An Alternative Information Plan

Matteo Monti
Steen Rasmussen
Marco Moschettini
Lorenzo Posani

SFI WORKING PAPER: 2017-07-021

An alternative information plan

Matteo Monti[abc1*], Steen Rasmussen[ad*], Marco Moschettini[c2] and Lorenzo Posani[e]

[a]Center for Fundamental Living Technology (FLinT),
University of Southern Denmark (SDU), Denmark;
[b]École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland;
[c1]Complex Systems Group, University of Bologna, Bologna, Italy;
[c2]Department of Computer Science and Engineering,
University of Bologna, Bologna, Italy;
[d]Santa Fe Institute (SFI), New Mexico, USA;
[e]Laboratoire de Physique Statistique, École Normale Superiéure, Paris, France

[*]Corresponding authors. Emails: matteo.monti@msoftprogramming.com, steen@sdu.dk

June 27, 2017

Abstract

We present and evaluate an alternative architecture for data storage in distributed networks that ensure privacy and security, we call RAIN[1]. The RAIN network architecture offers a distributed file storage service that: (1) has privacy by design, (2) is open source, (3) is more secure, (4) is scalable, (5) is more sustainable, (6) has community ownership, (7) is inexpensive, (8) is potentially faster, more efficient and reliable. RAIN has the potential to democratize and disrupt cloud storage by eliminating the middle-man: the large centralized data centers. Further, we propose that a RAIN style privacy and security by design architecture could form the backbone of multiple current and future infrastructures ranging from online services, cryptocurrency, to part of government administration.

Keywords: distributed storage; privacy by design; internet of things; community ownership; sustainability; bio-inspired design

[1] RAIN is a metaphor for what comes after the clouds.

Part A
Introduction

A.1 Background

The Internet data storage services provided today violate privacy, are expensive, and come at a high environmental cost. More than 3% of the world's power consumption is currently due to datacenters, its CO_2 footprint surpassed that of global air traffic in 2013 [1], and their power consumption rate is rapidly growing [2]. Our preliminary estimates suggest that a distributed infrastructure made of low-energy devices, finely distributed in close proximity to the clients they service, would have significantly lower power requirements than a centralized paradigm, which requires active cooling and increases the load on long-range Internet routing infrastructure.

Further, the high entrance cost to the market creates monopolies where only the largest companies are capable of offering scalable cost-efficient services. Owned by the community of its users, e.g. citizens, businesses and organizations, this network service will be free to join, democratic and designed to guarantee the privacy and security of the data it stores.

Since it is now possible to have cheap, energy efficient, fast, reliable, and always online computing nodes in our homes and businesses, our architecture relies on privately owned computing devices (e.g. `Raspberry Pis` with flash drives). Our network design leverages on the collective storage power of these devices: every node will store parts of other nodes' data to guarantee redundancy and reliability, and a carefully designed cryptographic architecture will prevent unwanted access to the stored data.

A.2 State of the art

A wide variety of technologies have been developed along the way from mainframe-terminal paradigm to Internet of Things and decentralized networks. In this paper, we will underline, when needed, what technologies still need to be perfected to achieve our goal of developing an headless, peer-to-peer infrastructure offering a level of reliability, security and performance comparable to that of a centralized paradigm to an arbitrarily large user base. However, most of this work will arguably consist in using technologies that are already in place under stronger reliability assumptions offered by the possibility of having permanently online, low-energy devices in our homes and businesses.

A.2.1 Technologies

Hardware **Internet of Things** technology is arguably at the core of RAIN's architecture, as we plan to develop our entire infrastructure on home-based single-board computers such as `Raspberry Pi`. While no significant difference would exist in terms of security and reliability between a network made of, e.g., personal desktops and one made of single-board computers, the latter make the deployment of RAIN's network feasible, as their cost and energy requirements are small enough to make a RAIN node part of a normal network setup.

Decentralization and distribution Distributed computing significantly predates peer-to-peer technology: Grid Computing (see, e.g., [3]) quickly became an effective replacement for supercomputers as soon as the computing power of many slower nodes exceeded that of a mainframe. When an increasing number of users had access to high-speed Internet connections, Grid Computing projects (like `BOINC` [4]) were developed to tackle not only the computing power of arrays of nodes in large computing facilities, but also the spare resources of personal computers of volunteers.

A wide variety of Internet services is now offered by Cloud infrastructure, that builds on top of Grid an abstraction layer that takes care of autonomously mapping *services* to the hardware that offers them. However, it relies on a similar semi-centralized infrastructure, where multiple servers in one or more facilities collectively offer a service by load-balancing the requests.

