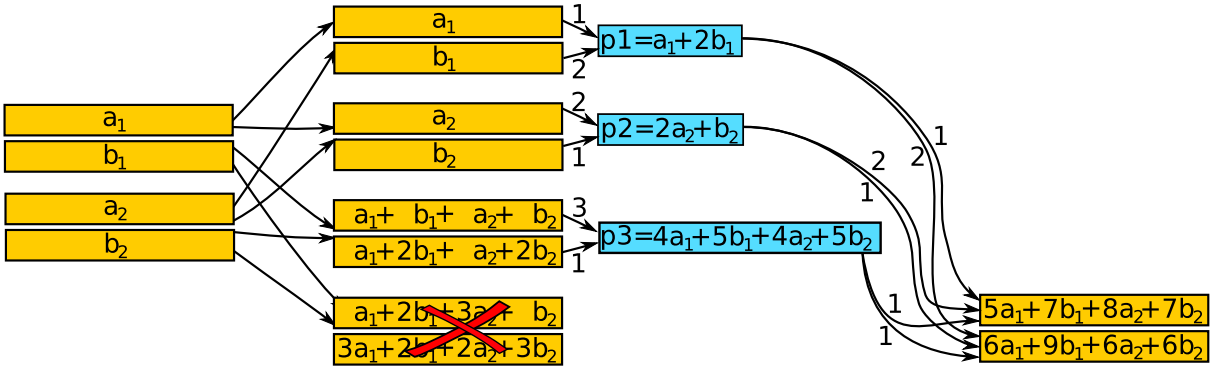


Fig. 2. Example: A repair for a (4,2)-Minimum-Storage Regenerating Code. All the packets (boxes) in this figure have size 0.5Mb and each node stores two packets. Note that any two nodes have four equations that can be used to recover the data, a_1, a_2, b_1, b_2. The parity packets p_1, p_2, p_3 are used to create the two packets of the newcomer, requiring repair bandwidth of 1.5MB. The multiplying coefficients are selected at random and the example is shown over the integers for simplicity (although any sufficiently large field would be enough). The key point is that nodes do not send their information but generate smaller parity packets of their data, and forward them to the newcomer, who further mixes them to generate two new packets. Note that the linear combination coefficients are recorded in the packet to identify the packet composition.

of size 1Mb at x^5. In general, if an object of size $\mathcal{M}$ is divided in k initial fragments, the repair bandwidth with this strategy is $\mathcal{M}$ bits to generate a fragment of size $\mathcal{M}/k$. In contrast, if replication is used instead, a new replica may simply be copied from any other existing node, incurring no bandwidth overhead. It was commonly believed that this k-factor overhead in repair bandwidth is an unavoidable disadvantage that comes with the benefits of coding (see, for example, [10]). Indeed, all known coding constructions require access to the original data object to generate encoded fragments.

In this paper we show that, surprisingly, there exist erasure codes that can be repaired without communicating the whole data object. In particular, for the $(4, 2)$ example, we show that the newcomer can communicate 1.5Mb to repair a failure and that this is the information theoretic minimum (see Fig. 2 for an example). More generally, we identify a tradeoff between storage and repair bandwidth and show that codes exist that achieve every point on this optimal tradeoff curve. We call codes that lie on this optimal tradeoff curve *regenerating codes*. Note that the tradeoff region computed corrects an error in the threshold a_c computed in [1] and generalizes the result to every feasible (α, γ) pair.

The two extremal points on the tradeoff curve are of special interest and we refer to them as minimum-storage regenerating (MSR) codes and minimum-bandwidth regenerating (MBR) codes. The former correspond to Maximum Distance Separable (MDS) codes that can also be efficiently repaired. At the other end of the tradeoff are the MBR codes, which have minimum repair bandwidth. The benefit of the MBR constructions is that if each storage node is allowed to store slightly more than $\mathcal{M}/k$ bits, the repair bandwidth can be significantly reduced.

The remainder of this paper is organized as follows: In Section II we discuss relevant background and related work from network coding theory and distributed storage systems. In Section III we introduce the notion of the information flow graph, which represents how information is communicated and stored in the network as nodes join and leave the system. In Section III-B we characterize the minimum storage and repair bandwidth and show that there is a tradeoff between these two quantities that can be expressed in terms of a maximum flow on this graph. We further show that for any finite information flow graph, there exists a regenerating code that can achieve any point on the minimum storage/ bandwidth feasible region we computed. In Section IV we evaluate the performance of the proposed regenerating codes using traces of failures in real systems and compare to alternative schemes previously proposed in the distributed storage literature. Finally, the technical part of our theoretical analysis can be found in the Appendix.

II. Background and Related Work

A. Erasure codes

Classical coding theory focuses on the tradeoff between redundancy and error tolerance. In terms of the redundancy-reliability tradeoff, the Maximum Distance Separable (MDS) codes are optimal. The most well-known family of MDS erasure codes is Reed-Solomon codes. More recent studies on erasure coding focus on other performance metrics. For instance, sparse graph codes [11], [12], [13] can achieve near-optimal performance in terms of the redundancy-reliability tradeoff and also require low encoding and decoding complexity. Another line of research for erasure coding in storage applications is parity array codes; see, e.g., [14], [15], [16], [17]. The array codes are based solely on XOR operations and they are generally designed with the objective of low encoding, decoding, and update complexities. See also the tutorial by Plank [18] on erasure coding for storage applications.

Compared to these studies, this paper focuses on different performance metrics. Specifically, motivated by practical concerns in large distributed storage systems, we explore erasure codes that offer good tradeoffs in terms of redundancy, reliability, and *repair bandwidth* tradeoff.

B. Network Coding

Network coding is a generalization of the conventional routing (store-and-forwarding) method. In conventional routing, each intermediate node in the network simply stores and forwards the information received. In contrast, network coding allows the intermediate nodes to generate output data by encoding (i.e., computing certain functions of) previously received input data. Thus, network coding allows information to be "mixed" at intermediate nodes. The potential advantages of network coding over routing include resource (e.g., bandwidth and power) efficiency, computational efficiency, and robustness to network dynamics. As shown by the pioneering work of Ahlswede et al. [19], network coding can achieve the cut-set bound throughput for the multicasting case.

Subsequent work [20], [21] showed that linear network codes suffice for the multicasting problem. The studies by Ho *et al.* [22] and Sanders *et al.* [23] further showed that random linear network coding over a sufficiently large finite field can (asymptotically) achieve the multicast capacity. A polynomial complexity procedure to construct deterministic network codes that achieve the multicast capacity is given by Jaggi *et al.* [24].

For distributed storage, network coding was introduced in [25], [6] for wireless sensor networks. Many aspects of coding were explored [7], [26], [27], [28] for networked storage applications.

Compared to previous studies of network coding, this paper investigates the application of network coding for the *repair* problem in distributed storage. The consideration of repair network bandwidth and the notion of code maintenance over time make the problem very unique. Our central objective is to investigate the fundamental tradeoff between storage and repair network bandwidth.

C. Distributed storage systems

A number of recent studies [8], [29], [30], [4] have designed and evaluated large-scale, peer-to-peer distributed storage systems. Redundancy management strategies for such systems have been evaluated in [9], [4], [10], [31], [32].

Among these, [9], [4], [10] compared replication with erasure codes in the bandwidth-reliability tradeoff space. The analysis of Weatherspoon and Kubiatowicz [9] showed that erasure codes could reduce bandwidth use by an order of magnitude compared with replication. Bhagwan et al. [4] came to a similar conclusion in a simulation of the Total Recall storage system.

Rodrigues and Liskov [10] propose a solution to the repair problem that we call the *Hybrid strategy*: one special storage node maintains one full replica in addition to multiple erasure-coded fragments. The node storing the replica can produce new fragments and send them to newcomers, thus transferring just $\mathcal{M}/k$ bytes for a new fragment. However, maintaining an extra replica on one node dilutes the bandwidth-efficiency of erasure codes and complicates system design. For example, if the replica is lost, new fragments cannot be created until it is restored. The authors show that in high-churn environments (i.e., high rate of node joins/leaves), erasure codes provide a large storage benefits but the bandwidth cost is too high to be practical for a P2P distributed storage system, using the Hybrid strategy. In low-churn environments, the reduction in bandwidth is negligible. In moderate-churn environments, there is some benefit, but this may be outweighed by the added architectural complexity that erasure codes introduce as discussed further in Section IV-E. These conclusions were based on an analytical model augmented with parameters estimated from traces of real systems. Compared with [9], [10] used a much smaller value of k (7 instead of 32) and the Hybrid strategy to address the code regeneration problem. In Section IV, we follow the evaluation methodology of [10] to measure the performance of the two redundancy maintenance schemes that we introduce.

