

Redundancy Management for P2P Backup

Laszlo Toka
Eurecom, France

Pasquale Cataldi
British Sky Broadcasting, UK

Matteo Dell’Amico
Eurecom, France

Pietro Michiardi
Eurecom, France

Abstract—We propose a redundancy management mechanism for peer-to-peer backup applications. Since, in a backup system, data is read over the network only during restore processes caused by data loss, redundancy management targets *data durability* rather than attempting to make each piece of information available at any time.

Each peer determines, in an on-line manner, an amount of redundancy sufficient to counter the effects of peer deaths, while preserving acceptable data restore times. Our experiments, based on trace-driven simulations, indicate that our mechanism can reduce the redundancy by a factor between two and three with respect to redundancy policies aiming for data availability. These results imply an according increase in storage capacity and decrease in time to complete backups, at the expense of longer times required to restore data. We believe this is a very reasonable price to pay, given the nature of the application.

I. INTRODUCTION

Many users do not backup their data regularly; costs and poor usability are among the main reasons why existing backup solutions are not used. A P2P approach to backup can be a viable technique to overcome these issues, providing a seamless and extremely cheap way to keep data safe.

As we discuss in Sec. II, the focus is on durability, *i.e.* guaranteeing that data is not lost. A specialized backup application has to fulfill less stringent requirements than a generic P2P storage application in several aspects. First, backups should only be readable by their owner, making confidentiality requirements easy to satisfy with standard cryptographic techniques. Second, backup involves the bulk transfer of potentially large quantities of data, both during regular backups and, in the event of data loss, during restore operations. Therefore, read and write latencies of hours have to be tolerated by users. Third, owners have access to the original copy of their data, making it easy to inject additional redundancy in case data stored remotely is partially lost. Fourth, since data is read only during restore operations, the application does not need to guarantee that any piece of the original data should be promptly accessible in any moment, as long as the time needed to restore the whole backup remains under control. For all these reasons, we claim that it is sensible to design peer-to-peer applications that perform exclusively backup.

We design and evaluate a new *redundancy management* mechanism for backup, which achieves data durability without requiring high redundancy levels nor fast mechanisms to detect node failures. In our mechanism, which is described in Sec. III, the redundancy level applied to backup data is computed in an *on-line* manner. Given a time window that accounts for failure detection and data repair delays and a system-wide statistic

on peer deaths, peers determine the redundancy rate while backing up data. A byproduct of our approach is that, if the system state changes, then peers can adapt to such dynamics and modify the redundancy level on the fly.

We evaluate our redundancy management scheme via trace-driven simulations. In Sec. IV, we show that our approach drastically decreases strain on resources, reducing the storage and bandwidth requirements by a factor between two and three, as compared to redundancy schemes that use a fixed, system-wide redundancy factor. This result yields higher storage capacity for the system and shorter backup times at the expense of longer restores, which is a very reasonable price to pay considering the requirements of backup applications.

II. APPLICATION SCENARIO

We assume *data owners* to specify one local folder containing important data to backup. Backup data remains available *locally* to data owners, unlike many online storage applications in which data is only stored remotely.

We consider the problem of long-term storage of *backup objects*: large, immutable, and opaque pieces of data. They consist of encrypted archives of changes to files, such that recovering them allows reconstructing the history of data.

Backup objects are stored on inherently unreliable peers, which join and leave the system unpredictably (*churn*). Moreover, peers may crash and possibly abandon the P2P application (*death*). As such, their connectivity must be continuously tracked, since it cannot be determined *a priori* [1].

While the literature provides a vast array of solutions to guarantee *data availability* when using failure-prone machines to store data [1], [2], we claim that online data backup applications should instead target *data durability*. Moreover, backup applications often involve the bulk transfer of a large quantity of data. Therefore, such applications should aim for throughput rather than aiming at low-latency read operations, in addition to be resilient against peer churn and deaths.

Data durability can be achieved by injecting a sufficient level of data redundancy in the system to make sure data gets never lost, despite peer churn and peer deaths, which cause the data redundancy level to drop. Hence, the focus of our work is to design a redundancy management mechanism that is tailored to the peculiar data access patterns of backup applications and that strives for data durability.

Using erasure coding, a backup object of size o is encoded in n fragments of a fixed size f which are ready to be placed on remote peers. Any k out of n fragments are sufficient to recover the original data; when using optimal erasure

coding techniques, $k = \lceil o/f \rceil$. The redundancy management mechanism determines the redundancy level $r = nf/o$.

