Towards reliable backup on the net
by combining markets, swarming and erasure codes

Willem de Bruijn
Vrije Universiteit Amsterdam
wdb@few.vu.nl

***Abstract*— We present *CyberStash*, a blueprint for a wide-area storage network that is both more reliable and more scalable than traditional storage methods. Our principal goal is to make advanced backup technology accessible to regular consumers. The chosen architecture is therefore simple to use and maintain. Because of its global span *CyberStash* can serve not only as a backup tool, but as a cornerstone enabling the vision of pervasive computing [1], [2]. We identify requirements, present an architecture and analytically find a lower bound on reliability that surpasses traditional approaches.**

I. Introduction

A. Motivation

Computer files are often the only product of a process that took up many man-hours. Their value may be high; much higher indeed than that of the medium on which they are saved. Wage increases and storage technology innovations are constantly widening this gap between perceived value and actual storage cost. We therefore expect that people are willing to offer up factors, if not orders, more room than stricly necessary to store a file in return for sharply increased reliability. Large organizations can rely on enterprise systems to secure their data. Current consumer backup strategies, on the other hand, are rigid, simplistic and weak. Even if people are aware of the unavoidable breakdown of hardware their options are often limited to copying a file to a floppy or CD.

Since hardware is fallible, higher-level strategies are needed to increase data safety. Replication is a logical method as it reduces dependence on individual physical devices. This way, minor increases in storage investment bring with it relatively high pay-offs. RAID [3], the de-facto standard for transparent replication, has long been restricted to high-end environments because of cost and complexity. However, as computational cost has dropped significantly software-based RAID has begun seeping into end-user systems. Software RAID is already being shipped with the popular Linux and Windows operating systems and firmware-based solutions are included in many consumer PCs. A drawback of local replication is that it cannot secure against sitewide issues (e.g. power failure or fire). Distributed solutions *can* shield the user from these errors, but were until recently only thought of as enterprise systems. The leap in consumer broadband we're experiencing is opening the way for more personal data replication. Case in point is the popularity of file-sharing networks.

The increasing interconnection and convergence of devices (e.g. PCs, phones, and TVs) brings about a second reason for turning to widely distributed storage. Appliances are more and more turning into access methods to an always-on - or ubiquitous - computing experience. Physically bounded resources do not fit this view. Instead, resources must be accessible regardless of location or device. This is the direction pervasive computing is taking. Distributed storage can serve as building block for such a global fabric, but only if the distributed datastore is conceptually detached from its physical substrate so that information is persistent. For all practical purposes it should also be simple to maintain and resistent to failure, especially when deployed in a public environment such as the internet. In principle, then, it demands the same features as an online backup tool.

Additionally, distributed storage makes more flexible, fine-grained control over storage cost and resultant quality possible. The value of data is subjective and volatile. It fluctuates widely, from next to nothing for a downloaded CD image to towering for a nearly finished manuscript. But the trade-off between storage cost and reliability is made at a coarse grain; adding a backup drive does not exactly consitute tweaking. Ideally we could make the trade-off for each individual object. Scaling upward makes that possible: replication to 10 thousand harddrives increases reliability far beyond what RAID can offer. It comes at a price, of course. But a price that the actors themselves can agree on, unbounded by physical factors. With storage turned into a commodity, matching multiple actors' data to a finite amount of storage space becomes a clearcut scarcity issue. Then, market mechanisms can be used to optimize resource distribution fairly and robustly.

B. Contribution

We investigate the applicability of wide-area networks to reliable storage. Our contribution is twofold. First, we present a simple yet powerful market-based distribution scheme (Section III). Second, we analytically verify applicability of unreliable peers for reliable storage (Section IV). Building on the acquired insights we propose an architecture (Section V) that is geared especially towards regular end-users: *CyberStash*.

II. Related Work

We are not the first to look into transparently distributing data to achieve higher reliability. Available distributed storage systems can be roughly divided into two groups: they are either scalable, but unreliable or reliable, yet oriented at small organizations of trusted peers.

One of the first endeavours into reliable distributed storage was RADD [4], a method of scaling RAID to small networks.

Later initiatives are Zebra [5], xFS [6], and Frangipani [7]. Zebra is notable in this context because it was the first to lack centralization. The two later designs nearly simultaneously extended the design by also removing the need for a central filesystem index. Frangipani - the authors remark - is, through building its filesystem on top of a simpler virtual block device (Petal [8]) less efficient, but easy to implement and maintain. These solutions do not scale well, however. Enterprise network attached storage (NAS) and storage area networks (SANs) fall in this same domain.

Peer-to-peer networks, on the other hand, are highly scalable. But so far they have not been entrusted valuable data because of a lack of (proven) reliability. Academic P2P storage projects, among which are CFS [9], PAST [10], Ivy [11] and pStore [12], show promising results but lack hard security guarantees. Also, we will show that they waste space by using mirroring as replication strategy. Their primary advantage - advanced lookup - goes unused with personal backup.

