Also publ. in: Third IEEE International Conference on Peer-to-Peer Computing (P2P 2003), September 2003

Establishing Trust in Distributed Storage Providers

Germano Caronni and Marcel Waldvogel

***Abstract*— Corporate IT as well as individuals show increasing interest in reliable outsourcing of storage infrastructure. Decentralized solutions with their resilience against partial outages are among the most attractive approaches. Irrespective of the form of the relationship, be it based on a contract or on the more flexible cooperative model, the problem of verifying whether someone promising to store one's data actually does so remains to be solved, especially in the presence of multiple replicas. In this paper, we introduce a lightweight mechanism that allows the *data originator* or a dedicated *verification agent* to build up trust in the *replica holder* by means of protocols that do not require prior trust or key establishment. We show how naive versions of the protocol do not prevent cheating, and then strengthen it by adding means that make it economically attractive to be honest. This provides a foundation for further work in providing trustworthy distributed storage.**

I. Introduction

The outsourcing of storage infrastructure is becoming increasingly interesting for corporate IT as well as for individuals. Besides the use of tainted peer-to-peer (P2P) content sharing systems such as Gnutella [1], the appeal of completely decentralized storage is steadily increasing. Ranging from niche applications such as censor-resistant publishing (e.g., Freenet/Eternity [2], [3] and Publius [4]), to the scalable use of promising overlay networks based on distributed hash tables (DHT) [5]–[9], the potential of storing Petabytes of data accessible whenever and wherever required has spurred great enthusiasm.

The advent of DHTs has given such systems a strong boost, as DHTs allow an efficient, scalable, and often failure-tolerant addressing of stored documents, including decentralized replication mechanisms [10], [11]. These and similar file distribution and storage mechanisms also make sense in more traditional scenarios, ranging from distributed backup (as e.g. marketed by HiveCache or Permabit) and mirroring facilities to distributed storage facilities for gridware [12] or a gridware-like environment.

There is, however, one crucial problem with independent agents holding copies of your data: How can you make sure that they really store a copy locally, and not just claim to do so? The existence of this problem becomes immediately evident in collaborative environments. Even in the presence of contracts, how can you verify that the storage providers are actually providing the promised number of replicas? If you ask them to provide you with some file content, they can easily forward that request to another replica holder without your knowledge.

Ignoring performance issues, the most immediate risk here is that replicas of your data are being retained in fewer places than you ask for. In the extreme, this can lead to the existence of a single point of failure, exactly the scenario distributed storage eagerly tries to avoid. If storing your data gives replica holders the right to store some of their data at your place, a financial compensation, or some other kind of tangible benefit, the economic incentive for falsely claiming to store your data becomes clear.

A. Our Contribution

In this paper, we introduce a mechanism that returns power to the *data originator*. It allows the originator (in fact, anyone) to establish trust into the *replica holder*. We show that naive approaches are susceptible to cheating by the replica holder. Our proposed lightweight protocol allows building up trust, even without a prior phase of authenticated key exchange and trust establishment, both major criteria in collaborative systems. The resulting protocol does not provide a direct proof of storage, but requires dishonest replica holders to use significantly more resources than an honest replica holder would have to use. It does not, however, prevent malicious, resource-rich entities from performing their intentionally malicious goal at considerable expense. Using appropriate cost functions, the protocol provides the necessary leverage against "lazy" replica holders, whose goal it is to obtain a seizable advantage and which we expect to be the common cheating candidate. Our symmetrical protocol limits the possibilities of Denial-of-Service (DoS) attacks.

B. Organization of the Paper

In Section II, we first introduce and then improve methods that help verify whether a replica holder actually keeps its promises. Section III describes how to minimize

G. Caronni is with the Zurich Information Security Center (ZISC) at ETH Zurich and with Sun Microsystems Inc., Sun Laboratories. He can be reached at `gec@acm.org`. M. Waldvogel is with IBM Research, Zurich Research Laboratory, and can be reached at `mwl@zurich.ibm.com`.

the impact of DoS attacks. In Section IV, we present related work, and in Section V, we draw our conclusions.

II. Verification of Content

Standard solutions to verify whether a replica holder has a copy of some file are to either ask it to send the file back to the verifier, or to compute a hash value over the file (a keyed hash with a verifier-defined key is used if a proof of freshness of the computation is required). Unfortunately, they require considerable host bandwidth (storage-to-CPU) and network bandwidth (former case) or CPU power (latter case) for each request. The exclusive use of optimizations such as hash trees over the file are not appropriate, because the supposed replica holder can just precompute them, thus obviating the need for storing the entire document.

