Byzantine fault tolerance of regenerating codes

Oggier, Frederique; Datta, Anwitaman

2011

Oggier, F., & Datta, A. (2011). Byzantine Fault Tolerance of Regenerating Codes.
Proceedings of the 11th IEEE International Conference on Peer-to-Peer Computing, Tokyo.

https://hdl.handle.net/10356/93864

Byzantine Fault Tolerance of Regenerating Codes

Frédérique Oggier
Division of Mathematical Sciences
School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
Email: frederique@ntu.edu.sg

Anwitaman Datta
Division of Computer Science
School of Computer Engineering
Nanyang Technological University, Singapore
Email: anwitaman@ntu.edu.sg

***Abstract*—Recent years have witnessed a slew of coding techniques custom designed for networked storage systems. Network coding inspired regenerating codes are the most prolifically studied among these new age storage centric codes. A lot of effort has been invested in understanding the fundamental achievable trade-offs of storage and bandwidth usage to maintain redundancy in presence of different models of failures, showcasing the efficacy of regenerating codes with respect to traditional erasure coding techniques. For practical usability in open and adversarial environments, as is typical in peer-to-peer systems, we need however not only resilience against erasures, but also from (adversarial) errors. In this paper, we study the resilience of generalized regenerating codes (supporting multi-repairs, using collaboration among newcomers) in the presence of two classes of Byzantine nodes, relatively benign selfish (non-cooperating) nodes, as well as under more active, malicious polluting nodes. We give upper bounds on the resilience capacity of regenerating codes, and show that the advantages of collaborative repair can turn to be detrimental in the presence of Byzantine nodes. We further exhibit that system mechanisms can be combined with regenerating codes to mitigate the effect of rogue nodes.**

Keywords: distributed storage, regenerating codes, Byzantine faults, pollution, resilience

I. Introduction

Redundancy is essential for reliably storing data. This basic principle has been adhered in designing diverse storage solutions such as CDs and DVDs, RAID systems as well as, more recently - networked distributed storage systems. Such redundancy may be achieved by replicating the data, or applying coding based techniques. Coding based techniques incur much less storage overhead with respect to replication based technique in order to achieve equivalent resilience (fault-tolerance). Thus, coding based redundancy is often preferred for efficiently storing large amount of data.

In networked storage systems, which may be as diverse as peer-to-peer (P2P) storage systems or data centers, redundant data is distributed across multiple storage devices. When some of these devices become unavailable - be it due to failure or (permanent) churn, redundancy needs to be replenished, otherwise, over time, the system will lose the stored data. If replication based redundancy is used, a new replica is created by copying data from existing replica(s). When using coding based techniques, each storage node typically possesses a small (w.r.to the size of the original data being stored) amount of the data, that we will call an *encoded block*. Since the data can be recovered by contacting a fraction of the storage nodes, redundancy can be replenished in the same way: first reconstruct the whole data, re-encode it, and re-distribute the encoded blocks.

This is the case when using traditional *erasure codes* (EC) such as Reed-Solomon codes [19]. In order to replenish lost redundancy, data equivalent in volume to the complete object needs to be transferred (or stored at one node a priori), in order to recreate even a single encoded block. To improve on such a naive approach, network coding based coding [6] was proposed to recreate one new encoded block by transferring much less data, upto possibly equivalent volume of data to only as is to be recreated. This new family of codes is called *regenerating codes* [21] - and the strategy may be applied on the original data itself, or on top of erasure encoding. Two different types of works have emerged on regenerating codes: those which establish the theoretical feasibility of such bandwidth efficient redundancy replenishment through min-cut bounds (such as [21], or [20] for more general bounds), and those which instead try to provide various coding strategies to do so in practice.

The current regeneration code related literature mostly (but for [3] and [18] that we will discuss later on) assumes a friendly environment, where all live nodes are well behaved. In open environments, particularly P2P environments, one should make such an assumption at his own peril.

We note that erasure codes such as Reed-Solomon codes are resilient against not only 'erasures' but are also capable of dealing with 'errors'. In contrast, while regenerating codes inherit the advantages of network coding such as bandwidth efficiency, they also likewise suffer from the same vulnerabilities of network coding. One of the most critical issues which intrinsically affect network coding is the family of pollution attacks. The idea behind network coding is to allow any intermediate node in the network to forward linear combinations of its incoming packets to its neighbors, which when done cleverly and diligently, results in throughput gain. However, it also means that one bogus packet can corrupt several other packets downstream, and thus spread over and contaminate a

F. Oggier's research for this work has been supported by the Singapore National Research Foundation under Research Grant NRF-RF2009-07. A. Datta's research for this work has been supported in part by A*Star TSRP grant number 102 158 0038 on pCloud project and in part by NTU AcRF Tier-1 grant number RG 29/09 on CrowdStore project. The authors will also like to thank Nicolas Le Scouarnec for his advices to carry out the numerical optimizations.

large portion of the network. Such attacks are not possible in a classical routing scenario.

The same problem of pollution attack can be directly translated in the context of coding for distributed storage based on network coding, in particular in the case of regenerating codes. In this paper, we study if and how well regenerating codes may tolerate Byzantine nodes. We identify the cardinal Byzantine attacks possible during the regeneration process. Specifically, we look at the following families of Byzantine nodes:

- *Selfish (non-cooperating) nodes*: Nodes may not actively attack the network, however they may prioritize their own interests, and might just decline to cooperate during the regeneration process, that is, refuse to provide the data that is requested from them to carry out regeneration. In absence of the contribution from such selfish or non-cooperating nodes, a regeneration protocol designed assuming their contribution will fail to carry out the regeneration task anymore.
- *Polluters*: Nodes may try to disrupt the regeneration process actively, by deliberately sending wrong data. Such active attack is particularly detrimental while using regenerating codes, since it would affect future regeneration processes where a victim participates and continues to further spread the pollution unconsciously and unintentionally.

The main contributions of this work are as follows. (i) We determine bounds on the resilience capacity of regenerating codes, taking into account the above mentioned adversarial behaviors. (ii) Our analysis reveals that though collaboration in regeneration can be beneficial in terms of bandwidth and storage costs, the penalty in presence of Byzantine nodes is also substantially larger. There is a blowback effect, in that, collaboration may not only be useless under Byzantine attacks, but can in fact be detrimental, such that one would be better off by avoiding collaboration. (iii) Finally, we outline how this effect can also be easily mitigated in practice using some additional information and extrinsic mechanisms.

II. Regenerating codes in a nutshell

Consider an object of size B to be stored in a network with n storage nodes, a source S which has adequate bandwidth to upload data over the network to these nodes, and a data collector DC which should be able to retrieve a given stored object by accessing data from any arbitrary choice of k out of the n nodes. Thus to say, such a storage network stores the object redundantly, and can tolerate up to $n-k$ failures without affecting the object's availability. For instance, erasure codes may be used to encode the object and achieve such redundancy.

Over time, some of the storage nodes may go offline (or crash), and if the redundancy is not restored then the system's fault tolerance will reduce, leading to, in the worst case, eventual loss of the stored object. Thus, mechanisms are needed to repair or regenerate the lost redundancy. Naive solutions include keeping a full copy of the object somewhere, which can be used to recreate the lost data at any node. Alternatively, if no such full copy is available, then one can download adequate, i.e., k encoded data blocks, and use these to regenerate the lost encoded data blocks. These naive solutions are sub-optimal in terms of efficient use of storage space and bandwidth for regeneration respectively, and have in the recent years prompted the exploration of better solutions - such as (chronologically) Pyramid codes [10], Regenerating codes [21], Hierarchical codes [7] and Self-repairing codes [16] to name some of the most prominent ones. We next summarize some key results related to regenerating codes, since this paper studies their Byzantine fault tolerance.