OceanStore [13] is by far the most mature framework combining the two ideas into a globally persistent filesystem. It is meant to solve many issues and is therefore much more complex than what we propose. Another possibly mature application is HiSpread [14], but little background information can be found about it.

Bolosky *et al.* [15] have conducted a study into the applicability of enterprise networks for peer-to-peer based backup. Their results point to more widely distributing data, as we propose. Another method we propose to use, erasure code based replication, has also been pursued earlier, in OceanStore and Reperasure [16]. Initial results show that software-based erasure encoding can limit throughput [17], but newer specialized codes may help to circumvent these problems in the future (e.g. [18]).

III. METHODOLOGY

To get the best of both worlds we intend to crossbreed the two existing classes of network storage solutions. Enterprise storage systems use advanced replication strategies to trade off reliability for capacity. Peer-to-peer (P2P) networks make data globally accessible and are invulnerable to sitewide failures. Additionally, they have been shown to be simple enough to be used by consumers and are highly scalable.

A. Scaling upwards

We do not intend to build a full-scale network filesystem. Personal backup has a few characteristics that allows us to circumvent a number of hard problems specific to distributed storage. We can safely assume that only large batches of data are stored, that only one user accesses any piece of data and that no updates can occur. Because of these simplications we don't need to worry about sophisticated consistency control and lookup algorithms. On the other hand, backup has as drawback that because data is highly personal there is little sharing involved; a backup network lacks the incentive to cooperate, especially with strangers.

File-sharing networks are, to the dismay of many musicians and filmmakers, proof that duplication can lead to online persistency. They also show that distributed systems are not the privilege of professional organizations. For this reason peer-to-peer technology is a prime candidate for taking consumer backup to wide-area networks. Relying on unknown peers for data replication however also introduces two potential issues: trust and motivation. Trust, it is our thesis, can be achieved by playing the numbers game: with enough redundancy the influence of individual actions on data reliability will be negligeable. Motivation is a different issue. Since remote peers are (ideally) unaware of the data they are storing no real sharing is involved. Why would anyone then want to carry your data? As with all successful P2P uses we must supply an incentive. If possible, the resulting mechanism should also intrinsically discourage fraude.

B. Introducing Markets

Remote peers have no interest in our data as it is (supposed to be) opaque to them. In the handling of this issue lies our first innovation. Recall that (1) we presuppose that people are willing to spare more storage space than necessary for backing up their data and (2) that with a proper incentive in place peers will see no reason to try to break the system.

By assuming that users are willing to increase storage cost beyond the minimal necessary level we can motivate others to participate. In principle, the proposed scheme is to reciprocate: users need to make available as much space as they claim. This base rule can, and will, be extended. But the simple form just given clearly shows the expected reason for success: because a user's data is held hostage by a third party he [1] must follow the rules of the game. His reason for participating in data sharing is not idealistic, but purely pragmatic and focused on maintaining a reasonable level of availability of his own data. Whatever constitutes a reasonable level is decided by the user himself.

The next question is how this system can be implemented. Micropayment schemes have so far had little success on the internet. We propose a direct value-exchange system paralleling cheques. Cheques have the unique trait that there is a single person issuing a cheque and any number of people accepting it as payment. It is not uncommon for a cheque to be transferred multiple times before being cashed in. The advantage of a cheque-based system over full-fledged micropayment is that it is much simpler to implement. With public/private key encryption digital cheques can be signed as easily as their paper counterparts. And by directly reciprocating for issued cheques no trusted third party (i.e. bank) is needed. Cheques, as long as the issuer reaccepts his, can be used to set up for instance our needed 2-way hosting agreement. By being transferable they can also be used in a P2P environment, where the issuer cannot control all actions and no trusted third party (bank) exists.

[1] or she, ofcourse. We can't continuously circumvent the language's gender bias

How do cheques apply to a storage network? Accompanying every block of data is a cheque reciprocating for the same amount. The hosted data serves as a collateral to this cheque. If the user from which data and cheque originate does not accept the cheque he risks removal of its own information.

The scheme becomes more interesting when we drop the 1:1 reciprocation agreement. Perhaps someone's willing to share 10x more diskspace than needed for his backup. In a fully balanced market he will receive the equivalent of 10 remote copies. In an environment where resources are scarce, however, it may be possible that people are willing to share more unused diskspace than they have data to store. Or vice versa. When cheques can be coupled to data blocks in any ratio a simple market can emerge.

Transforming storage into a commodity has had the interest of service-oriented corporations for a while (IBM's Autonomic Computing, HP Utility Data Center), because in a mature market they can request other goods than storage space in return.

A shortfall of micropayments is that their administration overhead often outweights the actual transactions. Cheques, too, would have this problem were it not for the fact that they serve as a limited form of money themselves. Exchange to hard currency can be delayed until sufficient amounts of "credits" have been obtained. The economies of online games (MMORGs) show that this method of linking nearly worthless online activity to a handful of "real" transactions can indeed build a viable market (e.g. [19]).