In order to process and make accessible the increasing amount of communication produced by a growing population of Internet of Things devices, Edge computing technology (see, e.g., [5] [6]) decentralizes the paradigm further, using finely-distributed cloud nodes that are geographically close to the source of data to perform pre-processing and improve distribution performance.

Redundancy & retrievability The problem of reliably storing and retrieving information on faulty devices is addressed in centralized as well as decentralized contexts by error-correcting codes (such as Solomon-Reed encoding [7]), `RAID` (Redundant Array of Independent Disks) technology (see, e.g., [8]), and in peer-to-peer contexts with Distributed Hash Tables, as in [9] (see, e.g., [10]).

Security The issue of storing data on untrusted nodes while guaranteeing confidentiality and integrity has been addressed in the past by means of asymmetric ciphers and Authenticated Data Structures [11] [12]. Specifically in the context of headless networks, the issue of removing trusted third-party authorities is currently addressed by blockchain technolgy (see, e.g., [13]): we later argue that Authenticated Distributed Hash Tables could be in principle used to substitute that technology in a context where all nodes have a relatively high uptime.

P2P security More specifically in the field of peer-to-peer networks, where no party is trusted a priori, we will be using Proofs of Space [14] [15] as a means to perform hardware commitment and limit Sybil attacks, and distributed, verifiable random number generation to guarantee security properties of the global organization of our network.

A.2.2 Related projects

Cubbit A startup project founded by two of the authors (Moschettini and Posani), implements the redundancy strategy and recovery procedures described in this paper to offer a distributed file storage service using single-board computers and home-grade Internet connections. Metadata is stored in a traditional client-server architecture.

Storj A cooperative storage cloud based on Bitcoin blockchain technology where cryptocurrency is exchanged for storage.

Lima A startup project offering a specialized single-board device to interface personal storage devices with an Internet connection. No redundancy across multiple devices is implemented.

IPFS An open source distributed file system protocol designed to persistently store and make available objects on the Internet. There is no central point of failure, and nodes do not need to trust each other.

Telhoc A company specialized in securing and optimizing peer-to-peer database storage.

A.3 Potential applications

The RAIN architecture is a bio-inspired network with redundancies, distributed control, high energy efficiency, error correction, self-repair, and obvious potential for autonomous adaptation (learning) in later versions. Together with these high level properties, the architecture's privacy by design, open source, community ownership, scalability, and performance makes it a candidate architecture for supporting a number of additional services besides cloud storage. These include secure communication (text, voice, video), content delivery, search engines, finance (cryptocurrency), digital manufacturing, transportation (autonomous vehicles), as well as part of secure e-governance (state administration). In short we can envision a RAIN style architecture with privacy and security by design at the center of the our emerging infrastructure of infrastructures.

A.4 Structure of this work

This work is divided into three parts.

The first part, **Architecture Feasibility**, demonstrates that a distributed infrastructure with a redundancy strategy based on Solomon-Reed encoding easily yields a *data lifetime* of 10^{16} years ($\sim$ age of the Earth), with a *storage ratio* (how much we need to write across the network vs. the size of the data uploaded) of 1.5, using a local *village* of 36 physical nodes where the primary data is stored. Assumptions, proofs and estimates are detailed in this section. This part provides a proof of principle for the proposed RAIN architecture.

The second part, **Architecture Challenges**, defines the scientific and technical challenges for designing data privacy and security, network organization as well as data, metadata and credential handling. As this cryptographic part of the architecture is both technical and lengthy these details will be published elsewhere. In this part we also briefly touch on the environmental impact issues of our distributed infrastructure and that of a traditional centralized client-server paradigm. This part outlines the next steps needed in developing the RAIN architecture.

In the **Potential Impact** part, we provide an overview of the potential impact a large distributed digital infrastructure like RAIN could cause. RAIN could support the development of communitarian services including telecommunication, content delivery, cryptocurrency, and distributed administrative (nation state and regional governmental), which currently are services managed in a centralized manner through trusted third parties.

Part B Architecture Feasibility

Our network will be free to join by anyone with a headless single-board computer (e.g. `Raspberry Pi`), a storage device and a home-grade Internet connection.

Nomenclature

Network: the set of interconnected single-board computers and servers operating our service.

Client: a device (e.g., personal computer, smartphone) that a **user** uses to access our service.