III. Analysis

Our analysis is based on a particular graphical representation of a distributed storage system, which we refer to as an *information flow graph* $\mathcal{G}$. This graph describes how the information of the data object is communicated through the network, stored in nodes with limited memory, and reaches reconstruction points at the data collectors.

A. Information Flow Graph

The information flow graph is a directed acyclic graph consisting of three kinds of nodes: a single data source S, storage nodes $\mathsf{x}_{in}^i, \mathsf{x}_{out}^i$ and data collectors DC_i. The single node S corresponds to the source of the original data. Storage node i in the system is represented by a storage input node x_{in}^i, and a storage output node x_{out}^i; these two nodes are connected by a directed edge $\mathsf{x}_{in}^i \rightarrow \mathsf{x}_{out}^i$ with capacity equal to the amount of data stored at node i. See Figure 3 for an illustration. Jiang [28] independently proposed a construction very similar to the information flow graph, but for optimizing a different objective.

Given the dynamic nature of the storage systems that we consider, the information flow graph also evolves in time. At any given time, each vertex in the graph is either *active* or *inactive*, depending on whether it is available in the network. At the initial time, only the source node S is active; it then contacts an initial set of storage nodes, and connects to their inputs (x_{in}) with directed edges of infinite capacity. From this point onwards, the original source node S becomes and remains inactive. At the next time step, the initially chosen storage nodes become now active; they represent a distributed erasure code, corresponding to the desired state of the system. If a new node j joins the system, it can only be connected with active nodes. If the newcomer j chooses to connect with active storage node i, then we add a directed edge from x_{out}^i to x_{in}^j, with capacity equal to the amount of information communicated from node i to the newcomer. Note that it might be necessary for nodes to communicate more data than they

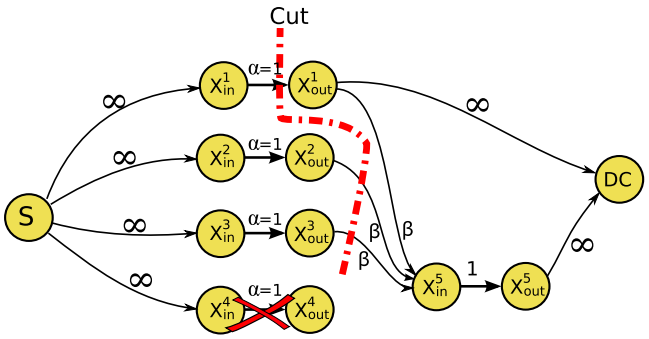


Fig. 3. Illustration of the information flow graph $\mathcal{G}$ corresponding to the (4,2) code of figure 1. A distributed storage scheme uses an $(4, 2)$ erasure code in which any 2 fragments suffice to recover the original data. If node x^4 becomes unavailable and a new node joins the system, we need to construct new encoded fragment in x^5. To do so, node x^5_{in} is connected to the $d = 3$ active storage nodes. Assuming β bits communicated from each active storage node, of interest is the minimum β required. The min-cut separating the source and the data collector must be larger than $\mathcal{M} = 2$Mb for reconstruction to be possible. For this graph, the min-cut value is given by $1 + 2\beta$, implying that $\beta \geq 0.5$Mb is sufficient and necessary.

store during repair, (i.e. $d\beta > \alpha$) as in the example of the $(4, 2)$-erasure code. If a node leaves the system, it becomes inactive. Finally, a data collector DC is a node that corresponds to a request to reconstruct the data. Data collectors connect to subsets of active nodes through edges with infinite capacity.

An important notion associated with the information flow graph is that of minimum cuts: A (directed) cut in the graph $\mathcal{G}$ between the source S and a fixed data collector node DC is a subset C of edges such that, there is no directed path starting from S to DC that does not have one or more edges in C. The minimum cut is the cut between S and DC in which the total sum of the edge capacities is smallest.

B. Storage-Bandwidth Tradeoff

We are now ready for the main result of this paper, the characterization of the feasible storage-repair bandwidth points. The setup is as follows: The normal redundancy we want to maintain requires n active storage nodes, each storing α bits. Whenever a node fails, a newcomer communicates β bits each from any d surviving nodes. Therefore the total repair bandwidth is $\gamma = d\beta$ (see figure 3). We restrict our attention to the symmetric setup where it is required that any k storage nodes can recover the original file, and a newcomer receives the same amount of information from each of the existing nodes.

For each set of parameters $(n, k, d, \alpha, \gamma)$, there is a family of information flow graphs, each of which corresponds to a particular evolution of node failures/repairs. We denote this family of directed acyclic graphs by $\mathcal{G}(n, k, d, \alpha, \gamma)$. An $(n, k, d, \alpha, \gamma)$ tuple will be feasible, if a code with storage α and repair bandwidth γ exists. For the example in figure 3, the point $(4, 2, 3, 1\text{Mb}, 1.5\text{Mb})$ is feasible (and a code that achieves it is shown in figure 2) and also on the optimal tradeoff whereas a standard erasure code which communicates the whole data object would correspond to $\gamma = 2$Mb instead. Note that n, k, d must be integers, if there is one failure the newcomer can connect to at most to all the $n - 1$ surviving nodes, so $d \leq n - 1$ and $\alpha, \beta, \gamma = d\beta$ are the non-negative real valued parameters of the repair process.

Theorem 1: For any $\alpha \geq \alpha^*(n, k, d, \gamma)$, the points $(n, k, d, \alpha, \gamma)$ are feasible, and linear network codes suffice to achieve them. It is information theoretically impossible to achieve points with $\alpha < \alpha^*(n, k, d, \gamma)$. The threshold function $\alpha^*(n, k, d, \gamma)$ is the following:

$$\alpha^*(n,k,d,\gamma) = \begin{cases} \frac{\mathcal{M}}{k}, & \gamma \in [f(0), +\infty) \\ \frac{\mathcal{M}-g(i)\gamma}{k-i}, & \gamma \in [f(i), f(i-1)), \end{cases} \quad (1)$$

where

$$f(i) \triangleq \frac{2\mathcal{M}d}{(2k-i-1)i + 2k(d-k+1)}, \quad (2)$$

$$g(i) \triangleq \frac{(2d-2k+i+1)i}{2d}, \quad (3)$$

where $d \leq n - 1$. For d, n, k given, the minimum repair bandwidth γ is

$$\gamma_{\min} = f(k-1) = \frac{2\mathcal{M}d}{2kd - k^2 + k}. \quad (4)$$

The complete proof of this theorem is given in the Appendix. One important observation is that the minimum repair bandwidth $\gamma = d\beta$ is a decreasing function of the number d of nodes that participate in the repair. While the newcomer communicates with more nodes, the size of each communicated packet β becomes smaller fast enough to make the product $d\beta$ decrease, as d increases, and therefore minimal for $d = n-1$.

The main idea of our analysis is that the code repair problem can be mapped to a multicasting problem on the information flow graph. Building on known results on network coding for multicasting, code repair can be achieved if and only if the underlying information flow graph has sufficiently large min-cuts. This condition leads to the repair rates computed in Theorem 1, and when these conditions are met, deterministic regenerating codes can be found (for a fixed number of failures) using the algorithm by Jaggi et al. [24]. Further, simple random linear combinations will suffice with high probability as the field size over which coding is performed grows, as shown by Ho. et al. [22]. The bulk of the technical analysis of the proof involves computing the minimum cuts on arbitrary graphs in $\mathcal{G}(n, k, d, \alpha, \gamma)$ and solving an optimization problem for minimizing α subject to a sufficient flow constraint.

The optimal tradeoff curves for $k = 5, n = 10, d = 9$ and $k = 10, n = 15, d = 14$ are shown in Figure 4 (top) and (bottom), respectively.