During the **backup phase**, data owners upload fragments to some selected remote peers. The backup phase completes when *all* n fragments are placed on remote peers.

Once the backup is completed, the **maintenance phase** begins: should the redundancy level decrease in the system due to peer deaths, it has to be reestablished by re-injecting new fragments. The crux of data maintenance is to determine when the redundancy is too low to allow data recovery and to generate other fragments to rebalance it. In the event of a peer death, the system may trigger the maintenance phase immediately (eager repairs) or may wait for a number of fragments to be tagged as lost before proceeding with the repairs (lazy repairs) [1], [3], [4]. As such, it is important to discern *unambiguously* permanent deaths from the normal online behavior of peers: this is generally achieved by setting a time-out value, Θ , for long-term peer unavailability.

As peers hold a local copy of their data, maintenance can be executed solely by the data owner, or it can be delegated. In both cases, it is important to consider the timeframe in which data cannot be maintained. First, fragments may be lost before a host failure is detected using the time-out mechanism outlined above. This problem is exacerbated by the availability pattern of the entity (data owner or other peers) in charge of the maintenance operation: indeed, host failures cannot be detected during the offline periods. Second, data loss can occur during the restore process. For these reasons, in Sec. III, we consider a redundancy management policy that ensures data is not lost in the time-window $w = \Theta + a_{\text{off}}$, where a_{off} is the (largest) transient off-line period of the entity in charge of data maintenance. For example, if the data owner executes data maintenance: first, it needs to be on-line to generate new fragments and upload them, and second, the timeout Θ has to be expired. Additionally, our mechanism selects a redundancy level such that data loss does not occur before the restore process is completed.

In the unfortunate case of a disk or host crash, the **restore phase** takes place. Data owners contact the remote machines holding their fragments, download at least k of them, and reconstruct the original backup data.

Before proceeding, we now define the performance metrics we are interested in for this work. Overall, we compute the performance of a P2P backup application in terms of the amount of time required to complete the backup and the restore phases, labelled *time to backup* (TTB) and *time to restore* (TTR). Moreover, in the following sections, we use baseline values for backup and restore operations which bound both TTB and TTR. We compute such bounds as follows: let us assume an *ideal* storage system with unlimited capacity and uninterrupted online time that backs up user data. In this case, TTB and TTR only depend on the size of a backup object and on uplink bandwidth and availability of the data owner. We label these ideal values minTTB and minTTR . Formally, we have that a peer i with upload and download bandwidth u_i and d_i , starting the backup of an object of size o at time

t , completes its backup at time t' , after having spent $\frac{o}{u_i}$ time online. Analogously, i restores a backup object with the same size at t'' after having spent $\frac{o}{d_i}$ time online. Hence, we have that $\text{minTTB}(i, t) = t' - t$ and $\text{minTTR}(i, t) = t'' - t$.

III. REDUNDANCY MANAGEMENT

Data can be considered as durable if the probability to lose it, due to the permanent failure of hosts in the system, is negligible. The problem of designing a system that guarantees data durability can be approached under different angles.

As noted in previous works [5], [6], data availability implies data durability: a system that injects sufficient redundancy for data to be available at any time, coupled with maintenance mechanisms, automatically achieves data durability. These solutions are, however, too expensive in our scenario: the amount of redundancy needed to guarantee availability is much higher than what is needed to obtain durability.

Instead of using high redundancy, data durability can also be achieved with efficient maintenance. For example, in a datacenter, each host is continuously monitored: based on statistics such as the mean time to failure of machines and their components, it is possible to store data with very little redundancy and rely on system monitoring to detect and react immediately to host failures. Failed machines are replaced and data is rapidly repaired due to the dedicated and over-dimensioned nature of datacenter networks. Unfortunately, this approach is not feasible in a P2P setting. First, the interplay of transient and permanent failures makes failure detection a difficult task. Since it is difficult to discern deaths from the ordinary online behavior of peers, the detection of permanent failures requires a delay during which data may be lost. Furthermore, data maintenance is not immediate: in a P2P application deployed on the Internet, bandwidth scarceness and peer churn make the repair operation slow.

In summary: on the one hand durability could be achieved with high data redundancy, but the cost in terms of resources required by peers would be overwhelming. On the other hand, with little redundancy, durability could be achieved with timely detection of host failures and fast repairs, which are not realistic in a P2P setting.