Node: a headless single-board computer, attached to a storage device (e.g., HDD, SSD), owned by a user and persistently connected to his/her home Internet connection.

Drive: an online environment where files can be stored and shared by one or more users.

Our Free and Open Source software will be available to the end user in the form of downloadable clients. Once installed on a new user's personal computer, our **client** will download a platform-specific disk image including a lightweight operating system and our pre-configured software, and use it to initialize the user's single-board **node**. This initialization procedure also serves the purpose to bind the node to its **user**'s account.

Once the node has been initialized, the user is asked to power it and persistently connect it to a home-grade Internet connection. Whenever a storage device (e.g., an hard disk or a flash drive) is connected to the node, *half* of its capacity is allocated to its owner's account.

A user can associate one or more clients to his/her account, and use them to access the service via a Graphical User Interface that allows file tree navigation. Users can create **drives**, namely, storage environments that can be shared among one or more users. Each drive has a distinct file tree and an owner, that can manage other users' privileges or revoke their access.

If a user's node experiences temporary *downtime* or permanent *failure*, its owner's files' availability is unaffected. Any user's files can be accessed from any Internet connection. Whenever a client is not available, access is guaranteed by a Web-based interface.

B.1 Node properties

Our network is based on unreliable nodes subject to *downtime* (temporary unreachability due to connectivity issues) and *failure* (permanent loss of data due to hardware breakage or wearing, or to human interaction). In order for it to reliably store data, a **redundancy strategy** must be implemented.

Our redundancy model depends on three **node properties**:

Lifetime ($[L] = \text{s}$): the amount of time between the moment a node enters the network and its first unrecoverable failure.

Downtime ($d \in [0, 1]$): the fraction of time a node is unreachable due to temporary network- or power-related issues.

Upload and download speed ($[S_u] = [S_d] = \text{B/s}$): the average amount of data that can be transferred from and to a node per time unit.

Disclaimer (A) As *example scenario*, we modeled our expected network behavior for a set of nodes in Italy, urban area of Bologna.

Disclaimer (B) Wherever little or no data is available, we will try to provide the most relaxed authenticated conservative estimate. This will often result in overly conservative estimates, which we will relax only when more data will be available.

B.1.1 Lifetime

Hazard function

Let $R(t)$ be the **survival rate** of a node, i.e., the probability that a node will not experience an unrecoverable failure before t.

The **hazard function** $h(t)$ is defined as the limit fraction of nodes experiencing failure per time unit:

$$h(t) = -\frac{1}{R(t)}\frac{dR}{dt}(t)$$

We can express R as a function of h by

$$R(t) = \exp\left(-\int_0^t h(t')dt'\right)$$

and note how, if $h(t) = \text{const}$, R is an exponential function and failure is a memory-less process. From $R(t)$ we get the probability density of lifetimes $l(t)$ by

$$l(t) = -\frac{dR(t)}{dt}$$

and the expected lifetime L by

$$L = \int_0^\infty t\, l(t) dt$$

And for $h(t) = h = \text{const}$ we have $L = 1/h$.

Exponential approximation

In our case, failure can be caused by technical malfunction or user interaction. We can therefore expand

$$h(t) = h_{\text{tech}}(t) + h_{\text{user}}(t)$$

Technical malfunction The failure rate of devices subject to wearing (storage devices among them) is commonly referred to as **bathtub curve**:

- Manufacturing imperfections generally result in early failures (*infant mortality*), thus producing a higher failure rate at the early stages of the device usage.
- Failure rate is then lower for a certain period of time where in general only failures of random nature occur.
- Failure rate increases again in the late stages of the device usage due to wearing.

Storage devices connected to our network, however, will not necessarily be new. The age distribution $a(t)$ of random, alive, not newly bought devices is

$$a(t) = \frac{R(t)}{\int_0^\infty R(t)}$$

therefore, let $b(t)$ be a bathtub hazard function and p_{new} the probability that a storage device is newly purchased when the node is initialized, then h_{tech} becomes

$$h_{\text{tech}}(t) = p_{\text{new}}\, b(t) + (1 - p_{\text{new}}) \left(\int_0^\infty a(t')b(t+t')\, dt'\right) \tag{1}$$

Figure 1 shows the effect of the above convolution on an experimental [16] bathtub function. As p_{new} decreases, the hazard function becomes smoother and closer to its asymptotic value.

Under the assumption $p_{\text{new}} \ll 1$, h_{tech} can be safely approximated to a constant equal to the asymptotic value of b.