C. Special Cases: Minimum-Storage Regenerating (MSR) Codes and Minimum-Bandwidth Regenerating (MBR) Codes

We now study two extremal points on the optimal tradeoff curve, which correspond to the best storage efficiency and the minimum repair bandwidth, respectively. We call codes that attain these points minimum-storage regenerating (MSR) codes and minimum-bandwidth regenerating (MBR) codes, respectively.

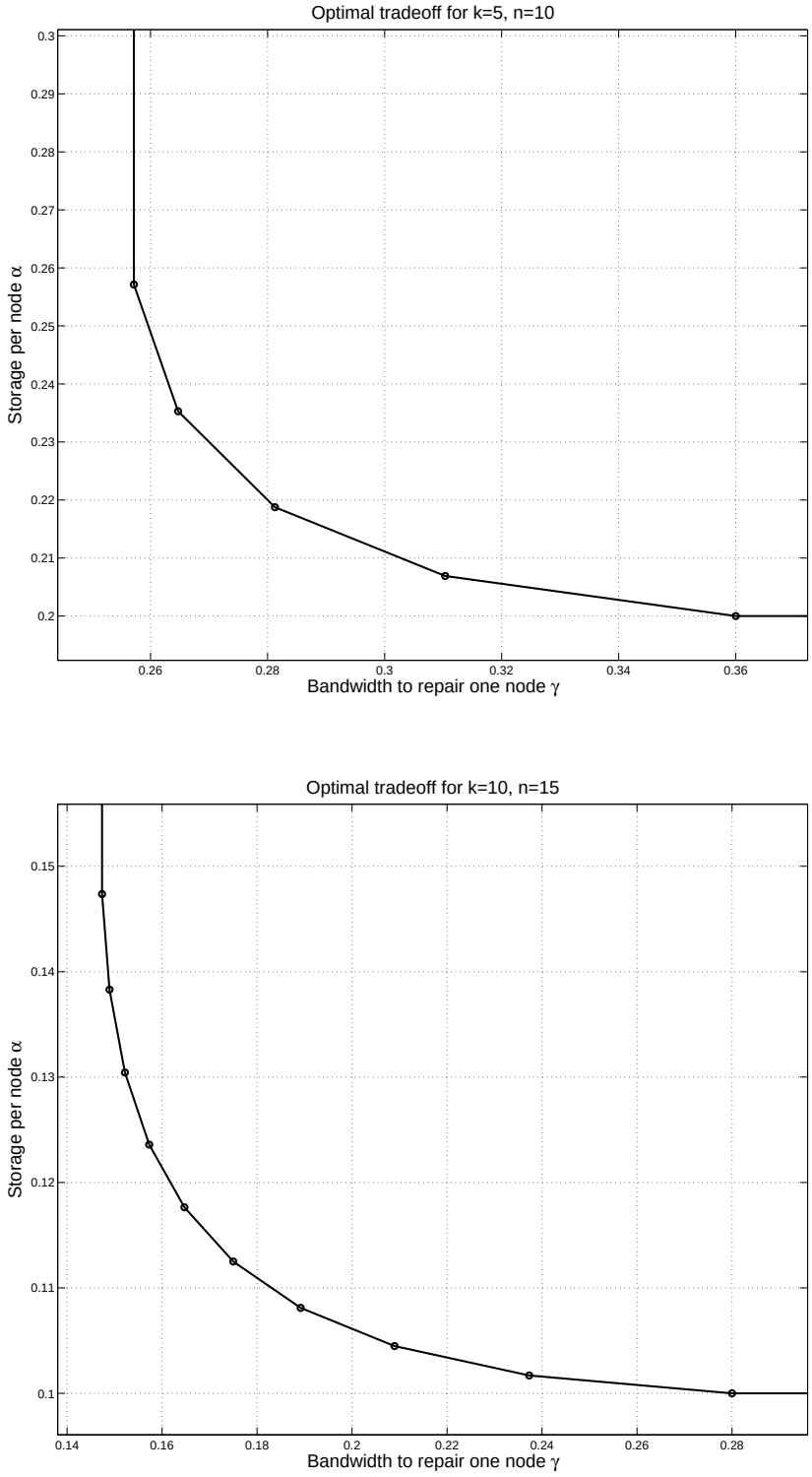


Fig. 4. Optimal tradeoff curve between storage α and repair bandwidth γ, for $k = 5, n = 10$ (left) and $k = 10, n = 15$ (right). For both plots $\mathcal{M} = 1$ and $d = n - 1$. Note that traditional erasure coding corresponds to the points $(\gamma = 1, \alpha = 0.2)$ and $(\gamma = 1, \alpha = 0.1)$ for the top and bottom plots.

From Theorem 1, it can be verified that the minimum storage point is achieved by the pair

$$(\alpha_{MSR}, \gamma_{MSR}) = \left(\frac{\mathcal{M}}{k}, \quad \frac{\mathcal{M}d}{k(d-k+1)} \right). \tag{5}$$

As discussed, the repair bandwidth $\gamma_{MSR} = d\beta_{MSR}$ is a decreasing function of the number of nodes d that participate in the repair. Since the MSR codes store $\frac{\mathcal{M}}{k}$ bits at each node while ensuring any k coded blocks can be used to recover the original file, they are equivalent with standard Maximum Distance Separable (MDS) codes. If we substitute $d = k$ into the above formula, we observe that the total communication for repair is $\mathcal{M}$, the size of the original file. Therefore, if we only allow a newcomer to contact k nodes, it is inevitable to communicate the whole data object to repair one new fragment and this is the naive repair method that can be performed to any MDS code.

However, if it is possible to for a newcomer to contact more than k nodes MSR codes can reduce the repair communication γ_{MSR}, which is minimized for $d = n - 1$:

$$(\alpha_{MSR}, \gamma^{min}_{MSR}) = \left(\frac{\mathcal{M}}{k}, \quad \frac{\mathcal{M}}{k} \cdot \frac{n-1}{n-k} \right). \tag{6}$$

We separated the $\mathcal{M}/k$ factor in γ^{min}_{MSR} to illustrate that MSR codes communicate an $\frac{n-1}{n-k}$ factor more than what they store, a fundamental expansion necessary for MDS constructions that are optimal on the reliability-redundancy tradeoff.

As a numerical example, for $(n, k) = (14, 7)$, the newcomer needs to communicate only $\frac{\mathcal{M}}{49}$ bits from each of the $d = n - 1 = 13$ active storage nodes, making the repair bandwidth equal to $\frac{13\mathcal{M}}{49}$, required to generate a fragment of size $\frac{\mathcal{M}}{7}$.

At the other end of the tradeoff are MBR codes, which have minimum repair bandwidth. It can be verified that the minimum repair bandwidth point is achieved by

$$(\alpha_{MBR}, \gamma_{MBR}) = \left(\frac{2\mathcal{M}d}{2kd - k^2 + k}, \quad \frac{2\mathcal{M}d}{2kd - k^2 + k} \right). \tag{7}$$

Note that the minimum bandwidth regenerating codes, the storage size α is equal to γ, the total number of bits communicated during repair. If we set the optimal value $d = n-1$, we obtain

$$(\alpha^{min}_{MBR}, \gamma^{min}_{MBR}) = \left(\frac{\mathcal{M}}{k} \cdot \frac{2n-2}{2n-k-1}, \quad \frac{\mathcal{M}}{k} \cdot \frac{2n-2}{2n-k-1} \right). \tag{8}$$

Therefore MBR codes incur no repair bandwidth expansion at all, just like a replication system does, communicating exactly the amount of information stored during a repair. However MBR codes require an expansion factor $\frac{2n-2}{2n-k-1}$ in the amount of stored information and are no longer optimal in terms of their reliability for the given redundancy.

IV. Evaluation

In this section, we compare regenerating codes with other redundancy management schemes in the context of distributed storage systems. We follow the evaluation methodology of [10], which consists of a simple analytical model whose parameters are obtained from traces of node availability measured in several real distributed systems.