An additional extension is to introduce explicit service constraints into the cheques. Especially of use in a P2P environment is to be able to notify peers of projected uptime. Then, a node can for instance communicate that it can only make data available during working hours. As we will show later, it is beneficial to reliability of the whole when individual actors clearly communicate their present and future state.

How do we set up the marketplace? The simplest solution is to advertise datablocks together with their cheques. In this buyer's market storage nodes can choose what they feel is the best deal. This is important as data publishers set the price when they tie cheques to data. And it is fully in line with the principle that we must give actors an incentive to cooperate, instead of forcing them through technical means (that can be broken).

There are many open questions and technical details that have to be filled in, for instance related to misuse, forgery and overhead. We will deal with those when discussing the architecture.

IV. Analysis

Because reliability is considered the single most important feature of a backup medium contending technologies must provide an acceptable lower bound on indicative metrics. Traditionally, the Mean Time To Failure (MTTF) for destructive and Mean Time Between Failure (MTBF) for non-destructive failures are used for these purposes. The latter gives, together with the Mean To To Recovery (MTTR), the availability of a medium using the following equation:

$$Availability = \frac{MTBF}{MTBF + MTTR} \tag{1}$$

The probability that a device X is still working at time t is given by:

$$P(alive(X,t)) = exp(\frac{-t}{MTTF_x}) \tag{2}$$

These metrics show that the term reliability conveys more than one meaning. It can be used to mean availability, as in "what is the probability that I can access my data at a random time t?". But, reliability can also be used to denote data integrity, as in "what is the probability that my data still exists at time t?". For backup, the second meaning is more important: timely access is useful, integrity a must.

Data distribution complicates matters, as it may lead to partial availability and thereby to loss of data. We take the view that in backup no errors are accepted, therefore partial integrity is considered complete failure. But is integrity closely related to availability? The main questions to answer are what the availability and integrity metrics for a distributed medium are, how these scale with the size of the network and how they compare with more traditional backup solutions.

A. *Integrity*

Harddrive manufacturers today list Mean Time To Failure metrics of between 300 thousand hours for laptop drives to 1.2 million hours for SCSI server devices. Clearly, the probability that a peer in the network will stay up for this long is next to nothing. More important, though, is that this number signifies Mean Time To Failure for physical devices, but Mean Time *Between* Failure for network nodes. The MTTF of a distributed backup solution is also dependent on another factor: replication rate, i.e. number of hosts carrying the data. Because it is based on an exponential distribution MTTF scales with this value, which we shall call R, as given by the following harmonic series:

$$MTTF_R = MTTF_1 * \sum_{i=1}^{R} \frac{1}{i} \tag{3}$$

Theoretically reachable reliability may be infinite, but MTTF *growth* diminishes with network size, as Figure 1 shows. From this observation follows that a small number of replicas optimizes return-on-investment, indicating the usefulness of RAID arrays. Why, then, do we propose to more widely distribute content? The reasons for this are mostly practical and have already been given: (1) distribution increases robustness against local failure, (2) P2P has a low entry threshold, which makes it suitable for end-users and (3) it makes global persistent storage possible. Without a more efficient replication strategy, however, cost per megabyte is much higher than that of RAID. The erasure-code based solution introduced in Section IV-C has the potential to significantly lower this cost.