User interaction The rate of failures caused by user interaction will highly depend on the usability, performance and reputation of our network, and is more difficult to estimate now.

User fidelity is subject to infant mortality (as every other Internet service, our network will

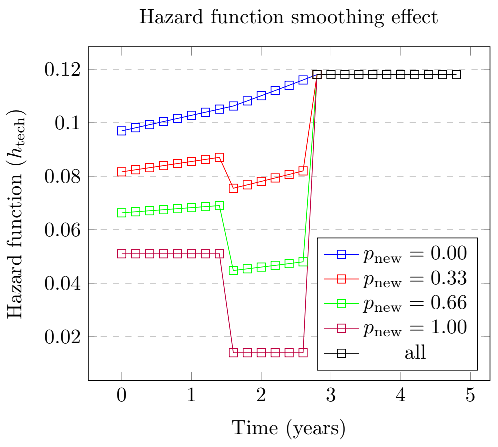

Figure 1: Hazard function h_{tech} as per Equation 1 for different values of p_{new}. For $p_{\text{new}} = 1$, h_{tech} reduces to the bathtub function b. As p_{new} decreases, h_{tech} becomes smoother and higher.

have a **bounce rate** of users that try our service but don't find it appealing to their needs) but not to wearing. h_{user} will therefore be higher for young nodes, and reach a *plateau* after a short trial period.

When discussing our upcoming security challenges, we will see how we plan to store no critical data on newly connected nodes. This will contribute to reduce the bounce rare effect and crop away the higher, non-uniform part of h_{user}, which we will therefore assume constant.

Experimental data

Hard Disk Drives (HDD) A four-year study on HDD wearing [16] (see Figure 1 for $p_{\text{new}} = 1$) provides experimental data in agreement with a bathtub curve model:

Regime	Failure rate
Infant mortality	5.1%
Random failures	1.4%
Wearing	11.8%

Under the limit approximation $h_{\text{HDD}}(t) = h_{\text{HDD}} = 0.126/\text{year}$ we get an expected lifetime

$$L_{\text{HDD}} \simeq 8 \text{ years}$$

Solid State Drives (SSD) Due to their extended lifetime, and to the fact that their high cost delays acquisition in the enterprise market, less experimental data is available concerning SSD lifetime.

Due to the absence of mechanical parts, SSD wearing is not determined by time. Each flash memory block can undergo a limited number of rewrites before it becomes unusable. Load balancing algorithms are implemented to evenly distribute the load across the sectors even if the same file is rewritten multiple times. To provide an estimate of SSD lifetimes, we then need to have an estimate of how much data will be written on the drive per time unit.

For a node in our network, SSD write speed is bound by its download bandwidth, which we can preliminarily estimate (see later) to 6.5Mbit/s = 70GB/day. It is important to remark that this is **a large overestimate**, as it assumes that every user will use 100% of his nominal connection bandwidth only to flood our service with write requests.

An 18 month experimental study [17] has been carried out to determine how much data could be written on 6 different Solid State Drives. Data was continuously written on them at maximum throughput for the whole duration of the experiment. The first SSD to experience a failure broke down after 728 TB of data were written on it (the last two failed around the 2.5 PB mark).

Together with the write overestimate above, this yields an SSD lifetime expectancy of

$$L_{SSD} \gg 28.5 \text{ years}$$

Node lifetime No data is available on the failure rate of common single-board computers like `Raspberry Pi`, most informal failure reports referring to physical damage and not to wearing. Due to this and to the fact that single-board node failures are recoverable, we will therefore assume the failure rate of nodes due to single-board computer failure to be negligible.

As we said, we cannot provide estimates for the user contribution to failure rate. In line with our policy to provide conservative estimates, we will then set the average expected node lifetime to

$$L = 2 \text{ years}$$

B.1.2 Speed and downtime

Download speed . Accurate, high quality data is available [18] from global Content Delivery Networks that can use their service as a means to study the download speed distribution of a broad sample of Internet users.

Upload speed . Data on upload speeds is more difficult to obtain, as common users use their connection mostly to download data (hence the bandwidth asymmetry in home-grade Internet connections). Data is explicitly gathered [19] by organizations that offer connection speed tests online. Their sampling, however, are biased, as connectivity issues often lead users to make speed tests, and only the more experienced users are aware of speed test services.

Downtime . Downtime data for home-grade Internet connections is not publicly available and difficult to gather. Uptime (real time = uptime + downtime) monitoring would require a node to be permanently active in users' homes to monitor their connection for extended periods of time.