Suppose that each node has a storage capacity of α, i.e., the size of the encoded data block stored at a node is of the size α. When one data block needs to be regenerated, a new node contacts d ($k \leq d$) other existing nodes, and downloads β amount of data from each of the contacted nodes (referred to as the bandwidth capacity of the connections between any node pair)[1]. By considering an information flow from the source to the data collector, a trade-off between the nodes' storage capacity and bandwidth can be computed [21], through a min-cut bound.

Proposition 1: [21] A min-cut bound of an information flow between the source and a data collector is

$$\text{mincut}(\mathrm{S},\mathrm{DC}) \geq \sum_{i=0}^{k-1} \min\{\alpha,(d-i)\beta\}.$$

Note that such a min-cut bound determines achievability - without necessarily stating any specific way to actually do so. Furthermore, it is required that

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

for regeneration to be possible. Two sub-families of regenerating codes have consequently emerged [22] - coined as *functional*, respectively *exact*, to provide actual coding strategies. Functional repair strategies rely on random network coding arguments, and while they regenerate lost redundancy, the data stored by new nodes is not 'bit-by-bit' identical to the encoded block that previously existed: it is enough that it allows the retrieval of the stored data. In contrast, exact repair leads to regeneration of bit-by-bit identical encoded block as was lost. Exact regeneration is preferable since it translates to simplicity in system design and management. A more detailed comparison between exact and functional repair can be found in [8].

The original bound reported in Proposition 1 was derived assuming that only one encoded data block for a single node is being regenerated. However, this is not a realistic assumption to build practical networked storage systems. In highly dynamic scenarios, which is typical in peer-to-peer environments, but also may happen in more static (data-center

[1]Note that, in contrast to conventional techniques which download the whole encoded data block, only a smaller β/α fraction of data from each contacted node is being transferred.

like) environments due to correlated failures, it may be necessary to regenerate data for multiple nodes. Naive strategies would include regenerations sequentially, or in parallel, but independently of each other.

In [11], the above framework has been extended for multiple new nodes to carry out regeneration by not only downloading data from (old) live nodes, but also by additionally collaborating among each other under some specific settings. A more generalized result is provided in [20] (and also, independently in [12]).

The regeneration process is carried out in two phases, a download phase during which a batch of t newcomers download data from any d live nodes each, and a collaborative phase, where each newcomer shares some of its data to help the $t-1$ other new nodes. Such a two phase regeneration involving collaboration among new nodes can lead to reduction in the overall bandwidth usage for the regenerations.

Under such a setting, a more general min-cut bound is derived. In the following, β' represents the bandwidth during the collaborative phase, i.e., each new node sends (and also receives) β' data to (from) each other new node. Consider that the data collector contacts k nodes for reconstructing the data, such that the contacted nodes can be arranged in g groups of sizes u_i where $u_0 + \ldots + u_{g-1} = k$, where each such group represents a generation of t nodes which had joined the system together and carried out the regeneration collaboratively.

Proposition 2: [12], [20] A min-cut bound of an information flow between the source and a data collector is

$$\text{mincut}(\text{S}, \text{DC}) \geq \sum_{i=0}^{g-1} u_i \min\{\alpha, (d - \sum_{j=0}^{i-1} u_j)\beta + (t-u_i)\beta'\}$$

where $k = \sum_{i=0}^{g-1} u_i$ with $1 \leq u_i \leq t$,
and as above, we need

$$\sum_{i=0}^{g-1} u_i \min\{\alpha, (d - \sum_{j=0}^{i-1} u_j)\beta + (t-u_i)\beta'\} \geq B$$

for regeneration to be possible. When $t = 1$, we get that $u_i = 1$, thus $g = k$ and the more general bound matches the one given in Proposition 1.

As pointed out in [12], two extreme cases can be identified. First, if there is no contribution in β', then the highest contribution comes from β, that is $u_i = t$ and $g = k/t$, and the min-cut bound becomes

$$\text{mincut}(\text{S}, \text{DC}) \geq \sum_{i=0}^{k/t-1} t \min\{\alpha, (d-it)\beta\}. \quad (1)$$

Conversely, the highest contribution from β' comes when β is minimized, which occurs when $u_i = 1$ for all i and $g = k$. Then the min-cut bound simplifies to

$$\text{mincut}(\text{S}, \text{DC}) \geq \sum_{i=0}^{k-1} \min\{\alpha, (d-i)\beta + (t-1)\beta'\}. \quad (2)$$

The minimum possible amount of data that can be stored at a node is B/k, since the data collector must be able to retrieve the object out of any k nodes. Codes using the lowest amount of storage $\alpha = B/k$ are said to satisfy the *minimum storage regeneration (MSR) point*, and using (1) and (2) are shown to be characterized by [12]

$$\alpha = \frac{B}{k}, \ \beta = \beta' = \frac{B}{k}\frac{1}{d-k+t} \quad (3)$$

while codes requiring the minimum bandwidth for regeneration similarly satisfy

$$\alpha = \frac{B}{k}\frac{2d+t-1}{2d-k+t} \quad (4)$$

and

$$\beta = \frac{B}{k}\frac{2}{2d-k+t}, \ \beta' = \frac{B}{k}\frac{1}{2d-k+t}, \quad (5)$$

a point called the *minimum bandwidth regeneration (MBR) point*.

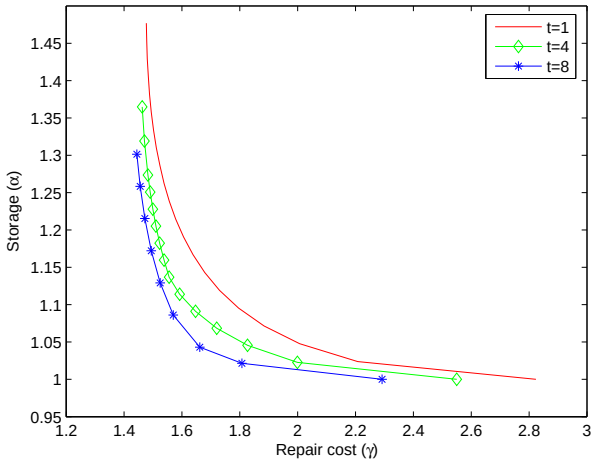


Fig. 1. The storage bandwidth (per repair) trade-off curve using regenerating codes with collaboration for $t = 1, 4, 8$. This plot (and all others plots in this paper) has been generated using linear non-convex optimization numerically. The values have been normalized by B/k.

The benefit of collaborative regenerating codes with respect to standard regenerating codes (that is, with no collaboration phase) is illustrated in Fig. 1, where we set $d = 48$ and $k = 32$. Trade-off curves between the storage cost α on the *y-axis* against the bandwidth cost per repair on the *x-axis*, determined in (6) and denoted as γ are shown for different scenarios. For collaborative regenerating codes, the total bandwidth for one node to be repaired is the data downloaded from live nodes, that is β from d nodes, and the data exchanged among newcomer nodes during collaboration, which is β' from $t-1$ nodes, for a total of

$$\gamma = d\beta + (t-1)\beta'. \quad (6)$$

If no collaboration is done, then $t = 1$ and $\gamma = d\beta$. The trade-off curve for $t = 1$ in Fig. 1 thus corresponds to standard (independent) regenerations. Larger value of t, implying multiple repairs being carried out collaboratively, allows the storage system to operate using both lower storage and bandwidth costs.