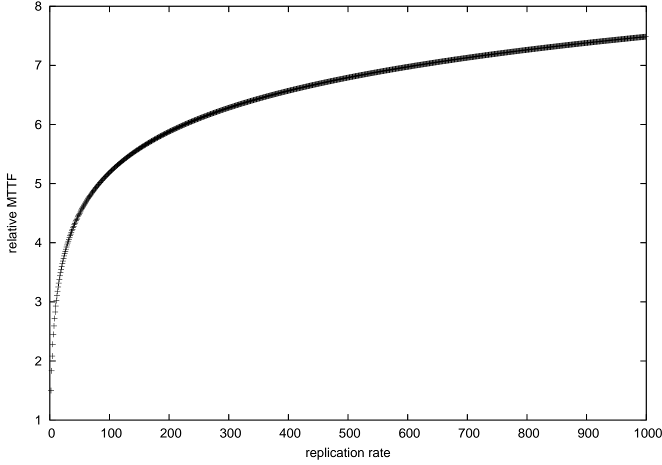

Fig. 1. increase in MTTF relative to number of replicas

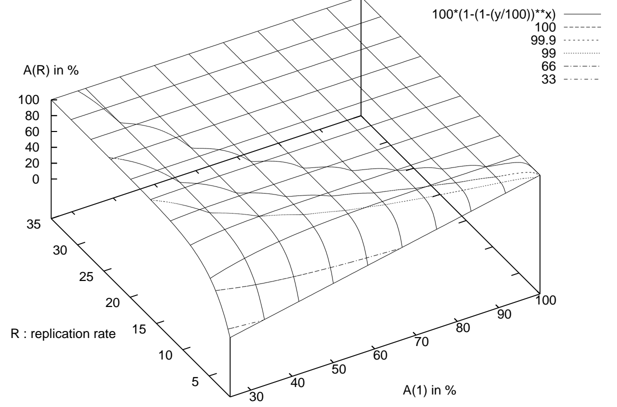

Fig. 2. relative increases in availability

B. Availability

Availability, similar to integrity, scales increasingly slowly. As Equation 1 shows, its range lies between 0 (none) and 1 (full). Regardless of the availability of a single host (A_1), increasing the number of copies R will make availability A_R approach 1. Figure 2 shows A_R in relation to availability of a single device A_1 and the replication rate. From it we can learn that A_R scales linearly with increases in A_1, but logarithmically with R. The lines plotted on the surface of the mesh indicate equal levels of availability. From the fact that they are not perfectly round we can deduce that the two variables do not impact availability equally. Increasing A_1 is, in general, the more effective strategy. However, as we cannot always influence base availability - especially not in peer-to-peer networks - the replication rate can be a useful instrument. And, as manufacturing cost often scales exponential with reliability, there is bound to be a point where adding a few nodes (linearly scaling cost) is cheaper than increasing per-node reliability.

When discussing availability the "rule of the 5 nines" is often mentioned, meaning that achieving 99.999% availability is considered good enough for many purposes. The earlier mentioned MTTF for physical drives ($3*10^5-1.2*10^6$ hours), coupled to an MTTR of a few hours results indeed in numbers between 99.99% and 99.999%. From Figure 2 we can see that approaching this level with only a low value for A_1 takes many nodes. Equation 4 gives the formal relation, whereby U stands for unavailability.

$$
\begin{aligned}
A_n &= 1 - U_n \\
U_n &= (U_1)^n
\end{aligned} \tag{4}
$$

As an example we calculate how many nodes are needed to reach the 5 nines when the average uptime is 8 hours per day, not uncommon for desktop PCs. As Equation 5 shows, under this assumption we need 29 replica's to reach the same level of availability as a single physical drive. However, we have implicitly assumed that uptimes are uniformly distributed. This assumption is questionable, especially for highly homogeneous environments, such as campus networks, where nodes show highly correlated behaviour over time [15]. It is more valid at larger scales, especially ones that include multiple timezones. Intelligent placement can further alleviate the negative consequences incurred by a skewed distribution.

$$
\begin{aligned}
U_1 &= \frac{16}{24} = \frac{2}{3} \\
A_n &= 1 - (\frac{2}{3})^n \\
A_n &>= 0.99999 \Leftrightarrow n >= 29
\end{aligned} \tag{5}
$$

C. Introducing erasure codes

We've seen that untrusted media such as P2P networks can be turned into reliable storage space by introducing redundancy. But the minimal replication number (R) is high: in the order of 20 to 50 for a low value of A_1. Is there a way to reduce this costfactor while retaining the same level of reliability, not counting storage reclaim through compression? There is: using erasure codes.

So far we've only looked at mirroring for replicating data. Mirroring is useful because it is intuitive and simple to implement. It lies at the basis of RAID level 1 and is an efficient 2 disk solution. But beyond 2 disks more efficient strategies can be used. RAID level 5, a popular mechanism, manages to reduce storage overhead significantly compared to RAID1 by using parity-data; it calculates the exclusive OR of all D datawords and stores the resultant $D + 1$ words. *Any* D of the $D + 1$ words now suffice to reconstruct the original data, making the system tolerant to one failure. RAID6 extends this idea by saving 2 different parity words and consequently allows recovery from 2 concurrent failures.

In the general solution the parity/data ratio $r_{(P:D)}$ is fully adjustable. Known as `Reed-Solomon` (RS) encoding, this block-level error-correction mechanism is used often in data transmission, from compact disk recording to deep space communication. RS codes are just as applicable to storage as to transmission, as storage can be seen as nothing more

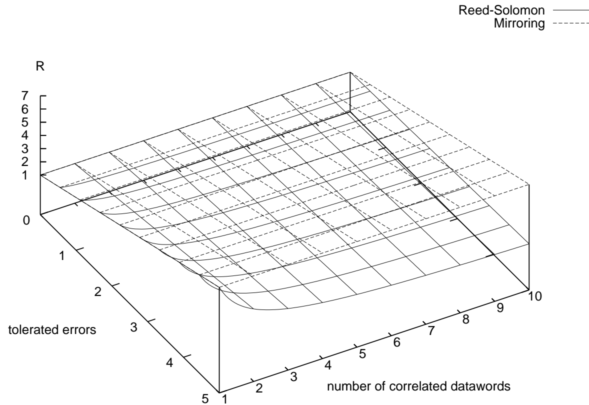

Fig. 3. influence of fault-tolerance on data replication

than freeze framed transmission. The embedding in CDs is indicative of this point.

RS codes are members of a class of encodings called *erasure codes*. Other members have better performance, more closely approach Shannon's theoretical limit on optimal information transfer (the Shannon-Hartley theorem [20]) or are less focused on burst errors. Because computational complexity appears to limit applicability [17], one especially effective code might be LEC [18], as it is much more conservative in this regard than others. We will not go further into which of these codes is best at the moment, however. Instead, we will paint an admittedly overly simplified picture by presenting only the efficiency increases obtainable by using Reed-Solomon codes.