We begin in Section IV-A with a discussion of node dynamics and the objectives relevant to distributed storage systems, namely reliability, bandwidth, and disk space. We introduce the model in Section IV-B and estimate realistic values for its parameters in Section IV-C. Section IV-D contains the quantitative results of our evaluation. In Section IV-E, we discuss qualitative tradeoffs between regenerating codes and other strategies, and how our results change the conclusion of [10] that erasure codes provide limited practical benefit.

A. Node dynamics and objectives

In this section we introduce some background and terminology which is common to most of the work discussed in Section II-C.

We draw a distinction between *permanent* and *transient* node failures. A permanent failure, such as the permanent departure of a node from the system or a disk failure, results in loss of the data stored on the node. In contrast, data is preserved across a transient failure, such as a reboot or temporary network disconnection. We say that a node is *available* when its data can be retrieved across the network.

Distributed storage systems attempt to provide two types of reliability: availability and durability. A file is *available* when it can be reconstructed from the data stored on currently available nodes. A file's *durability* is maintained if it has not been lost due to permanent node failures: that is, it may be available at some point in the future. Both properties are desirable, but in this paper we report results for availability only. Specifically, we will show *file unavailability*, the fraction of time that the file is not available.

B. Model

We use a model which is intended to capture the average-case bandwidth used to maintain a file in the system, and the resulting average availability of the file. With minor exceptions,[1]this model and the subsequent estimation of its parameters are equivalent to that of [10]. Although this evaluation methodology is a significant simplification of real storage systems, it allows us to compare directly with the conclusions of [10] as well as to calculate precise values for rare events.

The model has two key parameters, f and a. First, we assume that in expectation a fraction f of the nodes storing file data fail permanently per unit time, causing data transfers to repair the lost redundancy. Second, we assume that at any given time while a node is storing data, the node is available with some probability a (and with probability $1-a$ is currently experiencing a transient failure). Moreover, the model assumes that the event that a node is available is independent of the availability of all other nodes.

Under these assumptions, we can compute the expected availability and maintenance bandwidth of various redundancy schemes to maintain a file of $\mathcal{M}$ bytes. We make use of the

[1]In addition to evaluating a larger set of strategies and using a somewhat different set of traces, we count bandwidth cost due to permanent node failure only, rather than both failures and joins. Most designs [4], [33] can avoid reacting to node joins. Additionally, we compute probabilities directly rather than using approximations to the binomial.

fact that for all schemes except MSR codes, the amount of bandwidth used is equal to the amount of redundancy that had to be replaced, which is in expectation f times the amount of storage used.

Replication: If we store $\mathcal{R}$ replicas of the file, then we store a total of $\mathcal{R} \cdot \mathcal{M}$ bytes, and in expectation we must replace $f \cdot \mathcal{R} \cdot \mathcal{M}$ bytes per unit time. The file is unavailable if no replica is available, which happens with probability $(1-a)^{\mathcal{R}}$.

Ideal Erasure Codes: For comparison, we show the bandwidth and availability of a hypothetical (n, k) erasure code strategy which can "magically" create a new packet while transferring just $\mathcal{M}/k$ bytes (*i.e.*, the size of the packet). Setting $n = k \cdot \mathcal{R}$, this strategy sends $f \cdot \mathcal{R} \cdot \mathcal{M}$ bytes per unit time and has unavailability probability $U_{\text{ideal}}(n, k) := \sum_{i=0}^{k-1} \binom{n}{i} a^i (1-a)^{n-i}$.

Hybrid: If we store one full replica plus an (n, k) erasure code where $n = k \cdot (\mathcal{R} - 1)$, then we again store $\mathcal{R} \cdot \mathcal{M}$ bytes in total so we transfer $f \cdot \mathcal{R} \cdot \mathcal{M}$ bytes per unit time in expectation. The file is unavailable if the replica is unavailable *and* fewer than k erasure-coded packets are available, which happens with probability $(1-a) \cdot U_{\text{ideal}}(n, k)$.

Minimum-Storage Regenerating Codes: An (n, k) MSR Code with redundancy $\mathcal{R} = n/k$ stores $\mathcal{R}\mathcal{M}$ bytes in total, so $f \cdot \mathcal{R} \cdot \mathcal{M}$ bytes must be replaced per unit time. As mentioned, we refer to the *overhead* of an MSR code δ_{MSR} as the extra amount of information that needs to be transfered compared to the fragment size $\mathcal{M}/k$:

$$\delta_{MSR} \triangleq \frac{(n-1)\beta_{MSR}}{\mathcal{M}/k} = \frac{n-1}{n-k}. \tag{9}$$

Therefore, replacing a fragment requires transferring over the network δ_{MSR} times the size of the fragment in the most favorable case when newcomers connect to $d = n - 1$ nodes to construct a new fragment. Therefore, this results in $f \cdot \mathcal{R} \cdot \mathcal{M} \cdot \delta_{\text{MSR}}$ bytes sent per unit time, and unavailability $U_{\text{ideal}}(n, k)$.

Minimum-Bandwidth Regenerating Codes: Similarly, it is convenient to define the MBR code overhead as the amount of information transfered over the ideal fragment size:

$$\delta_{MBR} \triangleq \frac{(n-1)\beta_{MBR}}{\mathcal{M}/k} = \frac{2(n-1)}{2n-k-1}. \tag{10}$$

Therefore, an (n, k) MBR Code stores $\mathcal{M} \cdot n \cdot \delta_{\text{MBR}}$ bytes in total. So in expectation $f \cdot \mathcal{M} \cdot n \cdot \delta_{\text{MBR}}$ bytes are transfered per unit time, and the unavailability is again $U_{\text{ideal}}(n, k)$.

C. Estimating f and a

In this section we describe how we estimate f, the fraction of nodes that permanently fail per unit time, and a, the mean node availability, based on traces of node availability in several distributed systems.

We use four traces of node availability with widely varying characteristics, summarized in Table I. The **PlanetLab All Pairs Ping [34]** trace is based on pings sent every 15 minutes between all pairs of 200-400 nodes in PlanetLab, a stable, managed network research testbed. We consider a node to be up in one 15-minute interval when at least half of the pings sent to it in that interval succeeded. In a number of periods, all or nearly all PlanetLab nodes were down, most likely due to planned system upgrades or measurement errors. To exclude these cases, we "cleaned" the trace as follows: for each period of downtime at a particular node, we remove that period (i.e. we consider the node up during that interval) when the average number of nodes up during that period is less than half the average number of nodes up over all time. The **Microsoft PCs [35]** trace is derived from hourly pings to desktop PCs within Microsoft Corporation. The **Skype superpeers [36]** trace is based on application-level pings at 30-minute intervals to nodes in the Skype superpeer network, which may approximate the behavior of a set of well-provisioned endhosts, since superpeers may be selected in part based on bandwidth availability [36]. Finally, the trace of **Gnutella peers [37]** is based on application-level pings to ordinary Gnutella peers at 7-minute intervals.

We next describe how we derive f and a from these traces. It is of key importance for the storage system to distinguish between permanent and transient failures (defined in Section IV-A), since only the former requires bandwidth-intensive replacement of lost redundancy. Most systems use a *timeout* heuristic: when a node has not responded to network-level probes after some period of time t, it is considered to have failed permanently. To approximate a storage system's behavior, we use the same heuristic. Node availability a is then calculated as the mean (over time) fraction of nodes which were available among those which were not considered permanently failed at that time.

The resulting values of f and a appear in Table I, where we have fixed the timeout t at 1 day. Longer timeouts reduce overall bandwidth costs [10], [33], but begin to impact durability [33] and are more likely to produce artificial effects in the short (2.5-day) Gnutella trace.

We emphasize that the procedure described above only provides an estimate of f and a which may be biased in several ways. Some designs [33] reincorporate data on nodes which return after transient failures which were longer than the timeout t, which would reduce f. Additionally, even placing files on uniform-random nodes results in selecting nodes that are more available [31] and less prone to failure [32] than the average node. Finally, we have not accounted for the time needed to transfer data onto a node, during which it is effectively unavailable. However, we consider it unlikely that these biases would impact our main results since we are primarily concerned with the *relative* performance of the strategies we compare.