A viable compromise is to have the verifier request the hash value to be computed only over a chunk of the data at one time, with the chunk being selected at random. Thus a smooth trade-off between full verification and partial verification as well as between CPU/disk load and bandwidth usage is achieved.

Let us now take this as the most naive practical starting point. By considering the different ways in which a replica holder might cheat, followed by appropriate adaptations to our protocol, this initial idea will steadily evolve into a simple, efficient, and secure protocol.

A. Options of Dishonest Replica Holders

As mentioned above, a replica holder can try to cheat in many ways. It can

1) pre-compute information and only remember that data,
2) forward the request for verification to another unsuspecting replica holder,
3) forward the request to a colluding replica holder, or
4) download the file or parts of the file needed for verification from another replica holder on demand.

We will discuss these attacks and appropriate countermeasures in the following subsections. Most items deal with the possibility of smart fraudulent replica holders trying to obtain full benefits of the "contract" while delegating the costs to unsuspecting third party. Unfortunately, there are clear economic benefits of such parasitic behavior; fortunately, there are also efficient technical countermeasures.

Item 3 above is the exception to the parasitic behavior, differing substantially from item 2. Technical countermeasures against such symbiotic collusion are rather limited, but so are the incentives, fortunately.

Step	**A**		**B**
1	$A, R_1, H(R_1 \| R_2)$	$\longrightarrow$	
2		$\longleftarrow$	$B, R_3, H(R_3 \| R_4)$
3	R_2	$\longrightarrow$	
4		$\longleftarrow$	R_4, K

R_i are random values; A, B are not easily forgeable identities to deter work delegation. The result of the protocol, the key K, is defined as $K = H(A \| B \| R1 \| R2 \| R3 \| R4)$.

Fig. 1. Key exchange protocol

B. Precomputation

The attempt using pre-computed data is most easily thwarted by making all computations over the data involve some fresh nonce as the key to a Message Authentication Code (MAC) [13] or by modifying the range of data to be verified sufficiently (see also Section II-E).

C. Delegation to Unsuspecting Replica Holders

One way to counter this would be to have the data originator produce "personalized" replicas for each holder, e.g. by tying them to the identity of the replica holder.[1] However, this would require the underlying replication architecture to be strictly star-shaped, with the originator being the hub: no replica holder would be able to forward its copy to somebody else, without going first through the data originator. This contradicts the goal of avoiding single points of failure and performance bottlenecks.

A far more effective, but still simple solution is to use a key for a MAC that is derived from random inputs of both verifier and replica holder in a secure fashion. In this way, neither party can *a priori* force a result with specific parameters. Therefore, the replica holder cannot forward the request, as a delegation attempt would create a different key. The initial agreement can be kept simple because no confidentiality or authenticity issues exist; a man-in-the-middle trying to hijack the protocol at worst can persuade the verifier that the replica holder is uncooperative, an accusation that can be achieved more easily by simply dropping the packets (Section III).

A sample key agreement protocol is shown in Figure 1. Note that B's messages are not chained to A's messages, until the key is sent back. The inclusion of the identities of A and B is required to prevent a dishonest replica holder B from transparently forwarding traffic between A and C. Sending back the key K is not necessary, as both parties are able to calculate its value. However, exchanging K in or after step 4 helps determine protocol errors before more resources are spent.

[1] Identities, as used in this paper, might be as simple as the addresses used by the communications protocols, e.g., IP addresses. They are neither required to be cryptographically strong nor signed by a central authority.

TABLE I

LATENCY AND BANDWIDTH COMPARISON

		Disk	LAN	MAN	Intra-	Inter-
					continental	
Latency	[ms]	10^a	1	10	50…100	100…400
Bandwidth	[Mbps]	250…500	10…1000	1…100	1…10	0.1…10

[a] The disk time is for a random seek; track-to-track seek is around 1ms. As disks typically buffer at least an entire track, seek times within a track become basically negligible.

D. Delegation to Colluding Replica Holders

In Section II-C, we have seen how to prevent a dishonest replica holder from abusing an honest one. But the problem becomes much more difficult if the two replica holders actively collaborate and share information beyond what is exchanged in the protocol in Figure 1. This might include having random number generators running in lockstep, exchanging the random values R_3 and R_4, or allowing the key K to be set without going through the key exchange.