Though several works discussed min-cut bounds for collaborative regeneration codes, we are aware of only one family of collaborative regenerating codes [20], which provides exact repair for $d = k$ at MSR point. It is noted in [12] that for $d = k$, the repair cost is the same as for erasure correcting codes using delayed repair.

III. Byzantine faults model

The regeneration process can be dramatically affected if some of the live nodes behave in a Byzantine manner, that is, act in a manner different than as expected by the regeneration process. So far, and to the best of our knowledge, [18] is the only work looking at security issues related to regenerating codes. Besides considering a passive adversary who eavesdrops, it also looks at malicious behaviors affecting data integrity at nodes during the regeneration process, but all the considered scenarios assume a single regeneration at a time, rather than the more general problem of multiple simultaneous regenerations. This naturally excludes the complications arising due to the collaboration phase, where a single Byzantine node can potentially contaminate all the other regenerating nodes simultaneously.

In this paper, we consider two types of Byzantine adversaries. A relatively benign form of faulty behavior is when a live node does not provide any data for the regeneration process. We will refer to such Byzantine nodes as *selfish nodes*. Note that we distinguish a selfish node from an unavailable (offline) node in that a selfish node is expected to continue to respond to a data collector trying to recreate the object. If a node refuses to help for both regeneration and also data access, then it can be treated analogously as any other offline node. Such a selfish behavior may arise due to various reasons: the node may be overloaded with other tasks, or there may be temporary problems in the communication link - so that the node can not respond in a timely manner to meaningfully contribute to the regeneration process. No such time-bounded response is assumed for data reconstruction by a data collector. Alternatively, a node participating in a peer-to-peer back-up system may be comfortable with responding to data access requests which are relatively infrequent, and hence less taxing on its bandwidth resources, than regeneration process which could be frequent due to system churn, prompting the node to act selfishly for the regeneration process.

A more malign faulty behavior is when wrong data is sent by a node. Such a behavior even by a single node, if unchecked, may corrupt many nodes downstream. We will refer to such nodes as *polluting nodes*. Rapid propagation of pollution is an inherent and general weakness of network coding, on which rely regenerating codes, making the system extremely vulnerable in the presence of even one or very few polluting nodes.

We note that, for collaborative regeneration, the Byzantine nodes may be among the originally online nodes when the regeneration process is initiated; or among the newly joining nodes, i.e., during the collaboration phase; or a mix of both. Clearly, the amount of data that can be stored reliably and needs to be transferred during regeneration will change under these adversarial constraints, and in particular, so will the trade-off between the storage α and the bandwidth β, β' as described by the min-cut bounds in Propositions 1 & 2.

In the spirit of [18], we consider the resiliency capacity of the distributed storage system as the maximum amount of data that can be stored reliably over the network in the presence of malign nodes, and made available to a legitimate data collector. More precisely, we will focus on the resiliency capacity $C_{r,s}(\alpha, \beta, \beta')$ in the presence of selfish nodes, and $C_{r,p}(\alpha, \beta, \beta')$ when polluting nodes are active.

IV. Min-cut bounds under Byzantine failures

We will analyze how the storage bandwidth trade-off given in Proposition 2 is affected in presence of the various Byzantine nodes. We study the general case of regenerating codes, which studies multiple simultaneous regenerations, and with collaboration among the new nodes.

We determine upper-bounds, which means that it is not possible to do any better than the constraints of the corresponding bounds. Note that this is in contrast to the Propositions 1 & 2, which determined achievability, though both bounds are derived through min-cut computations. Since we derive upper bounds here, we can make simplifying (optimistic) assumptions, implying that, under more realistic assumptions and complicated derivations, it may be possible to determine tighter bounds.

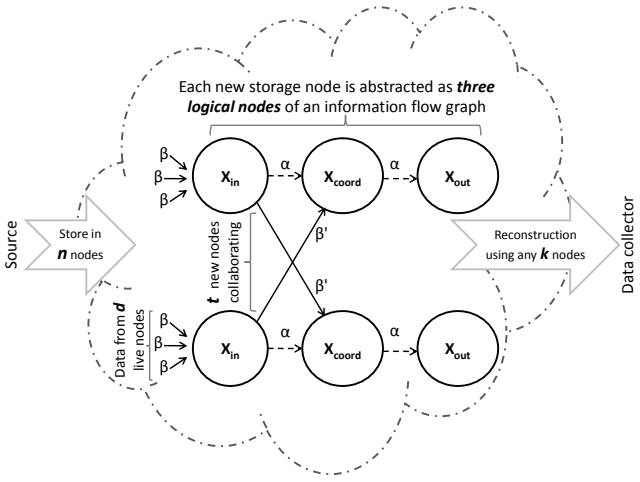


Fig. 2. An abstract information flow graph model for the coordinated regeneration process.

For determining a min-cut, we consider an information flow graph and use the same abstraction as in [12], which is illustrated in Fig. 2. Each new storage node is modeled using three logical nodes in an information flow graph connecting the source to the data collector, namely $x_{\rm in}$, $x_{\rm coord}$ and $x_{\rm out}$. It is assumed that t such new nodes carry out the regeneration in a collaborative manner. $x_{\rm in}$ represents the aggregation of information by a new node from d of the existing live nodes, collecting β data from each such contacted live nodes. In the next (collaborative) phase, each new node provides (and

also obtains) β' data from each of the other new nodes. This collected data is then processed at individual nodes, and finally they retain (store) α amount of data each. Thus to each node corresponds a triple $x_{\rm in} \to x_{\rm coord} \to x_{\rm out}$ where both edges $x_{\rm in} \to x_{\rm coord}$ and $x_{\rm coord} \to x_{\rm out}$ have a capacity of α. We will later (in Example 1, Section V) elaborate a concrete example of multiple regenerations with coordination.

A. Effect of selfish nodes

In the following we assume that the number of selfish nodes among the live (old) nodes is given by $\mathcal{L}_0$ in any generation, and $l_i \leq l^{max}$ is the number of selfish nodes among the ith group of new comers, for some upper bound l^{max}. The total number of selfish nodes participating in the collaborative phase of regeneration over g generations is $\mathcal{L} = \sum_{i=0}^{g-1} l_i$.

Proposition 3: The resiliency capacity $C_{r,s}(\alpha, \beta, \beta')$ in the presence of selfish nodes is upper bounded by

$$C_{r,s}(\alpha, \beta, \beta') \leq \sum_{i=0}^{g-1} u_i \min\{\alpha, (d - \mathcal{L}_0 - \sum_{j=0}^{i-1} u_j)\beta + (t - l_i - u_i)\beta'\}.$$

Proof: Consider a cut of the network, between the set U which contains the source S, and its complementary set $\bar{U}$ which contains the data collector DC. The information flow goes from the source to the data collector, through $x_{\rm in} \to x_{\rm coord} \to x_{\rm out}$, where both edges are assumed to have capacity α. Let u_0 be the number of new comers contacted by the data collector in the first group of t new comers, with m of them in U, and $u_0 - m$ of the others in $\bar{U}$. Take a first node, if it belongs to U, then it contributes to α (if either $x_{\rm coord} \in U$ or $x_{\rm coord} \in \bar{U}$) to the cut, thus the m nodes in U contribute to a total of $m\alpha$ to the cut.