Figure 3 shows how data correlation relates to replication efficiency. For each point in the plot the total number of words is $R = Data + Parity$, but we vary $r_{(P:D)}$ and level of fault-tolerance F. $r = 0 : 2$ corresponds to mirroring, while $r = \frac{1}{F} : 1$ gives RAID5. Both of these can recover from a single failure. The latter, however, needs $D - 1$ fewer words than the first to achieve this. When using as much space for error-correction as mirroring we can even calculate $P = D$ unique recovery words, allowing recovery from D errors. Now fault-tolerance scales linearly with D, not with R! Computational complexity sets a practical limit on the size of the correlated dataset D.

Figure 3 shows that the efficiency gain obtainable by using RS over mirroring not only grows with D, but also with F. This is beneficial in a volatile environment where F must be high, such as a P2P network. At the same time P2P networks makes it easy to reach high values for D. In general we need $D+P-P = D$ chunks to be able to reconstruct all D original datachunks. The trick that enables P2P storage is that we don't necessarily need the original D chunks. With erasure codes we can more widely disperse data, thereby increasing the chance of recovery, as $D + P$ now equals R. So, for D datachunks, we can have $2D$ chunks. Dispersing them ideally means over $2D$ hosts. With the original setup we would now have exactly two replicas of all chunks ($R = 2$) and therefore only two hosts would need to go offline to make some data irretrievable. With erasure codes $D + 1$ hosts must go offline before this happens. From this follows that with D large enough, e.g. 100, downtimes of single hosts are hardly noticeable. A second benefit is that both $r_{(P:D)}$ and D are explicit and easily adjustable, making the whole framework tunable.

How is integrity affected by this solution? With a parity-based solution $MTTF_R$ is no longer directly dependent on the number of replica's, but on the minimal number of words needed to recover all the data. With $D = P$, minimally necessary recoverable data is only 50%. Recall from Figure 2 that 50% *availability* can be satisfied with only a handful of hosts. However, with so few hosts the value can rise and shrink sharply. A sharp decline, even for a short while, can result in dataloss when a peer never returns. A more stable solution is achieved by spreading data over as many hosts as possible. Wide dispersal of chunks is beneficial because the law of large numbers increases resilience against site-specific issues and reduces availability variability.

Contrary to intuition, erasure codes have been shown to be less efficient than mirroring when peer availability A_1 and R are both low [21]. But when users offer up factors or orders of diskspace more than necessary - as we assume - erasure codes are indeed an efficient choice.

V. Architecture

CyberStash builds a network from equal hosts. The architecture therefore contains two parts: the software to build the hosts and the interconnections between them. We will first describe the communicate patterns, and thereafter individual peer details. We conclude by discussing the e-cheque framework.

A. Interconnection

Peers communicate with one another over an unreliable network. Figure 4 displays the communication paths needed to support the mechanisms identified in the methodology. Which type of network then best suits our needs? Peer-to-peer networks can be largely subdivided into two groups: search-oriented networks and data-oriented networks. We are interested in the second, *swarming protocols*, as they are concerned with copying. Swarming's novelty lies in peer cooperation during transfer. Simply put, when n peers want to fetch a file from m servers they coordinate their work to each copy different parts. At the same time they make available already downloaded parts, becoming servers in themselves.

The practice is interesting because it exploits available parallelism more than traditional client-server models do, thereby increasing throughput. It also has some features that uniquely serve our goal. First of all, swarming relies on "chunking", breaking up data into smaller pieces reminiscent of traditional file systems. Second, it is great for replication as chunk exchange is baked into the protocol. Swarming lacks lookup mechanisms. But that should not worry us as metadata-exchange is simply not necessary for *personal* backup; linking files to chunks and locating remote chunks are tasks carried out solely by the data-owner. The best known implementation of swarming is BitTorrent (BT).