In general, it seems difficult to even identify any willful collusion between replicas. The available options are either extremely expensive or do not seem effective. They are listed below mainly for completeness.

1) Detecting the additional message delay (this seems impossible to detect even if all nodes and links provide hard real-time guarantees; even in that case, it is likely that verification would require immense resources for all parties involved[2]).
2) Having each replica hold a different version of the document (this would remove all incentives, but see the discussion in Section II-C above).
3) Having the computation include another piece of information that the replica holder would not want others, including the co-conspirators, to know (but why would this information be shared with the verifier?).

Instead, we believe in non-technical measures against collusion. Economic incentives toward collusion seem to be limited, and are likely to be outweighed by the risks incurred. Consider the following example:

A partnership of colluders provides a substantial portion of the storage infrastructure, say, 10%. In addition, assume that the typical customer requests two replicas. Given a random selection of replica holders, the dishonest service could be offered at a storage savings of about 10% of their storage infrastructure, and no bandwidth savings (which currently is more costly). We believe that the savings obtained are not worth the risk of suddenly being put out of business because of fraudulent behavior. A company owning the sizable business of 10% of the total storage market should not incur this risk lightly.

Therefore, the incentives at work against collusion in both large (business risk) and small (no profit gain) partnerships are expected to support honest behavior.

E. Download on Demand

Whenever a verifier requests a check from a replica holder, the replica holder could theoretically request the specific data to be covered by a check from other replica holders. This works well when the range to be verified is mostly contiguous and relatively small. Otherwise, the efforts and bandwidth spent by the fraudulent replica holder in downloading the data are large compared with the storage space saved.

The predicament for the replica holder can be made even stronger, by making the range to be verified consist of several hundred very small chunks (e.g. only a single byte), each residing a random distance apart from the previous chunk. The selection of actual distances should discourage downloading large chunks. The best distance would be such that issuing a separate request for each chunk of a pair is considered at least as costly as requesting a range that includes both chunks. Taking into account packet headers and other overhead, the traffic would greatly exceed the number of bytes over which to checksum.

If the bytes are sufficiently close (Table I), the replica holder will not even incur many disk-seek penalties.[3]

As can be seen from typical disk and network parameters as listed in Table I, accessing local disks is significantly faster than network accesses, both in terms of bandwidth and latency, for all except local replicas. The main differentiating factor is latency, unless huge

[2] A potential way to do this is to reveal only half of R2 and R4, and have the peer use these revealed halves, while the other party uses the non-revealed halves, and reveal them only after a sufficient number of verification steps. This has the added benefit that both the replica holder and the verifier need to perform some work that the other can then check on.

[3] The verifier could even analyze the delays incurred by the replica holder when answering verification requests and try to derive a model of the disk parameters, e.g. rotational delay and cylinder or cache size, by using techniques similar to the DiskSim [14] parameter discovery.

Listing 1 First-cut verification procedure

```
Verifier and replica holder agree on common key (Figure 1)
Verifier specifies chunk size, maximum step size, and upper
  number of steps.
Replica holder and verifier seed their stepping function
  (RC4) with the key
current position ← 8 bytes out of RC4 (modulo file size)
while number of steps < upper number of steps do
    value ← chunk at current position in file
    insert value into hash
    step size ← (value + 8 bytes out of RC4) (modulo
      maximum step size)
    next position ← (current position + step size) (modulo
      file size)
end while
```

data blocks are to be transferred. Latency hiding through pipelining of requests can be efficiently prevented by reverting to a data-dependent step function, in which the address of the next item to be checksummed becomes available only after the current data value has been incorporated into the checksum result.

This procedure can be used to differentiate between honest replica holders that access the document from a disk and dishonest replica holders that access it over a network.

F. *Resulting Verification Process*

As a first cut, consider the verification procedure in Listing 1. Given the amount of randomness involved in the selection of bytes to verify, we cannot conceive of any way a dishonest replica holder would use this to precompute and store information about the verification value. Thus we suggest to use an inexpensive checksumming function such as Adler-32 [15], instead of a more expensive cryptographically strong hash function.

Using Listing 1 with the data provided in Table I, we need about 1 s of disk drive time (probably much less CPU time) to check some 30 MB available on the local disk, independent of the number of samples, assuming a contiguous layout of the file. Over a network (50 ms at 1 MB/s (∼10 Mb/s), which is roughly the rate with which medium-sized enterprises connect to the Internet), it takes 30 s to download everything; the same time is necessary when requesting 1200 data-dependent samples.