Consider now the $u_0 - m$ nodes in $\bar{U}$. There are two contributions to the cut, coming from either $x_{\rm in}$ or $x_{\rm coord}$. The $x_{\rm in}$ part downloads from live nodes, of which, there are $\mathcal{L}_0$ selfish nodes. In an adversarial scenario, the first $\mathcal{L}_0$ nodes contacted may all be selfish, and as a result, the contribution to the cut would be $(d-\mathcal{L}_0)\beta$. Now for $x_{\rm coord}$, it contacts $t-1$ other new comers, $u_0 - m$ could already be in $\bar{U}$ (including itself), and l_0 could be selfish, thus the cut is increased of $(t - (u_0 - m) - l_0)\beta'$, for a total of

$$\begin{aligned} c_0(m) &\geq m\alpha + (u_0 - m)[(d - \mathcal{L}_0)\beta + \\ &\quad (t - u_0 + m - l_0)\beta'] \\ &\geq u_0 \min\{\alpha, (d - \mathcal{L}_0)\beta + (t - l_0 - u_1)\beta'\} \end{aligned}$$

by a concavity argument: since we have a function concave in m, it takes values always greater than in its minima which are on the domain boundary, namely in $m = 0$ (for which we have $u_0[(d - \mathcal{L}_0)\beta + (t - l_0 - u_1)\beta']$) and in $m = u_0$ (for which we have $u_0\alpha$). Thus the function is always greater than in the value it takes at the smallest of its minima.

Analogously, for the second group u_1, taking into account that $x_{\rm in}$ might contact among the live nodes those who joined in the first group of u_0 nodes, we get

$$\begin{aligned} c_1(m) &\geq m\alpha + (u_1 - m)[(d - \mathcal{L}_0 - u_0)\beta + \\ &\quad (t - u_1 + m - l_1)\beta'] \\ &\geq u_1 \min\{\alpha, (d - \mathcal{L}_0 - u_0)\beta + (t - l_1 - u_1)\beta'\}. \end{aligned}$$

By iteration and by summing over all the groups $u_0, \ldots, u_{g-1}$ such that $u_0 + \ldots + u_{g-1} = k$ we get

$$\sum_{i=0}^{g-1} u_i \min\{\alpha, (d - \mathcal{L}_0 - \sum_{j=0}^{i-1} u_j)\beta + (t - l_i - u_i)\beta'\}. \quad (7)$$

■

As explained in Section II, one of the two extremes in the storage-bandwidth trade-off is the minimum storage regeneration (MSR) point, which corresponds to the minimum amount of storage that is needed at each node to support data reconstruction by data collector by contacting k nodes. The minimum storage point continues to be $\alpha = B/k$ under our selfishness model.

Since Proposition IV-A is true for all possible values of u_i, it also holds particularly when $u_i = t - l_i$ for all i. Such a choice of u_is eliminates the β' component from the min-cut equation, allowing us to bound the value of β at the MSR point as follows.

Recall that $\sum_{i=0}^{g-1} u_i = k$, hence, $gt - \mathcal{L} = k$, so when $u_i = t - l_i$ we have

$$g = \frac{k + \mathcal{L}}{t}.$$

For data reconstruction, we need $B \leq C_{r,s}(\alpha, \beta, \beta')$, hence

$$B \leq \sum_{i=0}^{\frac{k+\mathcal{L}}{t}-1} (t - l_i) \min\{\alpha, (d - \mathcal{L}_0 - \sum_{j=0}^{i-1}(t - l_j))\beta\},$$

where $\alpha = B/k$. Note that the expression on the right hand side is less than or equal to $\frac{B}{k}\sum_{i=0}^{\frac{k+\mathcal{L}}{t}-1}(t - l_i)$, which is however equal to B (the same as the expression on the left hand side).

Thus, for every i

$$(d - \mathcal{L}_0 - \sum_{j=0}^{i-1}(t - l_j))\beta \geq B/k.$$

Indeed, we know that having all the min terms equal to B/k gives B, thus it cannot be that one of the terms is strictly smaller than B/k. The expression on the left hand side is the smallest when $i = \frac{k+\mathcal{L}}{t} - 1$, which in turn means

$$(d - \mathcal{L}_0 - \sum_{j=0}^{\frac{k+\mathcal{L}}{t}-2} (t - l_j))\beta \geq B/k.$$

Consequently, the smallest feasible value for β (which in turn leads to the smallest usage of bandwidth for regeneration) is

$$\frac{B/k}{(d - \mathcal{L}_0) - k + (t - l_{(k+\mathcal{L})/t-1})}. \quad (8)$$

This suggests that the bandwidth needed for download from the live nodes only depends on the last phase of regeneration, where $d-\mathcal{L}_0$ and instead d where contacted, and likewise, only $(t-l_{(k+\mathcal{L})/t-1})$ nodes instead of $t-1$ actually participated in the collaborative phase. We can thus conclude that

$$\frac{B/k}{(d-\mathcal{L}_0)-k+t} \leq \beta \leq \frac{B/k}{(d-\mathcal{L}_0)-k+(t-l^{max})}. \quad (9)$$

We will like to specifically emphasize that the above bounds on β are not to be confused with the range of values β can take on the trade-off curve. Instead, what this result implies is that, even for the minimum storage point, the minimum feasible β can be anywhere within this range, and depends on the precise number of selfish nodes involved in the collaborative phase, as noted in (8).

To compute β', we consider the other extreme regime, where $u_i = 1$ for all i, and thus $g = k$ (recall that we still have $\alpha = B/k$). Then

$$B \leq \sum_{i=0}^{k-1} \min\{B/k, (d-\mathcal{L}_0-i)\beta + (t-l_i-1)\beta'\}.$$

Similarly as the computations done for β, since

$$\min\{B/k, (d-\mathcal{L}_0-i)\beta + (t-l_i-1)\beta'\} \leq B/k$$

we have that

$$\sum_{i=0}^{k-1} \min\{B/k, (d-\mathcal{L}_0-i)\beta + (t-l_i-1)\beta'\} \leq B,$$

and thus equality holds:

$$\sum_{i=0}^{k-1} \min\{B/k, (d-\mathcal{L}_0-i)\beta + (t-l_i-1)\beta'\} = B.$$

We observe that this is a sum of k terms, so if any of the *min* terms were smaller than B/k, there would be a contradiction. Thus, it must be that $(d-\mathcal{L}_0-i)\beta+(t-l_i-1)\beta' \geq B/k$ for all $i = 0, ..., k-1$. The smallest feasible β' then corresponds to $i = k-1$, and we obtain that

$$(d-\mathcal{L}_0-(k-1))\beta + (t-l_{k-1}-1))\beta' = B/k. \quad (10)$$

This simplifies to

$$\beta' = \frac{B/k-(d-\mathcal{L}_0-(k-1))\beta}{t-l_{k-1}-1}.$$

Using (9), we determine that

$$\frac{(B/k)(t-l^{max}-1)}{(d-\mathcal{L}_0-k+t-l^{max})(t-1)} \leq \beta' \quad (11)$$

and

$$\beta' \leq \frac{(B/k)(t-1)}{(d-\mathcal{L}_0-k+t)(t-l^{max}-1)}. \quad (12)$$

With suitable choices of parameters $\mathcal{L}_0, l^{max}$, the results from Proposition 1 on standard (independent) regenerations and Proposition 2 corresponding to collaborative regeneration can be deduced (not surprisingly) from the results of our generalization.