Our goal is to design a mechanism that achieves data durability without requiring high redundancy or fast failure detection and repair. Since data is written once, during backup, and read (hopefully) rarely, during restores, we design a mechanism that injects only the data redundancy level required to compensate failure detection and data repair delays. That is, we define data durability as follows.

Definition 1. *Data durability d is the probability to be able to access data after a time window t , during which no maintenance operations can be executed.*

Definition 2. *The time window t is defined as $t = w + TTR$, where w accounts for failure detection delays and TTR is the time required to download a number of fragments sufficient to recover the original data.*

As discussed in Sec. II, w depends on whether the maintenance is executed by the data owner or is delegated, and can be thought of a parameter of our scheme.

A peer with n fragments on remote peers could lose its data if more than $n - k$ of them would get lost as well within the time window t . Let the data redundancy required to avoid this event be $r = n/k$. Now, let us assume peer deaths to be memoryless events, with constant probability for any peer and at any time. Then peer lifetimes are exponentially distributed stochastic variables with a parametric average τ . Hence, the probability for a peer to be alive after a time t is $e^{-t/\tau}$. Assuming death events are independent, data durability is $d = \sum_{i=k}^n \binom{n}{i} (e^{-t/\tau})^i (1 - e^{-t/\tau})^{n-i}$.

The value of d depends on t which, in turn, is a function of TTR. We thus propose to use the following heuristic to estimate the TTR of a generic peer p_0 . In case of a crash, we assume p_0 to remain online during the whole restore process. Therefore, assuming no network bottlenecks, p_0 's TTR can be either bounded by the download bandwidth D_0 of peer p_0 , or the upload rate of remote peers holding p_0 's data. Let us focus on the second case: we define the *expected upload rate* μ_i of a generic remote peer p_i holding a backup fragment of p_0 as the product of the availability of peer p_i and its upload bandwidth, that is $\mu_i = u_i a_i$.

Peer p_0 needs to download at least k fragments to fully recover a backup object. Let us assume these k fragments are served by the k remote peers with the highest expected upload rate μ_i . In this case, the ‘‘bottleneck’’ is the k -th peer with the lowest expected upload rate μ_k . Then, an estimation of TTR, that we label $eTTR$, can be obtained as follows:

$$eTTR = \max\left(\frac{o}{D_0}, \frac{o}{k\mu_k}\right). \quad (1)$$

Our redundancy management scheme works as follows: the redundancy level applied to backup data is computed by the combination of d and $eTTR$. Let us assume, for the sake of simplicity, the presence of a central coordinator that performs membership management of the P2P network: the coordinator keeps track of users subscribed to the application, along with short-term measurements of their availability, their (application-level) uplink capacity and the average death rate τ in the system. While a decentralized approach to membership management and system monitoring is an appealing research subject, it is common practice (e.g., Wuala) to rely on a centralized infrastructure and a simple heartbeat mechanism.

During a backup operation, peers query the coordinator to obtain remote hosts that can be used to store fragments, along with their availability. A peer constructs a backup object, and subsequently uploads k fragments to distinct, randomly selected available remote hosts. Then the peer continues to inject redundancy in the system, by sending additional fragments to randomly selected available peers, until a stop condition is met. Every time new fragments are uploaded, the peer computes d and $eTTR$: the stop condition is met if $d \geq \sigma_1$ and $eTTR \leq \sigma_2$. While selecting an appropriate σ_1 is trivial, in the following we define σ_2 as $\sigma_2 = \alpha \cdot \min TTR$, where

α is a parameter that specifies the degradation of TTR with respect to an ideal system, tolerated by users.

We now study the impact of the ratio $\frac{w+eTTR}{\tau}$:

- $\tau \gg w + eTTR$: this case is representative of a ‘‘mature’’ P2P application in which the dominant factor that characterizes peer deaths are permanent host failures, rather than users abandoning the system. Hence, $e^{-t/\tau}$ is close to 1, which implies that the target durability σ_1 can be achieved with a small n . As such, the condition on $eTTR \leq \sigma_2$ prevails on $d \geq \sigma_1$ in determining the redundancy level to apply to backup data. This means that the accuracy of the estimate $eTTR$ plays an important role in guaranteeing acceptable restore times; instead, errors on $eTTR$ have only slight impact on data durability.
- $\tau \sim w + eTTR$: this case is representative of a P2P application in the early stages of its deployment, where the abandon rate of users is crucial in determining the death rate. In this case, $e^{-t/\tau}$ can be arbitrarily small, which implies that $n \gg k$, i.e., the target durability d requires higher data redundancy. In this case, the condition $d \geq \sigma_1$ prevails on $eTTR \leq \sigma_2$. Hence, estimation errors on the restore times may have an impact on data durability: e.g., underestimating the TTR may cause n to be too small to guarantee the target σ_1 .