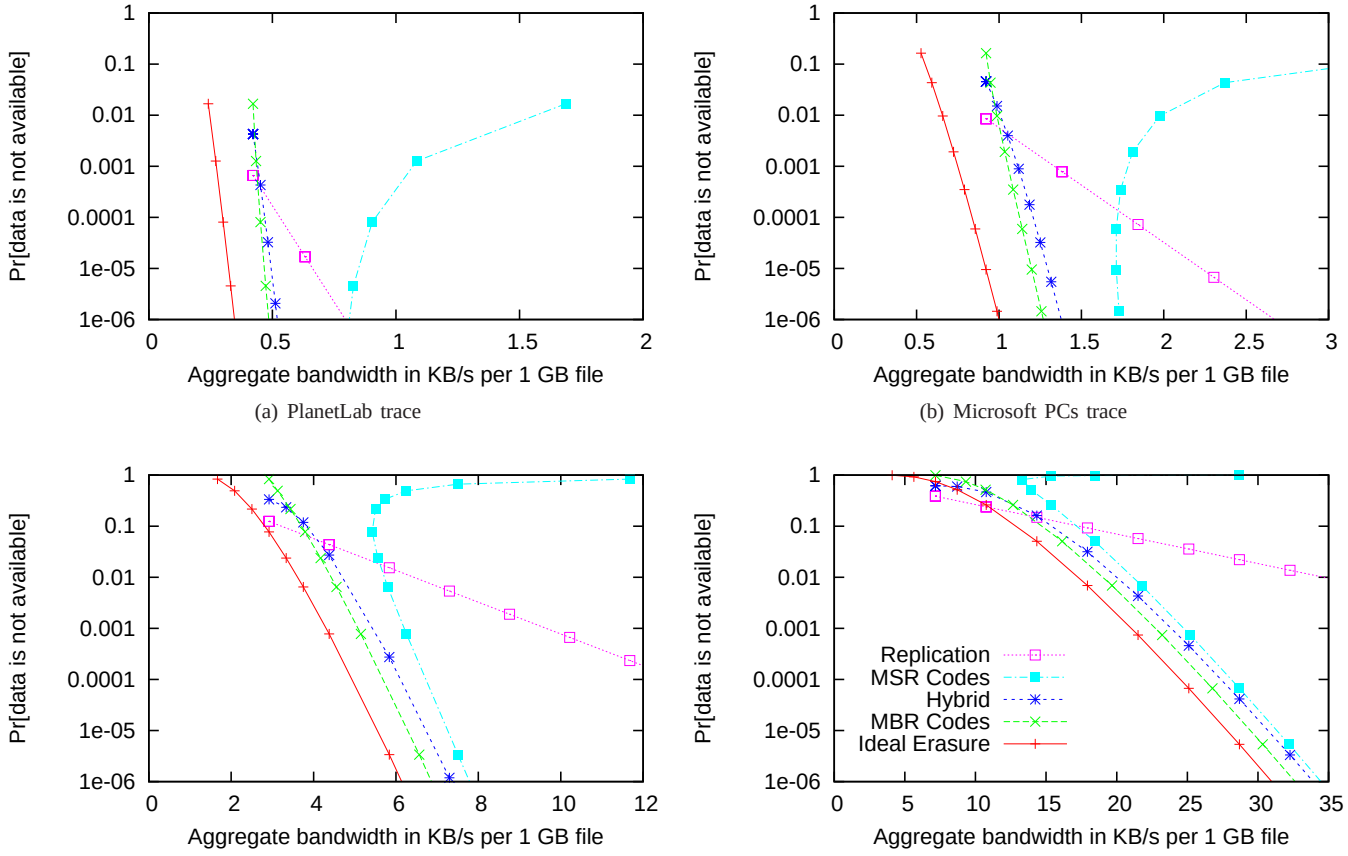
D. Quantitative results

Figure 5 shows the tradeoff between mean unavailability and mean maintenance bandwidth in each of the strategies of Section IV-B using the values of f and a from Section IV-C and $k = 7$. Feasible points in the tradeoff space are produced by varying the redundancy factor $\mathcal{R}$. The marked points along each curve highlight a subset of the feasible points (i.e., points for which n is integral).

Figure 6 shows that relative performance of the various strategies is similar for $k = 14$.

Trace	Length (days)	Start date	Mean # nodes up	f (fraction failed per day)	a
PlanetLab	527	Jan. 2004	303	0.017	0.97
Microsoft PCs	35	Jul. 6, 1999	41970	0.038	0.91
Skype	25	Sept. 12, 2005	710	0.12	0.65
Gnutella	2.5	May, 2001	1846	0.30	0.38

TABLE I
THE AVAILABILITY TRACES USED IN THIS PAPER.



(a) PlanetLab trace

(b) Microsoft PCs trace

(c) Skype superpeers trace

(d) Gnutella peers trace

Fig. 5. Availability-bandwidth tradeoff for $k = 7$ with parameters derived from each of the traces. The key in (d) applies to all four plots.

For conciseness, we omit plots of storage used by the schemes. However, disk usage is proportional to bandwidth for all schemes we evaluate in this section, with the exception of minimum storage regenerating codes. This is because MSR codes are the only scheme in which the data transferred onto a newcomer is not equal to the amount of data that the newcomer finally stores. Instead, the storage used by MSR codes is equal to that of the storage used by hypothetical ideal erasure codes, and hence MSR codes' space usage is proportional to the bandwidth used by ideal codes.

For example, from Figure 5(b) we can compare the strategies at their feasible points closest to unavailability 0.0001, i.e., four nines of availability. At these points, MSR codes use about 44% more bandwidth and 28% less storage space than Hybrid, while MBR codes use about 3.7% less bandwidth and storage space than Hybrid. Additionally, these feasible points give MSR and MBR codes somewhat better unavailability than Hybrid ($.000059$ vs. 0.00018).

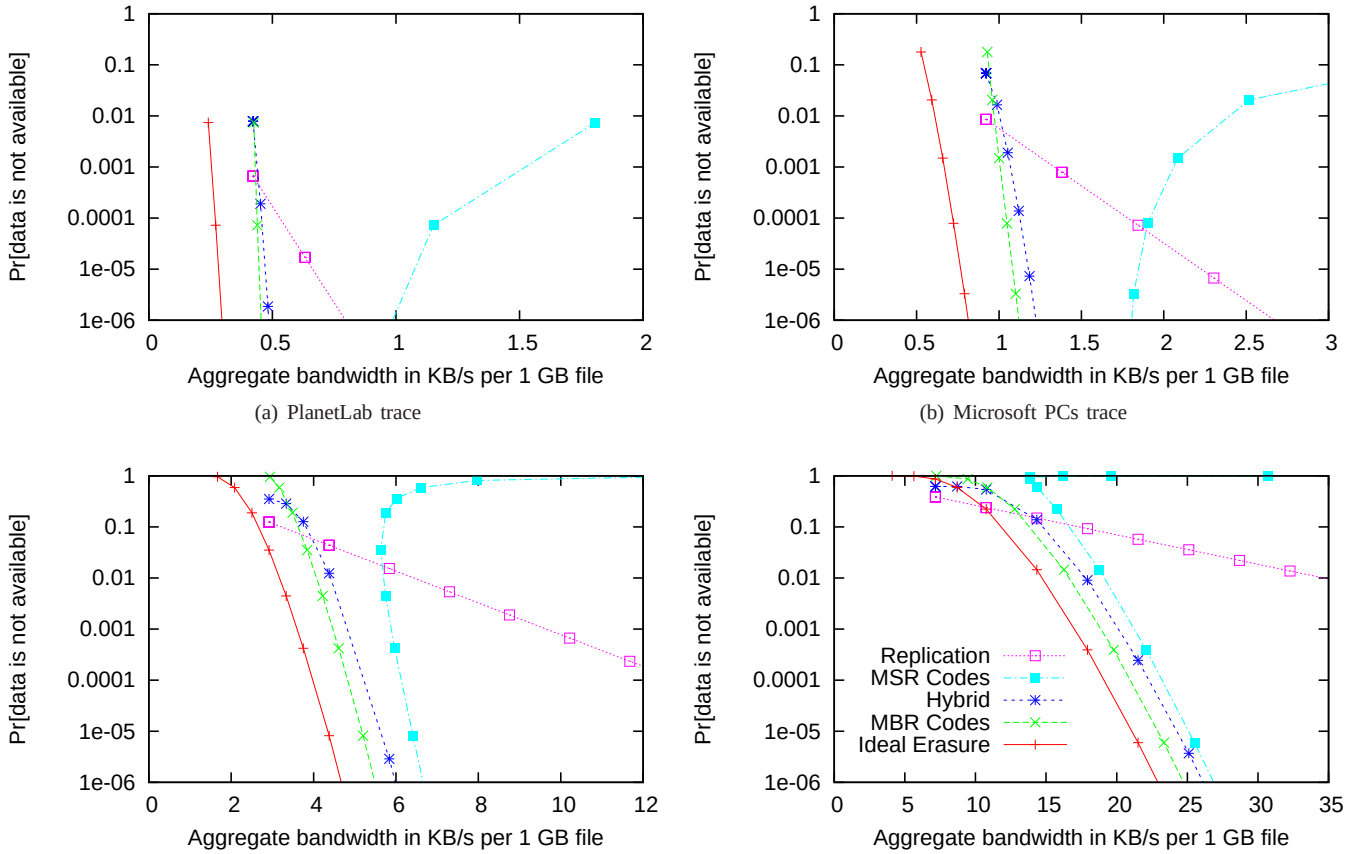
One interesting effect apparent in the plots is that MSR codes' maintenance bandwidth actually *decreases* as the redundancy factor $\mathcal{R}$ increases, before coming to a minimum and then increasing again. Intuitively, while increasing $\mathcal{R}$ increases the total amount of data that needs to be maintained, for small $\mathcal{R}$ this is more than compensated for by the reduction in overhead. The expected maintenance bandwidth per unit time is