- If $l^{max} = 0$, then there is no selfish node in the collaborative phase, only $\mathcal{L}_0$ live nodes might be selfish, and thus the bounds described in (9) and (11)-(12) give
$$\alpha = \frac{B}{k},\ \beta = \beta' = \frac{B}{k}\frac{1}{d-\mathcal{L}_0-k+t}.$$
Note that this is analogous to using less (i.e., $d-\mathcal{L}_0$ instead of d) nodes from among the live nodes for regeneration, and the specific result from Proposition 2 can be obtained by furthermore setting $\mathcal{L}_0 = 0$.
- If l^{max} takes its maximum value, that is $l^{max} = t-1$, that would imply that there is no collaboration. The upper bound in (9) is then satisfied only corresponding to $t = 1$, giving $\beta = \frac{B/k}{(d-\mathcal{L}_0)-k+1}$ which is analogous to the result from Proposition 1 for standard independent regeneration when $\mathcal{L}_0 = 0$. Also, $l^{max} = t-1$ implies that the coefficient of β' in (10) is zero, and hence there is no information from the collaborative flows, and thus there is no practical meaning in discussing about β'.

These extreme cases are essentially a sanity check of our generalization, and the drawn conclusions are on expected lines. Similar conclusions can also be drawn about the other extreme point (minimum bandwidth regeneration) in the trade-off curve. Unlike the extreme points however, the intermediate points in the trade-off curve are not as amenable to closed form analysis, and comprise of an interesting regime, which we study using numerical optimization and discuss later in Section IV-C.

B. Effect of polluting nodes

We now consider a worse case where the nodes are not selfish anymore, but are maliciously sending wrong data. We assume that there are $\mathcal{B}_0$ polluting nodes among the live nodes in any generation of regeneration, while $b_i \leq b^{max}$ is the number of polluting nodes among the ith group of newcomers, with $\mathcal{B} = \sum_{i=0}^{g-1} b_i$.

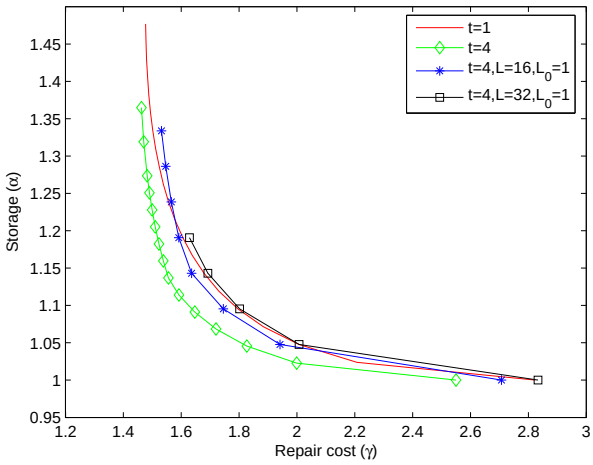
Proposition 4: The resiliency capacity $C_{r,p}(\alpha, \beta, \beta')$ in the presence of polluting nodes is upper bounded by

$$C_{r,p}(\alpha, \beta, \beta') \leq \sum_{i=0}^{g} u_i \min\{\alpha, (d-2\mathcal{B}_0-\sum_{j=0}^{i-1} u_j)\beta + (t-2b_i-u_i)\beta'\}.$$

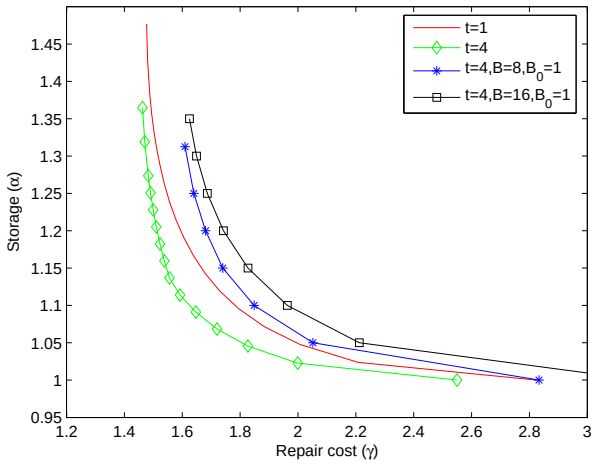
Proof: Let u_0 be the number of new comers contacted by the data collector in the first group of t comers, with m of them in U, and $u_0 - m$ of the others in $\bar{U}$. As in the proof above for selfish nodes, the contribution to the bound is $m\alpha$.

We now look at the $u_0 - m$ nodes in $\bar{U}$. There are two contributions to the cut, coming from either x_{in} or x_{coord}. Take the first node. The x_{in} part downloads from live nodes. Among these live nodes, there could be $\mathcal{B}_0$ polluting nodes. It thus gets a system of linear equations[2] from the d nodes, solving which would provide the unknown pieces of the encoded blocks. In the standard regeneration scenario, the unknowns correspond to the different pieces stored in the node itself. In the collaborative regeneration scenario, the unknowns include

[2]Since all the network coding results used rely on linear network coding, we use an argument valid in this setting.



(a) Trade-off under selfish attacks: $\mathcal{L}_0 = 1, l_i \leq 1, \mathcal{L} = 16/32$



(b) Trade-off under pollution attacks: $\mathcal{B}_0 = 1, b_i \leq 1, \mathcal{B} = 16/32$

Fig. 3. Storage-bandwidth tradeoff curves (normalized with B/k) using collaborative regenerating codes under Byzantine (selfish and pollution, respectively) attacks, determined by considering $g = 32$ generations or regenerations where t new nodes join and collaborate in each generation.

a subset of its own pieces, and additional information which allows it to collaborate and help other nodes regenerate.

There might or might not be wrong equations, depending on whether any of the live Byzantine nodes are contacted, but to be able to detect them, a naive, brute-force technique will be to solve all possible valid combinations (determined by the number of unknowns) of the subsets of equations, and choose the solution which concurs in majority of these combinations. Independently of even if more elegant mechanisms are employed, in order to actually figure out which equations are valid, it requires $\mathcal{B}_0$ good equations to compensate for the $\mathcal{B}_0$ potentially wrong ones. This is a more fundamental limit in Byzantine settings [14]. Having said that, we will like to note that if some extrinsic information is available, better Byzantine fault tolerance may be achievable, which we will briefly discuss later in Section VI. However, the rest of this section continues the analysis under the assumption that no other extrinsic (side-channel) information is available.

Thus among the d nodes contacted, those which will provide actual information to recover the lost data contribute to the cut by only $(d - 2\mathcal{B}_0)\beta$. Now for x_{coord}, it contacts $t - 1$ other new comers (together with the edge $x_{\text{in}} \to x_{\text{coord}}$), $u_0 - m$ could already be in $\bar{U}$, and b_0 could be bad, thus using the same argument as for $\mathcal{B}_0$, the contribution to the cut is $(t - (u_0 - m) - 2b_0)\beta'$, for a total of

$$\begin{aligned} c_0(m) &\geq m\alpha + (u_0 - m)[(d - 2\mathcal{B}_0)\beta + \\ &\quad (t - u_0 + m - 2b_0)\beta'] \\ &\geq u_0 \min\{\alpha, (d - 2\mathcal{B}_0)\beta + (t - 2b_0 - u_1)\beta'\}. \end{aligned}$$

Likewise, for the second group u_1, we get

$$\begin{aligned} c_1(m) &\geq m\alpha + (u_1 - m)[(d - 2\mathcal{B}_0 - u_0)\beta + \\ &\quad (t - u_1 + m - 2b_2)\beta'] \\ &\geq u_1 \min\{\alpha, (d - 2\mathcal{B}_0 - u_0)\beta + (t - 2b_2 - u_1)\beta'\}. \end{aligned}$$

By iterating and by summing over all the groups $u_0, \ldots, u_{g-1}$ such that $u_0 + \ldots + u_{g-1} = k$ we get

$$\sum_{i=0}^{g} u_i \min\{\alpha, (d - 2\mathcal{B}_0 - \sum_{j=0}^{i-1} u_j)\beta + (t - 2b_i - u_i)\beta'\}.$$

■

C. Interpretation of the analysis

We are interested in understanding the effects of both selfish and polluting nodes on the storage-bandwidth storage trade-off curve. To do so, we numerically minimize the bandwidth under the respective min-cut constraints, and report some of our results in Fig 3 corresponding to $d = 48$, $k = 32$, $t = 4$, $g = 32$ and compare how the trade-off curves for different adversarial scenarios behave with respect to both collaborative and standard regenerating codes.