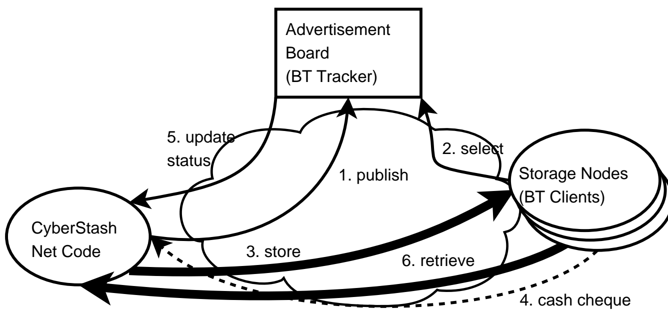

Fig. 4. network topology architecture

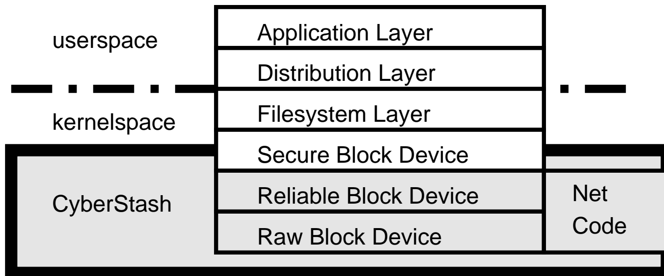

Fig. 5. software stack

Chunks continuously move around. Communication of metadata between peers is handled by posting updates to and reading status from a centralized advertising board. These boards, called ”trackers” in BT lingo, are simple webpages, one for each logical data-unit (file). As shown on Figure 4, trackers are the objects used to create the illusion of data persistency. The client can retrieve his data from anywhere in the world by connecting to his tracker.

B. Software Stack

Building a storage infrastructure involves dealing with many distinct issues, not just reliability. To keep these concerns separated we opt for a layered software stack. The *CyberStash* stack is shown in Figure 5. Its basic building block is a virtual disk, similar to Petal [8]. On top of that lies a redundancy-increasing layer that implements a reliable virtual drive. Advanced behaviour is layered on top op either of these two disks. Most needs will be application specific, but we identify three that are widely applicable. First, encryption will be widely employed by end-users to keep their private data safe from prying eyes. Second, compression greatly enhances storage efficiency at near-zero cost. Third, grouping into logical units (files) is so common that storage simply demands it.

1) Transport Layer: The transport layer implements a virtual block device. Block devices are much simpler than file systems; the set of actions that they must be able to carry out is restricted: space allocation and deallocation and data storage and retrieval. The vdisk mimics a contiguous storage area through its interface. Internally it connects discontiguous globs of data to form this virtual structure. For this scheme to work remote data must be uniquely identifiable (e.g. through URLs), but *there is no need for a contiguous global block address space*. Removing that obstacle greatly simplifies implementation. On the other hand, it ties translation of virtual to physical locations to individual block devices, thus voiding any chance of building a fullscale global filesystem. As we're dealing with personal storage this is not an issue. A method of amending this shortfall - though suboptimally - is explained in the distribution layer section.

Space allocation, or finding the right place to park data, is handled by the market mechanism described in Section III. Storage and retrieval are no more difficult than standard data transfer, as there is no advantage in using swarming protocols for non-replicated chunks. Deallocation, finally, is based on lazy updating. Cheques are useless when they cannot be cashed. Data deletion, then, is a two part process. In phase one the block device ‘forgets’ a mapping between logical block and physical data. Phase two occurs when a remote party produces a cheque. The cheque will simply bounce, whereafter its carrier deletes the data serving as counterweight.

Reliability metrics greatly improve when nodes signal that they are going to deallocate data [15]. Because there's no incentive in our architecture to signal and many things can go wrong on consumer machines, reliability will not reach optimal levels in practice.

2) Reliability Layer: The reliable block device exports the same interface as its underlying sibling. Indeed, the 4 basic actions it must carry out are eventually all handled by the underlying layer. This layer's only task is parity-data interleaving: translating between raw chunks and reliable chunks. It is mostly concerned with accounting: setting up replication strategies fitting the chosen cost level and optimizing retrieval. Uploading is achieved through job advertising, after which remote nodes use a swarming protocol to fetch the chunks. Downloading is based on a parallelized client/server protocol. As only a subset of peers need to be contacted for content recovery this too can be optimized. This layer duplicates some work also taken care of in the lower layer, e.g. advertising data-chunks. Therefore the two layers are best built in cooperation.

3) Translation Layer: The translation layer orders and applies encryption and compression algorithms (e.g. gzip followed by AES) to fresh chunks and reverts this process upon retrieval. The layer can be implicit, i.e. a manual task, or it can be implemented as a virtual block device itself, exporting the same interface as its sublayers but with different data semantics.

The advantage of using a block device for this layer is that the same techniques can be applied to all types of underlying block devices, not just our virtual devices. Modern operating systems often already natively support transparent encryption and compression of block devices, in which case implementation is straightforward. Additionally, inter-file compression, pioneered within deep archival systems [22], can be incorporated.

4) Filesystem Layer: At some point block addresses must be translated into files: the task of a filesystem. In our model the FS layer must only handle translation between filename/offset pairs and logical blocks, because other FS

mechanisms have a high probability of clashing with the near-inexistent update model of the underlying block device, resulting in dogged performance and high storage overhead.

Updating is not encouraged, although a delete followed by a store can always be used to simulate an atomic operation. The zebra networked filesystem [5] implements updates over a striped network by using a log-based filesystem [23]. By caching small writes and processing these in batches they managed to reach acceptable levels of performance. Stodolsky extends this idea to parity-based block devices [24]. But he does not take the expected high degree of unavailable nodes in a peer-to-peer storage network into account.

Implementing global consistency control is notoriously hard. It is further complicated when dealing with parity words. Because our use-cases don't depend on it we chose to disregard the issue in principle. This is not to say that such methods *cannot* be applied. As the block device is the bottleneck of this system it can initiate pseudo-atomic operations, lock regions and flag dirty blocks (e.g. on offline nodes).