$$f\,\mathcal{M}\,\mathcal{R}\,\delta_{\text{MSR}} = f\,\mathcal{M}\frac{n}{k}\frac{n-1}{n-k}. \tag{11}$$

It is easy to see that this function is minimized by selecting n one of the two integers closest to

$$n_{opt} = k + \sqrt{k^2 - k}. \tag{12}$$

which approaches a redundancy factor of 2 as $k \to \infty$.



(a) PlanetLab trace

(b) Microsoft PCs trace

(c) Skype superpeers trace

(d) Gnutella peers trace

Fig. 6. Availability-bandwidth tradeoff for $k = 14$ with parameters derived from each of the traces.

E. Qualitative comparison

In this section we discuss two questions: First, based on the results of the previous section, what are the qualitative advantages and disadvantages of the two extremal regenerating codes compared with the Hybrid coding scheme? Second, do our results affect the conclusion of Rodrigues and Liskov [10] that erasure codes offer too little improvement in bandwidth use to clearly offset the added complexity that they add to the system?

1) Comparison with Hybrid: Compared with Hybrid, for a given target availability, minimum bandwidth regenerating codes offer slightly lower maintenance bandwidth and storage, and a simpler system architecture since only one type of redundancy needs to be maintained. An important practical disadvantage of using the Hybrid scheme is asymmetric design which can cause the disk I/O to become the bottleneck of the system during repairs. This is because the disc storing the full replica and generates the encoded fragments need to read the whole data object and compute the encoded fragment.

However, MBR codes have at least two disadvantages. First, constructing a new packet, or reconstructing the entire file, requires communcation with $n-1$ nodes[2] rather than one (in Hybrid, the node holding the single replica). This adds overhead that could be significant for sufficiently small files or sufficiently large n. Perhaps more importantly, there is a factor δ_{MBR} increase in total data transferred to *read* the file, roughly 30% for a redundancy factor $\mathcal{R} = 2$ and $k = 7$ or 13% for $\mathcal{R} = 4$, Thus, if the frequency that a file is read is sufficiently high and k is sufficiently small, this inefficiency could become unacceptable. Again compared with Hybrid, MSR codes offer a simpler, symmetric system design and somewhat lower storage space for the same reliability. However, MSR codes have somewhat higher maintenance bandwidth and like MSB codes require that newcomers and data collectors connect to multiple nodes.

Rodrigues et al. [10] discussed two principal disadvantages of using erasure codes in a widely distributed system: coding—in particular, the Hybrid strategy—complicates the system architecture; and the improvement in maintenance bandwidth was minimal in more stable environments, which are the more likely deployment scenario. Regenerating codes address the first of these issues, which may make coding more broadly applicable.

[2]The scheme could be adapted to connect to fewer than $n-1$ nodes, but this would increase maintenance bandwidth.

V. Conclusions

We presented a general theoretic framework that can determine the information that must be communicated to repair failures in encoded systems and identified a tradeoff between storage and repair bandwidth.

Certainly there are many issues that remain to be addressed before these ideas can be implemented in practical systems. In future work we plan to investigate deterministic designs of regenerating codes over small finite fields, the existence of *systematic* regenerating codes, designs that minimize the overhead storage of the coefficients, as well as the impact of node dynamics in reliability. Other issues of interest involve how CPU processing and disk I/O will influence the system performance, as well as integrity and security for the linear combination packets (see [38] for a related analysis for content distribution). An online bibliography of recent papers and preprints that relate to regenerating codes and other problems on coding for distributed storage can be found in [39].

One potential application area for the proposed regenerating codes is distributed archival storage or backup, where files are typically large and infrequently read. In such scenarios, the regenerating codes potentially can offer desirable tradeoffs in terms of redundancy, reliability, and repair bandwidth.

REFERENCES

[1] A. G. Dimakis, P. G. Godfrey, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *IEEE Proc. INFOCOM*, (Anchorage, Alaska), May 2007.

[2] Y. Wu, A. G. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *Allerton Conference on Control, Computing, and Communication*, (Urbana-Champaign, IL), September 2007.

[3] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global data storage," *IEEE Internet Computing*, pp. 40–49, September 2001.

[4] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *Proc. of the Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[5] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proc. of the Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[6] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized erasure codes for distributed networked storage," in *Joint special issue, IEEE Trans. on Info. Theory and IEEE/ACM Trans. on Networking*, June 2006.

[7] A. Kamra, J. Feldman, V. Misra, and D. Rubenstein, "Growth codes: Maximizing sensor network data persistence," *ACM SIGCOMM*, 2006.

[8] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: the OceanStore prototype," in *Proc. USENIX File and Storage Technologies (FAST)*, 2003.

[9] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: a quantitiative comparison," in *Proc. IPTPS*, 2002.

[10] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," in *Proc. IPTPS*, 2005.

[11] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, "Improved low-density parity check codes using irregular graphs," *IEEE Trans. Info. Theory*, vol. 47, pp. 585–598, February 2001.

[12] M. Luby, "LT codes," *Proc. IEEE Foundations of Computer Science (FOCS)*, 2002.

[13] A. Shokrollahi, "Raptor codes," *IEEE Trans. on Information Theory*, vol. 52, pp. 2551–2567, June 2006.

[14] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures," *IEEE Trans. on Computing*, vol. 44, pp. 192–202, February 1995.

[15] L. Xu and J. Bruck, "X-code: MDS array codes with optimal encoding," *IEEE Trans. on Information Theory*, vol. 45, pp. 272–276, January 1999.

[16] C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," in *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, (San Francisco, CA), December 2005.

[17] J. L. Hafner, "WEAVER codes: Highly fault tolerant erasure codes for storage systems," in *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, (San Francisco, CA), December 2005.

[18] J. S. Plank, "Erasure codes for storage applications," in *Tutorial, FAST-2005: 4th Usenix Conference on File and Storage Technologies*, (San Francisco, CA), December 2005. [online] http://www.cs.utk.edu/ plank/plank/papers/FAST-2005.html.

[19] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Info. Theory*, vol. 46, pp. 1204–1216, July 2000.

[20] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. on Information Theory*, vol. 49, pp. 371–381, February 2003.

[21] R. Koetter and M. Médard, "An algebraic approach to network coding," in *IEEE/ACM Transactions on Networking*, October 2003.

[22] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Inform. Theory*, vol. 52, pp. 4413–4430, Oct. 2006.

[23] P. Sander, S. Egner, and L. Tolhuizen, "Polynomial time algorithms for network information flow," in *Symposium on Parallel Algorithms and Architectures (SPAA)*, (San Diego, CA), pp. 286–294, ACM, June 2003.