In Fig. 3 (a), selfish nodes are introduced in the network. We fix their maximum number among the live nodes to be only $\mathcal{L}_0 = 1$, and similarly $l^{max} = 1$ bounds the number of selfish nodes during collaboration. We consider two cases: when $\mathcal{L} = 16$, that is all together 16 selfish nodes interfered during collaboration, and $\mathcal{L} = 32$, that is one selfish node was present at each stage of the regeneration process. The optimization was performed by letting the parameters β, β' range through a range of values limited by the MSR and MBR points. Derivation for the MSR points were provided above, analogous formulas can be derived for the MBR points. We observe in Fig.3(a) that when only half of the g groups had selfish nodes ($\mathcal{L} = 16$), the performance gets close to standard regenerating codes for a middle range of repair cost values, while it is even worse for $\mathcal{L} = 32$. For the later, the trade-off curve is worse, as expected, since not only the collaboration phase is not contributing, but there is furthermore one selfish node in the live nodes themselves.

In Fig. 3(b), the same setting is repeated, this time with polluting nodes. We see that even a small number of pollutant

nodes in a collaborative regeneration group, or among the live nodes leads to drastic deterioration of what can be achieved using collaborative regeneration - casting some doubt on the efficacy of regenerating codes. In practice, some additional extrinsic mechanisms can alleviate the situation, which we will briefly mention in Section VI.

It is important to note that the plot for pollution attacks corresponds to the case where polluting nodes actually answer correctly to the request of a data collector, meaning in particular that the minimum storage point is still $\alpha = B/k$. If it were not the case, namely, the polluting nodes could give wrong data to the data collector, then the minimum storage point would shift to $\alpha = B/(k - 2\mathcal{B}_0)$. Further analysis is needed to comprehend the impact of the same, which we defer for future investigation.

V. Exact Collaborative Regenerating Codes

Currently, [20] is, up to our knowledge, the only example of explicit codes for exact regeneration with collaboration, which works specifically for only the minimum storage regeneration point. We will first recall the construction, before considering it in the context of Byzantine adversaries. Note that in presence of Byzantine nodes, the number of nodes to be accessed might be different than what is used if there are no Byzantine nodes, for example as noted above, the minimum storage point is shifted from B/k to $B/(k-2\mathcal{B}_0)$ where $\mathcal{B}_0$ is the number of Byzantine nodes that might send wrong information during data collection. Thus in what follows, we will retain k to denote the number of nodes that the data collector accesses to retrieve the data stored, while κ is used as the dimension of the codes used, such as for Reed-Solomon codes.

Consider the (n, κ) Reed-Solomon code which is defined over the finite field $\mathbb{F}_q$ with $q \geq n$ a power of a prime. Suppose that the object $\mathbf{o}$ to be stored in n nodes can be written as $\mathbf{o}^T = (\mathbf{o}_{11}, \ldots, \mathbf{o}_{1\kappa}, \ldots, \mathbf{o}_{t1}, \ldots, \mathbf{o}_{t\kappa})$ with $\mathbf{o}_{ij}$ in either $\mathbb{F}_q$ or any finite field extension of $\mathbb{F}_q$. Note that this means that the object is cut into a number of pieces which depends on the number t of (predetermined, expected) failures,[3] with $k < n - t$. Furthermore, [20] considers only the regime $k = d$.

The generator matrix G of the Reed-Solomon code is a $\kappa \times n$ Vandermonde matrix whose columns are denoted by $\mathbf{g}_i$, $i = 1, \ldots, n$. Every node is assumed to know G. Now create a matrix $\mathbf{O}$ as follows:

$$\mathbf{O} = \begin{bmatrix} \mathbf{o}_{11} & \ldots & \mathbf{o}_{1\kappa} \\ & & \\ \mathbf{o}_{t1} & \ldots, & \mathbf{o}_{t\kappa} \end{bmatrix}.$$

The ith node stores $\mathbf{O}\mathbf{g}_i$ where $\mathbf{g}_i$ denotes the ith column of G, for example, node 1 stores

$$\mathbf{O}\mathbf{g}_1.$$

[3]While such an assumption is somewhat restrictive, and design of more adaptive codes constitute an interesting future direction of research, we note that such codes can nevertheless be practically used either by over-estimating the number of faults (though this may not be optimal anymore), and also when failures are corrected lazily by deliberately postponing the repair process till a predetermined number of faults are accumulated.

The t rows represent what we will call the t *pieces* that the corresponding node stores. That is, the *encoded data block* stored by each node comprises of *multiple pieces*. We will use the size of such a piece to define one unit of data.

Any choice of κ nodes $i_1, \ldots, i_\kappa$ clearly allows to retrieve $\mathbf{o}$ since we get

$$\mathbf{O}[\mathbf{g}_{i_1}, \ldots, \mathbf{g}_{i_\kappa}]$$

where the matrix formed by any κ columns of G is a Vandermonde matrix and is thus invertible.

Let us now assume that t nodes go offline, and t new nodes join. Let us call the t new nodes as nodes 1 to t. The ith newcomer will ask $(\mathbf{o}_{i1}, \ldots, \mathbf{o}_{i\kappa})\mathbf{g}_j$ for any choice of κ nodes among the live nodes.

Each newcomer can invert the matrix formed by the columns of G, and each decode $(\mathbf{o}_{i1}, \ldots, \mathbf{o}_{i\kappa})$ respectively. Thus it can compute the piece corresponding to its own first row, and also can compute $(\mathbf{o}_{i1}, \ldots, \mathbf{o}_{i\kappa})\mathbf{g}_j$ and send it to the jth node, which all will do similar computations and likewise deliver the missing pieces to the other newcomers, hence completing the collaborative regeneration process.