IV. PERFORMANCE EVALUATION

We proceed with a trace-driven system simulation, and focus on the performance metrics outlined in Sec. II. We perform a comparative study of the results achieved by a system using our redundancy management scheme and the traditional approach used for storage applications. For the latter case, we implement a technique in which the coding rate is set once and for all, based on a system-wide average of host availability.

We use traces as input to our simulator that cover both the online behavior of peers and their uplink and downlink capacities. Instead, long-term failures and the events of peers abandoning the applications, which constitute the peer deaths, follow a simple model, driven by the parameter τ , as explained in Sec. III. Due to the lack of traces that represent the realistic ‘‘data production rate’’ of Internet users, in this simulation study we confine our attention to a homogeneous setting: each user has an individual backup object of the same size.

Availability traces: The *online behavior* of users, i.e., their patterns of connection and disconnection over time, is difficult to capture analytically. We simulate a backup application using a real application trace that exhibits both heterogeneity and correlated user behavior. Our traces capture user availability, in terms of login/logoff events, from an instant messaging (IM) server for a duration of roughly 3 months. We argue that the behavior of regular IM users constitutes a representative case study. Indeed, for both IM and online backup, users are generally signed in for as long as their machine is connected to the Internet.

We only consider users that are online for an average of at least four hours per day, as done in Wuala. Once this filter

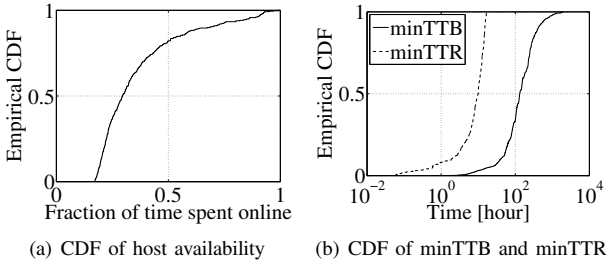


Fig. 1. Data resulting from the input traces. Note that users spending less than 4 hours per day online are filtered out.

is applied, we obtain the trace of 376 users. Since in P2P storage systems the number of neighbors each node interacts with is very often limited by design and scalability issues [7], we believe this trace size is acceptable. As shown in Fig. 1(a), most users are online for less than 40% of the trace length, while some of them are almost always connected.

Bandwidth distribution: Uplink capacities of peers are obtained by sampling a real bandwidth distribution measured at more than 300,000 unique Internet hosts for a 48 hour period from roughly 3,500 distinct ASes across 160 countries [8]. These values have a highly skewed distribution, with a median of 77 KBps and a mean of 428 KBps. To represent typical asymmetric residential Internet lines, we assign to each peer a downlink speed equal to four times its uplink.

Simulation Settings: The trace-driven online behavior of a peer is overridden only during the restore phase: we make the assumption that in such case, a peer remains online for the whole duration of the restore process.

In our study, each peer has $o = 10$ GB of data to backup (as soon as the simulation begins), and dedicates 50 GB of storage space to the application. The high ratio between these two values lets us disregard issues due to insufficient storage capacity. The fragment size is set to 160 MB, implying a minimum of $k = 64$ fragments needed for restores.

We define peers’ lifetimes to be exponentially distributed random variables with an expected value $\tau = \{90\text{days}, 1\text{year}, 4\text{years}\}$ (see Sec. III). Besides peer deaths, we study the impact of the parameter w , which contributes to the length of the time-window for which our redundancy management policy guarantees data durability, without maintenance (see Sec. III). As a reminder (see Sec. II), w accounts for failure detection delays. In our experiments w takes values from 0 to 4 weeks.