One specific scenario wherein updates would be useful is metadata interleaving. Introduced in xFS [6] and Frangipani [7], metadata interleaving lends the same level of security to the block device's lookup tables as to the stored data itself. A frequent problem with older filesystems, such as FAT, is that lookup tables are easily corrupted. Naturally, that is unacceptible behaviour for a storage device. Distributed allocation tables as we designed them - using BT trackers - are quite volatile, however. Tracker implementation is beyond the scope of this paper, but we will point that althought the reference implementation is centralized there is increasing interest in distributed solutions. Other metadata, such as the reliability layer's replication records, are ideally also distributed. With so much development going on in the area we leave this for future work.

5) Distribution Layer: CyberStash is not a traditional peer-to-peer program. Nevertheless it can aid data distribution through sharing part of its distributed allocation tables. Let's say that file X must be copied. Then the filesystem layer must supply its translation map ($F \rightarrow [B_1, ..., B_n]$) and the block device its lookup tables ($B_x \rightarrow http://...$). With this information anyone can duplicate the partial virtual devices in read-only mode. Since the cheques point to the original owner and only he can sign them the device cannot be shared in read-write mode. Integrity is lost only when this key is compromised.

6) Application Layer: The top, or application, layer is agnostic to the workings of the underlying system. Userspace applications can therefore interact with the filesystem through their normal API. We need to export additional mechanisms, however, to allow users to specificy the acceptible cost of a piece of stored data.

External management is a special class of tasks for which the API should be extended. As analysis pointed out, peer availability may not be uniformly distributed, in which case reliable block availability will also fluctuate. Availability can be kept high by continuously monitoring the state of the storage space and intervening when necessary. Historical trends, e.g. regarding node uptime, can be found, extrapolated and taken into account. For continuous operation this optimization application must live in the pervasive fabric of the net, not on an individual access device. We are in the midst of implementing a dedicated manager that fits this role (BetaGIS [25]).

C. Cheques

Cheque-based micropayment was introduced in Section III. Now we will discuss technical details. For a cheque-based system to work it must satisfy the needs of both the publisher and carrier of the cheque. Satisfying the carrier is the easy part. As he holds the other party's data hostage he has a powerful bargaining position. If his partner proves unreliable he can simply delete the data and reach a new agreement with someone else.

Safeguarding the publisher's rights is not so simple. What sets e-cheques apart from their everyday counterparts is that the initial exchange is made before the task is completed, with a partner that is in principle untrustworthy. Trust needs to be built, for instance through a question and answer scheme: "Tell me what is stored in the nth word of block b_x" or "how much padding is appended to block b_y?". A potential problem with Q&A schemes is that the publisher must be able to verify answers. The more elabore his questions, the more background information he himself must have available. Ofcourse, he can choose to bluf. But so can the other party.

What does an e-cheque look like? It must carry some information only visible to the publisher (i.e. encrypted): the block ID(s) for which it reciprocates and a serial number in case there are multiple replicas. Also, because cheques are transferable they must tell carriers where they can be cashed and state their worth, naturally in unmodifiable form. Finally, they must contain a hash connecting the cheque to a datachunk, e.g. an MD5.

At an abstract level e-cheques are simply transferable contracts. They state an agreement between two parties, whereby one party is not explicitly mentioned. The encoding can be extended in other meaningful ways. In our case it would, for instance, be useful to be able state partial availability, as in "I will try to make your content available between 8am and 5pm every weekday until december 31st". Such contracts are based on the same trust mechanism as the simpler version, but require additional language constructs and therefore additional logic in the chunk selection algorithms of storage nodes. Dedicated management applications (e.g. BetaGIS [25]) can be used to optimize the search strategy.

The last problem we must deal with is misuse. If possible, a distributed system should be built in such a way that little or no trust is required for operation. Has our cheque-based mechanism succeeded? What are obvious weak spots? The main problem lies, as pointed out, with trusting that a node is actually storing your data. We have yet to find a better solution than the one presented. What about vandalism? As there is no single point of failure breaking the entire system will be hard. Freeloading by not accepting your own cheques is discouraged

as remote data is then instantly destroyed. Counterfitting cheques is considered impossible as long as the publisher uses-up-to date encryption tools. Holding someone's data hostage and demanding additional rewards will be unsuccesful as most data is replicated. There are bound to be ways to break the system, but at least the obvious routes appear closed.

VI. Implementational remarks

How difficult is implementing the proposed architecture? By using common off-the-shelf (COTS) parts building a prototype is straightforward. As we have no working prototype we cant't discuss details, but we will give our preliminary vision.