Example 1: Consider the $(n, \kappa) = (7, 3)$ Reed-Solomon code which is defined over the finite field $\mathbb{F}_8 = \{0, 1, w, w^2, w^3, w^4, w^5, w^6, w^7\}$ with $w^3 = w + 1$. Suppose that the object $\mathbf{o}$ is to be stored in $n = 7$ nodes, while expecting to deal with $t = 2$ failures. First, represent the object as $\mathbf{o}^T = (\mathbf{o}_{11}, \mathbf{o}_{12}, \mathbf{o}_{13}, \mathbf{o}_{21}, \mathbf{o}_{22}, \mathbf{o}_{23})$ with $\mathbf{o}_{ij}$ in either $\mathbb{F}_8$ or any finite field extension of $\mathbb{F}_8$, say $\mathbb{F}_q$. The generator matrix G of the Reed-Solomon code is given by:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ w & w^2 & 1+w & w+w^2 & 1+w+w^2 & 1+w^2 & 1 \\ w^2 & w+w^2 & 1+w^2 & w & 1+w & 1+w+w^2 & 1 \end{bmatrix}.$$

Now create a matrix $\mathbf{O}$ as follows:

$$\mathbf{O} = \begin{bmatrix} \mathbf{o}_{11} & \mathbf{o}_{12} & \mathbf{o}_{13} \\ \mathbf{o}_{21} & \mathbf{o}_{22}, & \mathbf{o}_{23} \end{bmatrix}.$$

The ith node stores $\mathbf{O}\mathbf{g}_i$ where $\mathbf{g}_i$ denotes the ith column of G, for example, node 1 stores

$$\mathbf{O}\begin{bmatrix} 1 \\ w \\ w^2 \end{bmatrix} = \begin{bmatrix} \mathbf{o}_{11} + \mathbf{o}_{12}w + \mathbf{o}_{13}w^2. \\ \mathbf{o}_{21} + \mathbf{o}_{22}w + \mathbf{o}_{23}w^2 \end{bmatrix}.$$

Thus each encoded data block comprises of two pieces of size one unit each in this example, and the original object is of size six units, and each encoded block is of size two units.

Any choice of $k = \kappa = 3$ nodes i_1, i_2, i_3 clearly allows to retrieve $\mathbf{o}$ since we get

$$\mathbf{O}[\mathbf{g}_{i_1}, \mathbf{g}_{i_2}, \mathbf{g}_{i_3}]$$

where the matrix formed by any 3 columns of G is a Vandermonde matrix and is thus invertible.

Let us now assume that 2 nodes go offline, and 2 new nodes join. Let us call the two new nodes as node 1 and node 2. The first new comer will ask $(\mathbf{o}_{11}, \mathbf{o}_{12}, \mathbf{o}_{13})\mathbf{g}_i$ for any choice of 3 nodes among the 5 live nodes, while the second new

cost	$l_0 = \mathcal{L}_0 = 0$	$l_0 = 1, \mathcal{L}_0 = 0$	$l_0 = 0, \mathcal{L}_0 = 1$
β	1/2	1	β_{av}=3/4
β'	1/2	0	1/2
γ	2	3	2

TABLE I
SELFISH NODES: $\alpha = 1$, $t = 2$, $d = 3$

cost	$b_0 = \mathcal{B}_0 = 0$	$b_0 = 1, \mathcal{B}_0 = 0$	$b_0 = 0, \mathcal{B}_0 = 1$
β	1/2	1	1/2
β'	1/2	0	1/2
γ	2	3	3 ($d = 5$)

TABLE II
POLLUTING NODES: $\alpha = 1$, $t = 2$, $d = 3$

comer will similarly ask $(\mathbf{o}_{21}, \mathbf{o}_{22}, \mathbf{o}_{23})\mathbf{g}_i$ from any of the 5 live nodes.

Both new comers can invert the matrix formed by the columns of G, and decode each respectively $(\mathbf{o}_{11}, \mathbf{o}_{12}, \mathbf{o}_{13})$ and $(\mathbf{o}_{21}, \mathbf{o}_{22}, \mathbf{o}_{23})$. Now the first node can compute the piece corresponding its own first row, and also can compute $(\mathbf{o}_{11}, \mathbf{o}_{12}, \mathbf{o}_{13})\mathbf{g}_2$ and send it to the second node, which likewise can compute $(\mathbf{o}_{21}, \mathbf{o}_{22}, \mathbf{o}_{23})\mathbf{g}_1$ and send it to node 1, which completes the regeneration process.

Thus, overall, eight units of data transfer is needed in this example, in order to replenish four units of lost data. Note that if one node did regeneration of two pieces using six data transfer, it could send the other node the other two pieces directly, needing again a total of eight units of data transfer. As mentioned previously in Section II, when $d = k$ as is the case for this code construction, the repair cost is the same as that of erasure codes, though with a better load balancing, as seen in this example.

We use the above toy example to illustrate the effect of selfish/polluting nodes. We consider two scenarios with selfish nodes: (i) Consider that one of the two newcomer nodes does not agree to collaborate with the other. In this case, the other node has no choice than to download more data from the live nodes. Given that each node contacts $d = 3$ live nodes, this means downloading 2 encoded pieces from each of the 3 nodes, for a total of 6 pieces. The cost of one repair is then $\gamma = 3$. Note that all the bandwidth costs in this example are normalized with $B/k = 2$. Note also that, in a general scenario, during the collaborative regeneration process, different nodes may face different number of selfish nodes, affecting accordingly the necessary bandwidth for regenerations. (ii) Consider now that both newcomer nodes collaborate, but there is $\mathcal{L}_0 = 1$ selfish node among the live nodes. In the worst case, both newcomers try two well behaved lives nodes and the same non-responding node. It might or not be easy for these newcomers to contact other nodes that are willing to help with the download. So if the newcomers decide to keep on downloading more from only the already responding nodes, we get that they each need to download at least 1 piece of data from one responding live node, and 2 pieces from the other responding live node, for an average of $\beta_{av} = (1 + 1/2)/2 = 3/4$ download bandwidth, after which collaboration can proceed as normal. The bandwidth costs are summarized in Table I. It can be seen that in this case $\mathcal{L}_0$ is not harmful for total bandwidth cost per repair, though it does imbalance the network load.

Let us now consider the case of polluting nodes, where we first assume that the polluting nodes do not interfere with data collection. (i) If one of the two collaborating nodes is polluting, then the other node has no choice but to retrieve the whole object from the live nodes, and no collaboration is possible. In this particular toy example, this gives the same end result as with one selfish collaborating node, since in both cases reconstructing the object is needed. (ii) If $\mathcal{B}_0 = 1$, in the worst case, both collaborating nodes get 1 fake encoded piece of data, and 2 genuine ones. Now to check which data, if any, is corrupted, 2 more genuine encoded fragments are needed. However, since the nodes do not know which of the live nodes might have gone rogue, they are forced to contact the remaining two more nodes. This inflates the number $d = 3$ to $d = 5$, the maximum amount of available live nodes here. These results are summarized in Table II.

Finally, in the worst case, the polluting nodes can also send wrong information to the data collector. Since the stored data at the live nodes is encoded using Reed-Solomon code in this example, it is resistant to errors, as long as the number of errors is not more than twice the maximal number of tolerated erasures. However, this also means that either the number of contacted nodes is increased, or for a fixed k, the amount of data stored in each node has to be increased.

In this example, since we have 5 live nodes, only one polluting node sending wrong information to the data collector can be tolerated. More generally, a (n, κ) Reed-Solomon code is known to tolerate $n_s = n - \kappa$ erasures, or $n_b = (n - \kappa)/2$ errors, or more generally n_s erasures and n_b errors as long as $n_s + 2n_b \leq n - \kappa$.

VI. PRACTICAL CONSIDERATIONS

In practice, the number of Byzantine nodes is not known a priori. While selfish nodes are trivially dealt with, pollutants can not be detected a priori, and hence are difficult to deal with. Thus, regenerating nodes may try to first regenerate with responses from the minimal number of nodes, assuming (possibly, wrongly) that there are no pollutants. If there are however pollutants, then the regenerated block will be different from what ought to have been regenerated. For exact regeneration, a globally known hash function, and prior, secure and globally accessible look-up table with the hashes (signature) for the encoded fragments of an object can be used, to verify with low communication overhead whether the regenerated block is correct or not. If integrity violation is detected, then progressively more nodes data may be downloaded, possibly by contacting more nodes. Such an extrinsic information can alleviate the effect of Byzantine nodes. As soon as the node has enough good information to regenerate, it can be easily verified, thus, there is no need to waste one bit of good information just to negate each wrong bit. For the example in Section V, with the use of such extra information, if there is one pollutant among the live nodes, the regenerations could be carried out by contacting at most four of the live nodes,

and one can also tolerate upto two Byzantine nodes - both infeasible without such extra information.