Experimental procedure

In order to get a reliable estimate for speed and bandwidth, we develop a script to monitor home-grade Internet connections for extended periods of time with a `Raspberry Pi 3`.

- The script only runs when the Raspberry is connected to a router. Every log entry is paired with the router's physical address (`MAC` address) to permit analysis.
- Every two seconds, the script tries to `ping` an external node (Google Public DNS, address `8.8.8.8`). Both success and failure are logged.
- Every ten minutes, the external `IP` address of the node is logged. We will use `IP` consistency later as a means to detect Botnet Attacks.
- Every thirty minutes, a speed test (using `Ookla's Speedtest`) is run, determining and logging roundtrip time, download and upload speed with the aid of an external reliable server.
- Every time the script detects a new router (i.e., new `MAC` address), the connection's provider and location (latitude and longitude) are determined and logged using `ip-api.com` geolocation API and the router's vendor is determined and logged using `macvendors.com` `MAC` Vendor Lookup API.

Results We proceeded to monitor 7 home-grade Internet connections, each for up to 48 hours. Raw data (`MAC` addresses were obfuscated for privacy reasons) is available[1].

Downtime data

[1] `https://github.com/rainvg/supplementary-material/tree/master/The%20Alternative%20Information%20Plan%20(2017)/netmonitor`

#	Ping OK	Ping FAIL	Downtime
α	78	91088	$(8.5 \pm 1) \cdot 10^{-4}$
β	0	2018	$(0\pm?)$
γ	15	93013	$(1.6 \pm 0.4) \cdot 10^{-4}$
δ	68	3690	$(1.8 \pm 0.2) \cdot 10^{-2}$
ϵ	14	116101	$(1.2 \pm 0.3) \cdot 10^{-4}$
ζ	4	2288	$(1.74 \pm 0.04) \cdot 10^{-3}$
η	62	117561	$(5.2 \pm 0.7) \cdot 10^{-4}$

Speed data

#	Tests	Upload	Download
α	108	(0.855 ± 0.005)Mbit/s	(12.3 ± 0.2)Mbit/s
β	3	(72.7 ± 0.5)Mbit/s	(83 ± 5)Mbit/s
γ	103	(9.04 ± 0.04)Mbit/s	(9.29 ± 0.03)Mbit/s
δ	5	(25.1 ± 0.2)Mbit/s	(63 ± 3)Mbit/s
ϵ	130	(19.2 ± 0.2)Mbit/s	(47.2 ± 0.3)Mbit/s
ζ	4	(0.82 ± 0.09)Mbit/s	(9 ± 2)Mbit/s
η	131	(0.843 ± 0.004)Mbit/s	(12.9 ± 0.2)Mbit/s

Our experiment provides the following estimates:

Average downtime	0.003
Average upload speed	17Mbit/s
Average download speed	34Mbit/s

In line with our policy to keep all our estimates conservative and compatible with a worst-case scenario, for the rest of this work we will assume:

$$\begin{aligned} d &= 0.005 \\ S_u &= 0.2 \text{ MB/s} \\ S_d &= 0.5 \text{ MB/s} \end{aligned}$$

B.2 Solomon-Reed erasure codes

An erasure code is a forward error correction code which transforms a block of N symbols into a message with K symbols ($K > N$) such that the original block can be recovered from a subset of the message. We use Solomon-Reed encoding [7], an optimal (i.e., the original block can be recovered from any N symbols of the message) erasure code that uses polynomial oversampling and redundancy over finite fields.

B.2.1 Polynomial interpolation

Matrix preliminaries

Let $\mathbb{K}$ be a field. We define an $N-1$ degree polynomial on $\mathbb{K}$ by

$$p_{a_0,\ldots,a_{N-1}}(x) = \sum_{i=0}^{N-1} a_i x^i$$

with $a_0, \ldots, a_{N-1}, x \in \mathbb{K}$.

Let $\mathbf{x} = (x_0, \ldots, x_{N-1})$ be a vector of N distinct values in $\mathbb{K}$, which we call **sampling points**. We can define $\mathbf{x}$'s **Vandermonde matrix** by uplifting each sampling point successively to the power $0, \ldots, N-1$ in each column

$$V^{(\mathbf{x})} = \begin{pmatrix} x_0^0 & \cdots & x_0^{N-1} \\ \vdots & \cdots & \vdots \\ x_{N-1}^0 & \cdots & x_{N-1}^{N-1} \end{pmatrix}$$

It is possible to prove that the determinant of a Vandermonde matrix is

$$\det(V^{\mathbf{x}}) = \prod_{0 \le i < j \le N-1} (x_i - x_j)$$

which is non-null whenever $i \neq j \iff x_i \neq x_j$.