By embedding the virtual devices in the operating system kernel *CyberStash* provides a generic interface. Virtual block-devices are already used extensively, e.g. by Linux's Logical Volume Manager or Software RAID. Because data transfer can be relatively slow the back-end should be looped back into userspace, in a manner similar to userspace filesystems (e.g. FUSE and LUFS). Userspace is definitely preferable for storage daemons and management applications. The first is built on top of an open-source Bittorrent implementation. The second can use a management application like BetaGIS. The market can be implemented quickly by using Bittorrent trackers as advertisement boards. Thus, the whole networking code can be built from COTS parts, as can the security and filesystem layers. Even log filesystem code is readily available. What's left is laying the connection between logical blocks on the one hand and bittorrent objects on the other.

VII. Conclusion

We set out to design a cheap and easy method for bringing reliable storage to end-users. Unable at this point to show practical results, our conclusions are based on *a priori* reasoning alone and hence limited. Analysis has shown that a reliable storage medium can indeed be built on top of a network of untrusted peers. Moreover, the overhead in storage cost is limited, especially when using a distribution method based on erasure codes.

More questionable without hard data backing it is the feasibility of the proposed cheque-based market mechanism. With the advent of virtual worlds (e.g. MMORGs) digital currency has seen a spur of activity. We intend to more thorougly look into on-line payment schemes in the near future after which we hope to be able to give more conclusive results.

All things considered our contribution is only limited; many technicalities will undoubtedly have to be dealt with before a cheap and simple on-line backup medium materializes. But with these broad brush-strokes we hope to have at least cleared a path for others to continue on.

References

[1] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal Communications*, pp. 10–17, Aug. 2001.

[2] D. Saha and A. Mukherjee, "Pervasive computing: A paradigm for the 21st century," in *IEEE Computer*, 2003.

[3] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (raid)," in *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pp. 109–116, ACM Press, 1988.

[4] M. Stonebraker and G. A. Schloss, "Distributed RAID— A new multiple copy algorithm," in *Sixth Int'l. Conf on Data Engineering*, 1990.

[5] J. H. Hartman and J. K. Ousterhout, "The Zebra striped network file system," in *High Performance Mass Storage and Parallel I/O: Technologies and Applications* (H. Jin, T. Cortes, and R. Buyya, eds.), pp. 309–329, New York, NY: IEEE Computer Society Press and Wiley, 2001.

[6] T. Anderson, M. Dahlin, J. Neefe, D. Pat-terson, D. Roselli, and R. Wang, "Serverless network file systems," in *In Proceedings of the 15th Symposium on Operating System Principles. ACM*, (Copper Mountain Resort, Colorado), pp. 109–126, December 1995.

[7] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A scalable distributed file system," in *Symposium on Operating Systems Principles*, pp. 224–237, 1997.

[8] E. K. Lee and C. A. Thekkath, "Petal: Distributed virtual disks," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, (Cambridge, MA), pp. 84–92, 1996.

[9] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, (Chateau Lake Louise, Banff, Canada), Oct. 2001.

[10] A. I. T. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *Symposium on Operating Systems Principles*, pp. 188–201, 2001.

[11] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, "Ivy: A read/write peer-to-peer file system," in *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, 2002.

[12] C. Batten, K. Barr, A. Saraf, and S. Treptin, "pstore: A secure peer-to-peer backup system," 2001.

[13] J. Kubiatowicz, D. Bindel, Y. Chen, 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 *Proceedings of ACM ASPLOS*, ACM, November 2000.

[14] "Hispread homepage." http://www.hispread.com/. as available on January 24th, 2005.

[15] 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," *SIGMETRICS Perform. Eval. Rev.*, vol. 28, no. 1, pp. 34–43, 2000.

[16] Z. Zhang and Q. Lian, "Reperasure: Replication protocol using erasure-code in peer-to-peer storage."

[17] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: the oceanstore prototype," in *proceedings of FAST'03*, 2003.

[18] J. Cooley, J. Mineweaser, L. Servi, and E. Tsung, "Software-based erasure codes for scalable distributed storage," in *Proceedigns of 20th IEEE/Nasa MSS*, 2003.

[19] "Virtual island sells for $26.500." http://games.slashdot.org/article.pl?sid=04/12/14/1759253. as available on January 24th, 2005.

[20] C. E. Shannon, "A mathematical theory of communication," *The Bell Labs Technical Journal*, vol. 27, pp. 379–457,623,656, 1948.

[21] W. K. Lin, D. M. Chiu, and Y. B. Lee, "Erasure code replication revisited.," in *Peer-to-Peer Computing*, pp. 90–97, 2004.

[22] L. You and C. Karamanolis, "Evaluation of efficient archival storage techniques," in *Proceedings of 21st IEEE/NASA Goddard MSS*, 2004.

[23] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems*, vol. 10, no. 1, pp. 26–52, 1992.

[24] D. Stodolsky, G. Gibson, and M. Holland, "Parity logging overcoming the small write problem in redundant disk arrays," in *ISCA '93: Proceedings of the 20th annual international symposium on Computer architecture*, pp. 64–75, ACM Press, 1993.

[25] W. de Bruijn, H. Bos, and H. Bal, "robust distributed systems: achieving self-management through inference," in *Proceedings of WoWMoM'05*, 2005.