[24] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, "Polynomial time algorithms for network code construction," *IEEE Trans. Inform. Theory*, vol. 51, pp. 1973–1982, June 2005.

[25] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous Acess to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes," in *Proc. IEEE/ACM Int. Symposium on Information Processing in Sensor Networks (IPSN)*, April 2005.

[26] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," in *IEEE International Symposium on Network Computing and Applications (NCA 2007)*, July 2007.

[27] D. Wang, Q. Zhang, and J. Liu, "Partial network coding: Theory and application for continuous sensor data collection," *Fourteenth IEEE International Workshop on Quality of Service (IWQoS)*, 2006.

[28] A. Jiang, "Network coding for joint storage and transmission with minimum cost," in *International Symposium on Information Theory (ISIT)*, July 2006.

[29] F. Dabek, F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *Proc. ACM SOSP*, 2001.

[30] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," in *Proc. ACM SOSP*, 2001.

[31] K. Tati and G. M. Voelker, "On object maintenance in peer-to-peer systems," in *Proc. IPTPS*, 2006.

[32] P. B. Godfrey, S. Shenker, and I. Stoica, "Minimizing churn in distributed systems," in *Proc. ACM SIGCOMM*, 2006.

[33] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient replica maintenance for distributed storage systems," in *Proc. of the Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.

[34] J. Stribling, "Planetlab all pairs ping." http://infospect.planetlab.org/pings.

[35] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs," in *Proc. SIGMETRICS*, 2000.

[36] S. Guha, N. Daswani, and R. Jain, "An experimental study of the Skype peer-to-peer VoIP system," in *IPTPS*, 2006.

[37] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Proc. MMCN*, (San Jose, CA, USA), January 2002.

[38] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a P2P content distribution system with network coding," *Proceedings of IPTPS*, 2006.

[39] "Online wiki bibliography for distributed storage papers," `http://csi.usc.edu/~dimakis/StorageWiki`.

[40] J. Bang-Jensen and G. Gutin, "Digraphs: Theory, algorithms and applications," (New York), Springer, 2001.

VI. APPENDIX

Here we prove Theorem 1. We first start with the following simple lemma.

Lemma 1: No data collector DC can reconstruct the initial data object if the minimum cut in $\mathcal{G}$ between S and DC is smaller than the initial object size $\mathcal{M}$.

Proof: The information of the initial data object must be communicated from the source to the particular data collector. Since every link in the information flow graph can only be

used at most once, and since the point-to-point capacity is less than the data object size, a standard cut-set bound shows that the entropy of the data object conditioned on everything observable to the data collector is non-zero and therefore reconstruction is impossible. ■

The information flow graph casts the original storage problem as a network communication problem where the source s multicasts the file to the set of all possible data collectors. By analyzing the connectivity in the information flow graph, we obtain necessary conditions for all possible storage codes, as shown in Lemma 1. In addition to providing necessary conditions for all codes, the information flow graph can also imply the existence of codes under proper assumptions.

Proposition 1: Consider any given finite information flow graph $\mathcal{G}$, with a finite set of data collectors. If the minimum of the min-cuts separating the source with each data collector is larger or equal to the data object size $\mathcal{M}$, then there exists a linear network code defined over a sufficiently large finite field $\mathbb{F}$ (whose size depends on the graph size) such that all data collectors can recover the data object. Further, randomized network coding guarantees that all collectors can recover the data object with probability that can be driven arbitrarily high by increasing the field size.

Proof: The key point is observing that the reconstruction problem reduces exactly to multicasting on all the possible data collectors on the information flow graph $\mathcal{G}$. Therefore, the result follows directly from the constructive results in network coding theory for single source multicasting; see the discussion of related works on network coding in Section II-B. ■

To apply Proposition 1, consider an information flow graph $\mathcal{G}$ that enumerates all possible failure/repair patterns and all possible data collectors when the number of failures/repairs is bounded. This implies that there exists a valid regenerating code achieving the necessary cut bound (cf. Lemma 1), which can tolerate a bounded number of failures/repairs. In another paper [2], we present coding methods that construct deterministic regenerating codes that can tolerate infinite number of failures/repairs, with a bounded field size, assuming only the population of active nodes at any time is bounded. For the detailed coding theoretic construction, please refer to [2].

We analyze the connectivity in the information flow graph to find the minimum repair bandwidth. The next key lemma characterizes the flow in any information flow graph, under arbitrary failure pattern and connectivity.

Lemma 2: Consider any (potentially infinite) information flow graph G, formed by having n initial nodes that connect directly to the source and obtain α bits, while additional nodes join the graph by connecting to d existing nodes and obtaining β bits from each.[3] Any data collector t that connects to a k-subset of "out-nodes" (c.f. Figure 3) of G must satisfy:

$$\text{mincut}(s,t) \geq \sum_{i=0}^{\min\{d,k\}-1} \min\{(d-i)\beta, \alpha\}. \qquad (13)$$

[3]Note that this setup allows more graphs than those in $\mathcal{G}(n,k,d,\alpha,\beta)$. In a graph in $\mathcal{G}(n,k,d,\alpha,\beta)$, at any time there are n active storage nodes and a newcomer can only connect to the active nodes. In contrast, in a graph G described in this lemma, there is no notion of "active nodes" and a newcomer can connect to any d existing nodes.

Furthermore, there exists an information flow graph $G^* \in \mathcal{G}(n,k,d,\alpha,\beta)$ where this bound is matched with equality.

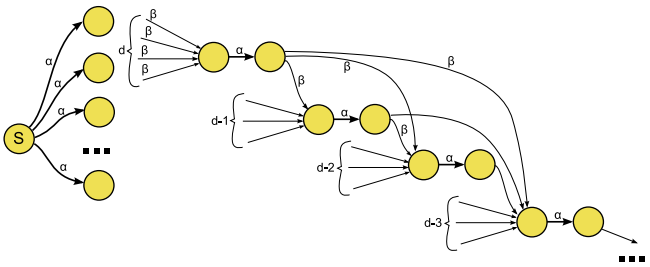


Fig. 7. G^* used in the proof of lemma 2

Proof: First, we show that there exists an information flow graph G^* where the bound (13) is matched with equality. This graph is illustrated by Figure 7. In this graph, there are initially n nodes labeled from 1 to n. Consider k newcomers labeled as $n+1,\ldots,n+k$. The newcomer node $n+i$ connects to nodes $n+i-d,\ldots,n+i-1$. Consider a data collector t that connects to the last k nodes, i.e., nodes $n+1,\ldots,n+k$. Consider a cut $(U,\overline{U})$ defined as follows. For each $i \in \{1,\ldots,k\}$, if $\alpha \leq (d-i)\beta$, then we include x_{out}^{n+i} in $\overline{U}$; otherwise, we include x_{out}^{n+i} and x_{in}^{n+i} in $\overline{U}$. Then this cut $(U,\overline{U})$ achieves (13) with equality.

We now show that (13) must be satisfied for any G formed by adding d in-degree nodes as described above. Consider a data collector t that connects to a k-subset of "out-nodes", say $\{\mathrm{x}_{out}^i : i \in I\}$. We want to show that any s–t cut in G has capacity at least

$$\sum_{i=0}^{\min\{d,k\}-1} \min\{(d-i)\beta, \alpha\}. \qquad (14)$$

Since the incoming edges of t all have infinite capacity, we only need to examine the cuts $(U,\overline{U})$ with $s \in U$,

$$\mathrm{x}_{out}^i \in \overline{U}, \forall i \in I. \qquad (15)$$

Let $\mathcal{C}$ denote the edges in the cut, i.e., the set of edges going from U to $\overline{U}$.

Every directed acyclic graph has a topological sorting (see, e.g., [40]), where a topological sorting (or acyclic ordering) is an ordering of its vertices such that the existence of an edge from v_i to v_j implies $i < j$. Let x_{out}^1 be the topologically first output node in $\overline{U}$. Consider two cases:

- If $\mathrm{x}_{in}^1 \in U$, then the edge $\mathrm{x}_{in}^1\mathrm{x}_{out}^1$ must be in $\mathcal{C}$.
- If $\mathrm{x}_{in}^1 \in \overline{U}$, since x_{in}^1 has an in-degree of d and it is the topologically first node in $\overline{U}$, all the incoming edges of x_{in}^1 must be in $\mathcal{C}$.