Now let $\delta_{i,j}$ denote the Kronecker delta ($\delta_{i,i} = 1$, $\delta_{i,j\neq i} = 0$). For any $j \in [0, N-1]$ we can solve

$$\begin{pmatrix} x_0^0 & \cdots & x_0^{N-1} \\ \vdots & \cdots & \vdots \\ x_{N-1}^0 & \cdots & x_{N-1}^{N-1} \end{pmatrix} \begin{pmatrix} a_0^{(\mathbf{x},j)} \\ \vdots \\ a_{N-1}^{(\mathbf{x},j)} \end{pmatrix} = \begin{pmatrix} \delta_{0,j} \\ \vdots \\ \delta_{N-1,j} \end{pmatrix} \tag{2}$$

for $a_0^{(\mathbf{x},j)}, \ldots, a_{N-1}^{(\mathbf{x},j)}$. Note that the right side of Equation 2 constrains the value of the polynomial of coefficients $a_0^{(\mathbf{x},j)}, \ldots, a_{N-1}^{(\mathbf{x},j)}$ to be 1 in x_j and 0 in $x_{i\neq j}$. As j can take N distinct values, Equation 2 defines N linear systems of N equations.

We now define the **Lagrange basis** $\{l^{(\mathbf{x},j)}\}_{j\in[0,N-1]}$ for $\mathbf{x}$ by

$$l^{(\mathbf{x},j)} = p_{a_0^{(\mathbf{x},j)},\ldots,a_{N-1}^{(\mathbf{x},j)}}$$

and by Equation 2 we have for each $j, i \in [0, N-1]$

$$l^{(\mathbf{x},j)}(x_i) = \delta_{i,j}.$$

Let $\mathbf{y} = (y_0, \ldots, y_{N-1}) \in \mathbb{K}^N$ define a particular set of desired polynomial values corresponding to each sampling value $\mathbf{x}$. Since polynomials form a vector space we have that

$$p^{(\mathbf{x},\mathbf{y})} = \sum_{j=0}^{N-1} y_j \, l^{(\mathbf{x},j)}$$

satisfies

$$p^{(\mathbf{x},\mathbf{y})}(x_i) = y_i$$

as the value of each $l^{(\mathbf{x},j)}$ is 1 only in x_j, and 0 in all the other sampling points.

In other words, for any set of N distinct points $\mathbf{x}$ and N values $\mathbf{y}$, it is possible to determine the coefficients of a polynomial whose value on each x_i is y_i.

Computational complexity

As we have seen, polynomial interpolation is a two-step procedure. For a given $\mathbf{x}$, the coefficients $\{a_i^{(\mathbf{x},j)}\}_{i,j}$ of the Lagrange basis on $\mathbf{x}$ can be determined as per Equation 2.

Indeed, Equation 2 can be rewritten as

$$\begin{pmatrix} a_0^{(\mathbf{x},j)} \\ \vdots \\ a_{N-1}^{(\mathbf{x},j)} \end{pmatrix} = \begin{pmatrix} x_0^0 & \cdots & x_0^{N-1} \\ \vdots & \cdots & \vdots \\ x_{N-1}^0 & \cdots & x_{N-1}^{N-1} \end{pmatrix}^{-1} \begin{pmatrix} \delta_{0,j} \\ \vdots \\ \delta_{N-1,j} \end{pmatrix}$$

where the Kronecker deltas simply select columns of the inverse matrix. Therefore the Lagrange basis for one $\mathbf{x}$ can be computed with one matrix inversion.

Once the Lagrange basis matrix has been computed, the coefficients $a_0^{(\mathbf{x},\mathbf{y})}, \ldots, a_{N-1}^{(\mathbf{x},\mathbf{y})}$ of $p^{(\mathbf{x},\mathbf{y})}$ are then given by

$$\begin{pmatrix} a_0^{(\mathbf{x},\mathbf{y})} \\ \vdots \\ a_{N-1}^{(\mathbf{x},\mathbf{y})} \end{pmatrix} = \begin{pmatrix} x_0^0 & \cdots & x_0^{N-1} \\ \vdots & \cdots & \vdots \\ x_{N-1}^0 & \cdots & x_{N-1}^{N-1} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ \vdots \\ y_{N-1} \end{pmatrix} \tag{3}$$

which requires only one matrix-vector multiplication.