Even at higher bandwidths, the replica being exploited also needs to have the same amount of bandwidth available and the dishonest replica needs to be willing to waste this (expensive) bandwidth to save some (cheap) disk space. By using this scheme, we can easily remove any economic benefits that cheating may have entailed.

III. Denial-of-Service Considerations

The verification protocol as outlined so far is an excellent way to run a DoS attack on replica holders: Just have them perform unlimited verification operations. That will keep them busy and unable to provide their normal service. Although this situation can be prevented by intricate identity and ownership management of files, we choose not to explore this direction. Instead, we keep things simple and again base trust only on verifiable behavior. We balance costs such that a verifier has to spend at least as much effort on continued verification processes as the replica holder does.

The first mechanism to limit DoS opportunities against replica holders is by giving the replica holder the option of requiring the verifier to perform a hash-cash [16], [17] operation. Thus, the verifier has to "pay" for the verification with CPU cycles. The amount of hash-cash will be defined by the current load experienced by the replica holder. The result of the calculation is a ticket of limited lifetime that grants access to the actual verification. Should the replica holder repeatedly pose impossibly high hash-cash challenges, the verifier can assume the replica holder does not want to comply with the protocol.

When replica holder and verifier are in symmetric positions, i.e., both are interested in verifying each other's content, then a second ticket-granting mechanism becomes available: A successfully executed verification of one party will grant this party the right to request a slightly larger verification from the other side by issuing an appropriate ticket. Here, both sides actually perform useful work, eliminating the waste of CPU cycles performed by hash-cash.

In effect, mutual work (through hash-cash, reciprocal verification, or a combination thereof) will be used to establish a lasting trust relationship.

A man-in-the-middle attacker can use the proposed mechanism for DoS by falsifying or suppressing the messages exchanged (see also Section II-C). Even though this attack is typically not controlled by the replica holder, the replica holder effectively becomes an unreliable storage provider. The use of overlay networks may help circumvent such packet dropping or modification attacks, but this is beyond the scope of this paper.

A possible implementation of the resulting protocol is sketched in Table II. To avoid inflation, issued tickets may be worth a little less than what they were offered for, and protocol initiation may lead into a small hash-cash challenge being required in addition to consuming a ticket.

TABLE II

POSSIBLE IMPLEMENTATION OF THE DOS PREVENTION PROTOCOL

Verifier	Replica Holder
Initiate protocol by requesting ticket for a given verification cost (i.e. number of steps in verify operation).	
	Note that no trust relationship exists so far. Ask for hash-cash appropriate for the requested verification and current load on holder.
Compute response and send.	
	Compute ticket, containing verifier identity (e.g. IP address), serial number, end-time stamp, and HMAC with a secret key, only known by holder. Send ticket.
Issue first verification request, including agreement on K, and ticket.	Receive verification request, note that K is correct. Verify ticket. Note ticket as used.
Optionally provide H(H(R)), to allow the replica holder to know that the verifier has also perfomed the verification operation.	Of significance later.
	Compute requested verification, and reply with H(R). If verifier provided H(H(R)), then also issue a ticket to facilitate the next round, since this proves that the verifier invested the same amount of work.
Receive and validate response. Store ticket for future use.	
	Optional: Issue verification request and if successful, issue ticket to verifier for the next round.

IV. RELATED WORK

While some of the work mentioned in the introduction (i.e. The Eternity Service [3]) has considered the issue of replica holders being able to cheat, the proposed solutions are somewhat simplistic. Most commonly, a MAC or chunk of the file is requested. As outlined above, the benefit of requesting a chunk out of the file is that the replica holder can not distinguish between verification checks and actual downloads. The same observation has already been made by Free Haven [18]. This property needs to be balanced against the increase in bandwidth consumption that is caused by the verifications.

Trust has been a research topic for decades [19], ranging from agreements even in the presence of untrusted entities [20] to the total trust in a peer in a web-of-trust [21] setting. In practice, however, trust is often handled through centralized hierarchies: Everyone ultimately trusts a single entity, which in turn delegates some of its trust to other principals, who may or may not have the right to delegate this further.

As all of these systems have weaknesses, people have started working on other, more immediate ways to deal with this issue, namely, reputation systems [22]. Advogato [23] uses a hybrid system in which users can rank their peers; the overall reputation of a ranked individual then depends on the result of a network flow calculation. Mobile ad-hoc networks are a prime area of reputation research to determine whether intermediate nodes actually do forward the packets or prefer to behave egotistically and instead conserve their own power by not helping the others [24], [25]. In this domain, it is relatively simple to see the result, either through reciprocal reception of the radio signal, through the help of other nodes, or by seeing communications progress and getting the desired answer from the communication peer. Systems such as CONFIDANT [26] use reputation gossip to augment their first-hand experience.