The actual achieved system performance will depend on the precise protocol details, and there will in all cases be additional protocol overheads, both in terms of storage as well as bandwidth needs. Such systems considerations were beyond the scope of the current paper which studies the theoretical constraints of regenerating codes in the presence of Byzantine nodes. Furthermore, regenerating codes incur high computational complexity [8] even without consideration of Byzantine failures. Byzantine nodes will further amplify the computational overheads. Thus, even though regenerating codes have promising qualities (theoretically), and have been much studied in the last few years, all these practical issues need to be taken into account and studied holistically, to determine their benefits and trade-offs in practice.

VII. Related works

We have already provided a concise survey of regenerating codes related literature in the discussion precursing the new bounds for collaborative regenerating codes under Byzantine faults determined in this paper. Thus, here we will discuss about pollution attacks in general, both in the context of different kinds of peer-to-peer systems, and in the context of network coding.

Pollution attacks are mitigated in peer-to-peer content dissemination systems [9], [5], [15] using a combination of proactive strategies such as digital signature provided by the content source or by reactive strategies such as by randomized probing of the content source, leveraging on the causal relationship in the sequence of content to be delivered, as well as by deploying reputation mechanisms. In such settings, the prevention of pollution attacks is furthermore facilitated by a continuous involvement of the content source, which is assumed to be online.

Generally speaking, P2P storage environments are fundamentally different from P2P content distribution networks. The content owner may or not be online all the while. Furthermore, the very premise of regenerating codes is a setting where no one node possesses the whole copy of the object to be stored, i.e., a hybrid storage strategy where one full copy of the data is stored in addition to the encoded blocks, is excluded for other practical considerations. Likewise, different stored objects may be independent of each other. Hence, mechanisms to provide protection against errors as an inherent property of the code (similar to error correcting codes) becomes essential. The presented study looks at the fundamental capacity of such codes under some specific adversarial models. This work is thus complementary to other existing storage systems approaches such as incentive and reputation mechanisms [4] and remote data checking techniques [3] for data outsourced to third parties to name a few. Likewise, Byzantine algorithms have been used in Oceanstore [13] to support reliable data updates. The focus there is on application level support for updating content, rather than storage infrastructure level Byzantine behavior studied in this paper.

Pollution attacks have also been studied specifically in the context of network coding where it has already been noticed that though collaboration among the nodes through coding does increase the throughput, it also makes the network much more vulnerable to pollution attacks than under traditional routing. To remedy this threat, several authentication techniques have been studied in the context of network coding, such as digital signatures (for e.g., [2], [23], [24], [1] and authentication codes [17].

VIII. Conclusion

Leveraging on network coding results, regenerating codes were introduced as a redundancy technique in networked distributed storage. Collaboration among the nodes participating in the regeneration process has recently been shown to improve the storage-bandwidth trade-offs. In this paper we determine the resilience capacity of collaborative regeneration in the presence of selfish or polluting nodes, and expose that collaboration may be detrimental under Byzantine attacks to such an extent that it may instead be better not to collaborate. We also show that, while collaborative regeneration is extremely vulnerable as a stand alone process, Byzantine attacks can be easily mitigated using some additional extrinsic information.

References

[1] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a Linear Subspace: Signature Schemes for Network Coding", PKC 2009, LNCS.

[2] D. Charles, K. Jain, and K. Lauter, "Signatures for Network Coding", Conference on Information Sciences and Systems, 2006.

[3] B. Chen, R. Curtmola, G. Ateniese, R. Burns, "Remote data checking for network coding-based distributed storage systems", *in Proceedings of the 2010 ACM workshop on Cloud computing security workshop*

[4] L.P. Cox and B.D. Noble, "Samsara: honor among thieves in peer-to-peer storage", *in ACM symposium on Operating systems principles, 2003*

[5] P. Dhungel, X. Hei, K. Ross and N. Saxena, "The Pollution Attack in P2P Live Video Streaming: Measurement Results and Defenses", *in Peer-to-Peer Streaming and IP-TV Workshop (P2P-TV 2007)*

[6] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, K. Ramchandran, "Network Coding for Distributed Storage Systems", *in INFOCOM 2007*.

[7] A. Duminuco, E. W. Biersack, "Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems", *in P2P 2008*

[8] A. Duminuco, E. W. Biersack, "A Practical Study of Regenerating Codes for Peer-to-Peer Backup Systems", *in ICDCS 2009*

[9] A. Habib, D. Xu, M. Atallah, B. Bhargava and J. Chuang, "Verifying Data Integrity in Peer-to-Peer Media Streaming", *in Twelfth Annual Multimedia Computing and Networking (MMCN 2005)*

[10] C. Huang, M. Chen, and J. Li, "Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems", *in Proceedings of IEEE International Symposium on Network Computing and Applications (NCA 2007)*

[11] Y. Hu, Y. Xu, X. Wang, C. Zhan, P. Li, "Cooperative Recovery of Distributed Storage Systems from Multiple Losses with Network Coding", *IEEE J. on Selected Areas in Comm., vol. 28, no. 2, pp.268276, Feb, 2010*

[12] A-M. Kermarrec, G. Straub, N. L. Scouarnec, "Repairing Multiple Failures with Coordinated and Adaptive Regenerating Codes", *in Arxiv Feb 2011*

[13] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage", *in ACM SIGARCH Computer Architecture News, 2000*

[14] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem", *in ACM Trans. Program. Lang. Syst. 1982, vol. 4, issue 3*

[15] X. Liu and A. Datta, "Attack resilient P2P dissemination of RSS feed", *in Peer-to-Peer Networking and Applications Journal, 2010*

[16] F. Oggier, A. Datta, “Self-repairing Homomorphic Codes for Distributed Storage Systems”, *in INFOCOM 2011*
[17] F. Oggier, H. Fathi, “Multi-receiver authentication code for network coding ”, 46th Annual Allerton Conference. 2008
[18] S. Pawar, S. El Rouayheb, K. Ramchandran, “Securing Distributed Storage Systems against Eavesdropping and Adversarial Attacks”, *Arxiv*
[19] I. S. Reed and G. Solomon, “Polynomial Codes Over Certain Finite Fields”, *Journal of the Society for Industrial and Appl. Mathematics*, no 2, vol. 8, SIAM, 1960.
[20] K. W. Shum, “Cooperative Regenerating Codes for Distributed Storage Systems”, to appear at *ICC 2011*, available at arXiv:1101.5257v1.
[21] Y. Wu, A. G. Dimakis, K. Ramchandran, “Deterministic Regenerating Codes for Distributed Storage”, *in Allerton 2007*.
[22] Y. Wu, A. G. Dimakis, “Reducing Repair Traffic for Erasure Coding-Based Storage via Interference Alignment”, *in ISIT 2009*.
[23] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan,”An Efficient Signature-based Scheme for Securing Network Coding against Pollution Attacks”, IEEE INFOCOM, 2008.
[24] F. Zhao, T. Kalker, M. Medard, and K.J. Han, ”Signatures for Content Distribution with Network Coding”, IEEE International Symposium on Information Theory, 2007.