Our adaptive redundancy policy uses the following parameters: we set the thresholds $\sigma_1 = 0.9999$, so that the durability $d \geq \sigma_1$ and $\sigma_2 \leq \max(1 \text{ day}, 2 \cdot \text{minTTR})$ so that $eTTR \leq \sigma_2$. We compare against a baseline redundancy policy that aims to guarantee data availability [1], labeled here as “availability-based”. We set a target data availability of 0.99, and use the system-wide average availability $a = 0.36$ as computed from our availability traces. We obtain a value $n = 228$ and a redundancy rate $r = 3.56$.

For each set of parameters, the simulation results are ob-

tained by averaging ten simulation runs.

Results: Fig. 1(b) shows the cumulative distribution functions (CDF) of minTTB and minTTR obtained using the input traces discussed above. While backups generally take days to complete, restores are around an order of magnitude faster, due to asymmetric bandwidth and the fact that peers stay online during restore operations.

We now compare our scheme to the traditional fixed-redundancy scheme. First, we focus on the data redundancy level (that is, the code rate r) imposed by each approach. In Fig. 2(a), we show the average redundancy factor for our mechanism and the one computed for the fixed availability-based scheme, as a function of the parameter w and for different values of τ . We omit error bars from the plot as the variance around the mean is negligible. Clearly, for increasing values of w the redundancy rate increases. Note that our simulations account for a realistic bandwidth distribution and for real on-line user behavior, which influence the eTTR computation. When the dominant effect of non-transient failures is the reliability of Internet hosts (*i.e.*, τ is large), our mechanism achieves data durability (and a controlled TTR) with a small redundancy factor. Instead, when peer deaths are dominated by peers abandoning the system (*i.e.*, τ is small), our mechanism compensates with a larger redundancy rate. In summary, our scheme obtains a redundancy factor ranging roughly between *half* and *a third* of the availability-based scheme, increasing the storage capacity of the system by a corresponding factor between two and three. Since the amount of data to upload in case of a crash is proportional to redundancy, the impact of maintenance on bandwidth decreases accordingly.

In addition to improving the aggregate storage capacity of the system, our scheme impacts both backup and restore operations. Fig. 2(b) and 2(c) report the CDF of the ratio of TTB and TTR over their respective ideal counterparts, minTTB and minTTR. These plots are obtained with different values of w , for a fixed $\tau = 3$ months, and illustrate the results of our mechanism and the availability-based scheme. Fig. 2(b) indicates that, due to a lower redundancy factor, the median of the distribution of TTB is roughly reduced by a factor of four. Moreover, increasing values of w have essentially little impact on TTB. The price to pay for fast backup operations is shown in Fig. 2(c): restore operations take more time to complete w.r.t. a traditional approach to redundancy management. Here the w parameter plays an important role: for small w values, little redundancy is applied to backup data. As such, the opportunity to retrieve enough encoded fragments to restore data is largely affected by peer availability. Instead, when w is large, restore operations are more efficient and less sensitive to peer availability.

In summary, our results support the rationale underlying the design of our redundancy management scheme: TTB is generally several times larger than TTR, even in an ideal case (as shown in Fig. 1(b)). Because of this unbalance, we argue that it is reasonable to use a redundancy management scheme that trades longer TTR (which affects only users that suffer a crash) for shorter TTB (which affects all users).

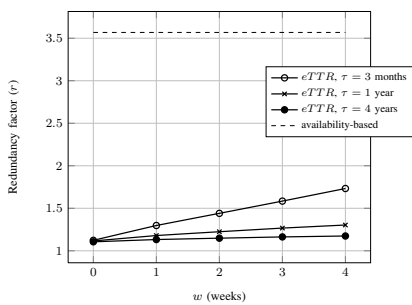
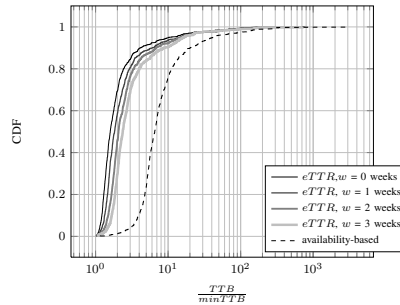
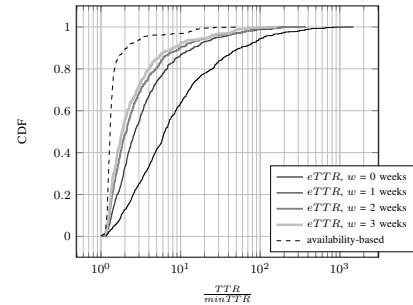
(a) Redundancy rate as a function of w .(b) CDF of TTB for $\tau = 3$ months.(c) CDF of TTR for $\tau = 3$ months.