It is a known result that matrix inversion can be computed in sub-cubic time. Strassen's algorithm [20], for example, allows $N \times N$ matrix inversion in $O(N^{2.807})$ time. Other algorithms are known with faster asymptotic complexity (see Coppersmith-Winograd algorithms) but their large constant factor makes them usually efficient only for very large matrices.

Matrix-vector multiplication can be optimized on finite semirings [21]. By allowing $O(N^{2+\epsilon})$ preprocessing on the matrix only, it is possible to compute a matrix-vector multiplication in $O\left(\frac{N^2}{(\epsilon \log N)^2}\right)$ time.

The two above results make the polynomial interpolation process efficient, especially for relatively small matrices, whenever we need to interpolate several polynomials on the same x_is but for distinct y_is, which, as we will see, is the case at hand.

B.2.2 Galois Fields

Finite fields (Galois fields) exist that contain a finite number of elements. A field with q elements exists if and only q can be expressed in the form

$$q = p^k,$$

where p is a prime number and k is a positive integer. Therefore, a field $GF(2^8)$ exists with 256 elements, and each element of $GF(2^8)$ can be represented by exactly one byte.

Computational remark . Summation on $GF(2^8)$ reduces to bitwise XOR-ing. Multiplication and inversion are more complex. However, unlike $GF(2^{32})$ and $GF(2^{64})$, which could take advantage of the larger registries of 32 and 64 bit CPUs, $GF(2^8)$ is small enough that multiplications and divisions can be pre-computed and stored in two exhaustive 64 KB lookup tables, small enough to fit the `L2` cache of a typical `ARM` CPU.

B.2.3 Polynomial oversampling for redundancy

Let $s = s_0, \ldots, s_{S-1}$ be a string of S bytes, that we want to reliably store. Let N, K be integers so that $N < K \ll S$. We can organize s in $L = \lceil S/N \rceil$ padded **blocks $\mathbf{a^i}$** of N bytes:

$$a_j^{(i)} = \begin{cases} s_{Ni+j} & \text{iff } Ni + j < S \\ 0 & \text{otherwise} \end{cases}$$

such that each $\mathbf{a^{(i)}}$ can be interpreted as the $GF(2^8)$ coefficients of an $N-1$ degree polynomial. Note that since $GF(2^8)$ is finite, we must have $N \leq 256$ to prevent degenerate polynomials over different coefficients. Let

$$p^{(i)} = p_{a_0^{(i)}, \ldots, a_{N-1}^{(i)}}$$

We can now define L K-bytes long **data blocks $\mathbf{b^{(i)}}$** by

$$b_j^{(i)} = p^{(i)}(j)$$

(note how again since $GF(2^8)$ is finite we must also have $K \leq 256$).

Now, let $\mathbf{x} \in \mathbb{K}^N$ be a vector with distinct components $x_0 \neq \ldots \neq x_{N-1} \in [0, K-1]$ and $\mathbf{y^i}$ defined by

$$y_j^{(i)} = b_{x_j}^{(i)}$$

we know from Section B.2.1 that we can use $\mathbf{x}$ and $\mathbf{y}$ as inputs for polynomial interpolation. In particular, we have

$$a_j^{(\mathbf{x},\mathbf{y}^{(\mathbf{i})})} = a_j^{(i)}$$

Due to oversampling, any N-subset of components of $\mathbf{b^{(i)}}$ is sufficient to recover the original $\mathbf{a^{(i)}}$.

Summing up To implement Solomon-Reed redundancy we organize s in N bytes long sequences $\mathbf{a^{(i)}}$. We interpret each $\mathbf{a^{(i)}}$ as coefficients for a $N-1$ degree polynomial that we oversample in $0, \ldots, K-1$, with $K > N$. Oversampling produces a $\mathbf{b^{(i)}} \in \mathbb{K}^K$ from each $\mathbf{a^{(i)}} \in \mathbb{K}^K$. As we just showed, from any N

components of each $\mathbf{b^{(i)}}$ we can recover the corresponding $\mathbf{a^{(i)}}$ by means of polynomial interpolation.

To implement redundancy, we can now store each component of $\mathbf{b^{(i)}}$ separately: $b_0^{(i)}$ will be stored on a node, $b_1^{(i)}$ will be stored on independent node, and so on. As a result, K nodes will be storing each one component of $\mathbf{b^{(i)}}$, and as long as any N of them are online and reachable, the original value of $\mathbf{a^{(i)}}$ can be retrieved.