V. CONCLUSIONS

In this paper we have presented a first algorithmic approach at fairly verifying whether replica holders indeed perform the service they promised. Our protocol is based on a checksum or hash that is calculated over key-defined ranges of shared data. This check is performed in an iterative fashion with alternating roles, or compensated by the calculation of responses to challenges to prevent DoS attacks. At the same time this builds a trust relationship between replica holder and verifier which can be reused in later rounds of the protocol.

To the best of our knowledge, this is the first paper raising the issue of fair verification of stored data in a non-trivial fashion. Given the increased interest in (potentially massively) distributed storage, the need for a lightweight mechanism is well covered in the protocol developed herein. We believe that it will give first answers to some of the issues that have arisen in the peer-to-peer and distributed storage communities, but also raise new questions and challenges. Open issues include tightening many of the loose ends and gathering experience in a real environment.

REFERENCES

[1] Eytan Adar and Bernardo A. Huberman. Free riding on Gnutella. *First Monday*, September 2000.

[2] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. http://freenetproject.org/, 1999.

[3] Ross J. Anderson. The Eternity service. In *Proceedings of Pragocrypt '96*, 1996.

[4] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. 9th USENIX Security Symposium*, pages 59–72, August 2000.

[5] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, September 2001.

[6] Anthony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.

[7] Ben Y. Zhao, John Kubiatowicz, and Anthony Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley, April 2001.

[8] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001*, pages 149–160, San Diego, CA, USA, August 2001.

[9] Marcel Waldvogel and Roberto Rinaldi. Efficient topology-aware overlay network. *ACM Computer Communications Review*, 33(1):101–106, January 2003. Proceedings of ACM HotNets-I (October 2002).

[10] John Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*, November 2000.

[11] Marcel Waldvogel, Paul Hurley, and Daniel Bauer. Dynamic replica management in distributed hash tables. Research Report RZ–3502, IBM, July 2003.

[12] Sun cluster grid architecture. `http://www.sun.com/software/grid/whitepapers.html`, 2002.

[13] Gene Tsudik. Message authentication with one-way hash functions. *ACM Computer Communication Review*, 22(5):29–38, 1992.

[14] Gregory Robert Ganger. *System-Oriented Evaluation of I/O Subsystem Performance*. PhD thesis, University of Michigan, Ann Arbor, 1995. Also available as technical report CSE-TR-243-95.

[15] Peter Deutsch and Jean-Loup Gailly. ZLIB compressed data format specification version 3.3. RFC 1950, Internet Engineering Task Force, May 1996.

[16] Adam Back. Hashcash—A Denial of Service Counter-Measure. `http://www.cypherspace.org/∼adam/hashcash/`, 1997.

[17] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *IEEE Symposium on Foundations of Computer Science*, pages 90–99, 1991.

[18] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. *Lecture Notes in Computer Science*, 2009:67–??, 2001.

[19] Matt Blaze, Joan Feigenbaum, John Ionnidis, and Angelos D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, pages 185–210. Springer Verlag, 1999.

[20] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[21] Ueli Maurer. Modelling a public-key infrastructure. In *ESORICS: European Symposium on Research in Computer Security*. LNCS, Springer-Verlag, 1996.

[22] Paul Resnick, Richard Zeckhauser, Eric Friedman, and Ko Kuwabara. Reputation systems. *Communications of the ACM*, 43(12):45–48, 2000.

[23] Advogato's trust metric. `http://www.advogato.org/trust-metric.html`, February 2000.

[24] Pietro Michiardi and Refik Molva. CORE: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *Sixth IFIP Conference on Security Communications, and Multimedia (CMS 2002)*, Portoroz, Slovenia, 2002.

[25] Krishna Paul and Dirk Westhoff. Context aware inferencing to rate a selfish node in dsr based ad-hoc networks. In *Proceedings of the IEEE Globecom Conference*, Taipeh, Taiwan, November 2002. IEEE.

[26] Sonja Buchegger and Jean-Yves Le Boudec. Performance Analysis of the CONFIDANT Protocol: Cooperation Of Nodes—Fairness In Dynamic Ad-hoc NeTworks. In *Proceedings of IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, Lausanne, June 2002.