Therefore, these edges related to x_{out}^1 will contribute a value of $\min\{d\beta, \alpha\}$ to the cut capacity.

Now consider x_{out}^2, the topologically second output node in $\overline{U}$. Similar to the above, we have two cases:

- If $\mathrm{x}_{in}^2 \in U$, then the edge $\mathrm{x}_{in}^2\mathrm{x}_{out}^2$ must be in $\mathcal{C}$.
- If $\mathrm{x}_{in}^2 \in \overline{U}$, since at most one of the incoming edges of x_{in}^2 can be from x_{out}^1, $d-1$ incoming edges of x_{in}^1 must be in $\mathcal{C}$.

Following the same reasoning we find that for the i-th node ($i = 0, \ldots, \min\{d,k\}-1$) in the sorted set $\overline{U}$, either one edge of capacity α or $(d-i)$ edges of capacity β must be in $\mathcal{C}$. Equation (13) is exactly summing these contributions. ∎

From Lemma 2, we know that there exists a graph $G^* \in \mathcal{G}(n,k,d,\alpha,\beta)$ whose mincut is exactly

$$\sum_{i=0}^{\min\{d,k\}-1} \min\{(d-i)\beta, \alpha\}.$$

This implies that if we want to ensure recoverability while allowing a newcomer to connect to *any* set of d existing nodes, then the following is a necessary condition[4]

$$\sum_{i=0}^{\min\{d,k\}-1} \min\{(d-i)\beta, \alpha\} \geq \mathcal{M}. \tag{16}$$

Furthermore, when this condition is satisfied, we know any graph in $\mathcal{G}(n,k,d,\alpha,\beta)$ will have enough flow from the source to each data collector. For this reason, we say

$$\mathsf{C} \triangleq \sum_{i=0}^{\min\{d,k\}-1} \min\{(d-i)\beta, \alpha\} \tag{17}$$

is the *capacity* for (n,k,d,α,β) regenerating codes (where each newcomer can access any arbitrary set of k nodes).

Note that if $d < k$, requiring any d storage nodes to have a flow of $\mathcal{M}$ will lead to the same condition (c.f. (16)) as requiring any k storage nodes to have a flow of $\mathcal{M}$. Hence in such a case, we might as well set k as d. For this reason, in the following we assume $d \geq k$ without loss of generality.

We are interested in characterizing the achievable tradeoffs between the storage α and the repair bandwidth $d\beta$ for some given (n,k). To simplify our notation we drop the explicit dependency of α^* in n,k for this optimization. To derive the optimal tradeoffs, we can fix the repair bandwidth and solve for the minimum α such that (16) is satisfied. Recall that $\gamma = d\beta$ the total repair bandwidth, and the parameters (n,k,d,α,γ) can be used to characterize the system. We are interested in finding the whole region of feasible points (α,γ) and then select the one that minimizes storage α or repair bandwidth γ. Consider fixing both γ and d (to some integer value) and minimize α;

$$\begin{aligned} \alpha^*(d,\gamma) \triangleq \min \quad & \alpha \\ \text{subject to:} \quad & \sum_{i=0}^{k-1} \min\left\{\left(1-\frac{i}{d}\right)\gamma, \alpha\right\} \geq \mathcal{M}. \end{aligned} \tag{18}$$

Now observe that the dependence on d must be monotone:

$$\alpha^*(d+1,\gamma) \leq \alpha^*(d,\gamma). \tag{19}$$

This is because $\alpha^*(d,\gamma)$ is always a feasible solution for the optimization for $\alpha^*(d+1,\gamma)$. Hence a larger d always implies a better storage–repair bandwidth tradeoff.

[4]This, however, does not rule out the possibility that the mincut is larger if a newcomer can choose the d existing nodes to connect to. We leave this as a future work.

The optimization (18) can be explicitly solved: We call the solution, the threshold function $\alpha^*(d,\gamma)$, which for a fixed d, is piecewise linear:

$$\alpha^*(d,\gamma) = \begin{cases} \frac{\mathcal{M}}{k}, & \gamma \in [f(0), +\infty) \\ \frac{\mathcal{M}-g(i)\gamma}{k-i}, & \gamma \in [f(i), f(i-1)), \end{cases} \tag{20}$$

where

$$f(i) \triangleq \frac{2\mathcal{M}d}{(2k-i-1)i + 2k(d-k+1)}, \tag{21}$$

$$g(i) \triangleq \frac{(2d-2k+i+1)i}{2d}. \tag{22}$$

The last part of the proof involves showing that the threshold function is the solution of this optimization. To simplify notation, introduce

$$b_i \triangleq \left(1 - \frac{k-1-i}{d}\right)\gamma, \quad \text{for } i = 0, \ldots, k-1. \tag{23}$$

Then the problem is to minimize α subject to the constraint:

$$\sum_{i=0}^{k-1} \min\{b_i, \alpha\} \geq B. \tag{24}$$

The left hand side of (24), as a function of α, is a piecewise-linear function of α:

$$\mathsf{C}(\alpha) = \begin{cases} k\alpha, & \alpha \in [0, b_0] \\ b_0 + (k-1)\alpha, & \alpha \in (b_0, b_1] \\ \vdots & \vdots \\ b_0 + \ldots + b_{k-2} + \alpha, & \alpha \in (b_{k-2}, b_{k-1}] \\ b_0 + \ldots + b_{k-1}, & \alpha \in (b_{k-1}, \infty) \end{cases}. \tag{25}$$

Note from this expression that $\mathsf{C}(\alpha)$ is strictly increasing from 0 to its maximum value $b_0 + \ldots + b_{k-1}$ as α increases from 0 to b_{k-1}. To find the minimum α such that $\mathsf{C}(\alpha) \geq B$, we simply let $\alpha^* = \mathsf{C}^{-1}(B)$ if $B \leq b_0 + \ldots + b_{k-1}$:

$$\alpha^* = \begin{cases} \frac{B}{k}, & B \in [0, kb_0] \\ \frac{B-b_0}{k-1}, & B \in (kb_0, b_0 + (k-1)b_1] \\ \vdots & \vdots \\ B - \sum_{j=0}^{k-2} b_j, & B \in \left(\sum_{j=0}^{k-2} b_j + b_{k-2},\ \sum_{j=0}^{k-1} b_j\right] \end{cases} \tag{26}$$

For $i = 1, \ldots, k-1$, the i-th condition in the above expression is:

$$\alpha* = \frac{B - \sum_{j=0}^{i-1} b_j}{k-i},$$
$$\text{for } B \in \left(\sum_{j=0}^{i-1} b_j + (k-i)b_{i-1}, \sum_{j=0}^{i} b_j + (k-i-1)b_i\right],$$

Note from the definition of $\{b_i\}$ (23) that

$$\begin{aligned} \sum_{j=0}^{i-1} b_j &= \sum_{j=0}^{i-1} \left(1 - \frac{k-1-j}{d}\right)\gamma \\ &= \gamma\left[i\left(1 - \frac{k-1}{d}\right) + \frac{i(i-1)}{2d}\right] \\ &= \gamma i \frac{2d-2k+i+1}{2d}, \\ &= \gamma g(i), \end{aligned}$$

and

$$\begin{aligned}
&\sum_{j=0}^{i} b_j + (k-i-1)b_i \\
=&\gamma(i+1)\frac{2d-2k+i+2}{2d} + (k-i-1)\gamma\left(1-\frac{k-1-i}{d}\right) \\
=&\gamma\frac{2ik-i^2-i+2k+2kd-2k^2}{2d}, \\
=&\gamma\frac{B}{f(i)},
\end{aligned}$$

where $f(i)$ and $g(i)$ are defined in (2)(3). Hence we have:

$$\alpha* = \frac{B-g(i)}{k-i}, \quad \text{for } B \in \left(\frac{\gamma B}{f(i-1)}, \frac{\gamma B}{f(i)}\right].$$

The expression of $\alpha^*(d,\gamma)$ then follows. ■