Fig. 2. Experimental results.

TABLE I
CATEGORIZATION OF DATA LOSS EVENTS

Avg. lifetime (τ)	Total events	Incomplete backup		Failed restore
		Total	Unavoidable	
3 months	13%	10.4%	8.4%	2.6%
1 year	2.6%	2.6%	2.3%	None
4 years	0.5%	0.5%	0.25%	None

The main reason for errors on eTTR are due to the fact that the heuristic defined in Eq. 1 assumes k encoded fragments to be downloaded from the k fastest peers that hold backup data. In practice, however, the k encoded fragments are downloaded from the peers that are available when a restore operation is executed. Depending on the bandwidth distribution of the peers in the system, such difference can cause the estimated TTR value to be different from what achieved in practice.

Data loss can be caused by underestimating eTTR, when τ is small and the redundancy rate is bound by the durability estimation. In Table I, we illustrate the effects discussed by quantifying data loss events for $w = 2$ weeks. Here we count the percentage of peers that have not been able to restore their data after a local disk crash, averaged over 10 simulation runs. We break down the data loss cases between *incomplete backup* and *failed restore*: the latter case encompasses all cases where peers lose data after completing their backup. Furthermore, we also specify the percentage of *unavoidable* cases in which peers fail before $minTTB$: in this case, not even an ideal system could guarantee a safe backup. Most data loss episodes are simply due to node failure *before the backup is completed*; this result confirms that it is sensible to optimize time to backup by reducing redundancy and hence also network load. In addition, it can be noted that a large majority of data loss episodes are *unavoidable* with any online storage solution: nodes with low bandwidth risk crashing before completing uploads even if saving data to a reliable server. “Failed restore” events – present only in unstable systems with low τ – are imputable to the impact of estimation error on durability, as discussed above. However, we remark that even in such a situation this effect is outnumbered by the unavoidable data loss episodes; this leads us to conclude that nodes with very low lifetime are intrinsically unsuited to any kind of online

storage solution, and not only to P2P backup.

V. CONCLUSION

We focused on P2P backup systems, and designed a redundancy management mechanism tailored to the specific data access patterns that characterize data backup. The goal of our mechanism was to achieve data durability without requiring large redundancy factors nor fast failure detection mechanisms.

Our experiments showed that, in a realistic setting, a redundancy that aims for data durability can be less than half of what is needed to guarantee availability. This results in a system where storage capacity is more than doubled, and backups are much faster (up to a factor of 4) than on a system using traditional redundancy management. This latter property is particularly desirable since, in most of the cases, peers suffering data loss were those that could not complete the backup before crashing. The price to pay for efficient backup was a decreased (but controlled) performance of restore operations.

Finally, we studied data loss events: our results indicated that such events are practically negligible for a mature P2P application in which permanent host failures dominate peer deaths. We also showed the limitations of our technique for a system characterized by a high application-level churn, which is typical of new P2P applications that must conquer user trust.

REFERENCES

- [1] R. Bhagwan, K. Tati, Y. chung Cheng, S. Savage, and G. M. Voelker, “Total recall: System support for automated availability management,” in *USENIX NSDI*, 2004.
- [2] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer, “FARSITE: Federated, available, and reliable storage for an incompletely trusted environment,” *ACM SIGOPS Operating Systems Review*, vol. 36, 2002.
- [3] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran, “Network coding for distributed storage systems,” in *IEEE INFOCOM*, 2007.
- [4] A. Duminuco and E. Biersack, “Hierarchical codes: How to make erasure codes attractive for peer-to-peer storage systems,” in *IEEE P2P*, 2008.
- [5] A. Duminuco, E. Biersack, and T. En-Najjary, “Proactive replication in distributed storage systems using machine availability estimation,” in *ACM CoNEXT*, 2007.
- [6] L. Parnies-Juarez and P. Garcia-Lopez, “Maintaining data reliability without availability in p2p storage systems,” in *ACM SAC*, 2010.
- [7] L. Toka, M. Dell’Amico, and P. Michiardi, “Data transfer scheduling for p2p storage,” in *IEEE P2P*, 2011.
- [8] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, “Do incentives build robustness in bittorrent,” in *USENIX NSDI*, 2007.