Samples generation Let $i \neq k$. Since $\mathbf{b^{(i)}}$ and $\mathbf{b^{(k)}}$ are the result of polynomial oversampling on two independent blocks $\mathbf{a^{(i)}}$ and $\mathbf{a^{(k)}}$, knowledge on the components of $\mathbf{b^{(i)}}$ contribute to the recovery of $\mathbf{a^{(k)}}$. Therefore, while each component of $\mathbf{b^{(i)}}$ needs to be stored on an independent node, we can safely store a component of $\mathbf{b^{(i)}}$ on the same node that also stores a component of $\mathbf{b^{(k)}}$, as this does not affect the overall probability to recover $\mathbf{a^{(i)}}$ or $\mathbf{a^{(k)}}$.

Indeed, the most efficient way to implement redundancy is to let each node store one of the **data samples** $\mathbf{s^{(0)}}, \ldots, \mathbf{s^{(K-1)}}$ defined by

$$s_j^{(i)} = b_i^{(j)}$$

Figure 2 shows the complete redundancy process, starting from s, to produce $\mathbf{s^{(0)}}, \ldots, \mathbf{s^{(K-1)}}$.

Data retrieval For any set of distinct $i_0, \ldots, i_{N-1}$, if we retrieve from the network $s^{(i_0)}, \ldots, s^{(i_{N-1})}$ then, for each $k \in [0, L-1]$, interpolation on $\{s_k^{(i_j)}\}_{j \in [0,N-1]}$ yields $\mathbf{a^{(k)}}$, whose concatenation produces the original s. Figure 3 displays the whole retrieval process, starting from $\mathbf{s^{(i_0)}}, \ldots, \mathbf{s^{(i_{N-1})}}$ to produce s.

Note how the above procedure requires L polynomial interpolations for different values of $s_k^{(i_j)}$, but the sampling points remain i_j for each polynomial. As showed in Section B.2.1, this makes the procedure particularly efficient for large values of L, as it allows to invert only one matrix and to amortize reconstruction time by means of preprocessing on the Lagrange basis matrix.

B.3 Redundancy strategy

B.3.1 Villages

Decay and recovery

In Section B.2, we have seen how Solomon-Reed erasure codes allow us to generate, from an S-bytes string of data, K data samples of size $\lceil S/N \rceil$. As long as any N data samples are available, the original string can be retrieved. This property alone, however, does not significantly increase the lifetime of the string.

Let X denote a random variable representing the lifetime of a node. Modeling failure as an exponential process (see Section B.1.1) the probability of a node being alive at time t is

$$P(X > t) = exp(-\frac{t}{L})$$

and the probability of H independent nodes being alive at time t

$$\begin{aligned} P(X_0 > t \wedge \ldots \wedge X_{H-1} > t) &= \prod_i^H P(X_i > t) \\ &= exp(-\frac{Ht}{L}) \end{aligned}$$

follows an exponential distribution with average L/H, i.e., it takes on average L/H for one of H independent nodes to experience a failure. Since exponential processes are memoryless the average time $\tilde{t}_d$ for a string to become unrecoverable is

$$\tilde{t}_d = L\left(\sum_{i=N}^{K} \frac{1}{i}\right) = \mathcal{O}\left(L(\log(K) - \log(N))\right)$$

Whenever at least N data samples are available, polynomial interpolation and re-sampling allow us to generate missing data samples and store them on new nodes. In order to extend *data lifetime* beyond the order of magnitude of node lifetime, we will implement real-time data monitoring and *recovery procedures*.

Optimal data distribution

A **file** is the minimum unit of data whose integrity we are safeguarding: failing to retrieve any block of a file will be equivalent to failing to recover the whole file. Under this premise, the probability of failed retrieval is minimized by storing corresponding samples of blocks in the same file on the same set of nodes.

Storing distinct files on distinct sets of nodes reduces correlations among failed retrievals without affecting their expected number. On the other hand, if multiple files are stored on the same set of nodes, upon failure of a node more files will simultaneously need to be recovered. This will make the recovery procedure longer and increase the probability of the redundancy level falling under N before the recovery procedure is completed. An optimal data distribution strategy would store each file on a distinct set of nodes.

$$\begin{pmatrix} s_0 & s_1 & \cdots & s_{N-1} \\ s_N & s_{N+1} & \cdots & s_{2N-1} \\ \vdots & \cdots & \cdots & \vdots \\ s_{S-1} & 0 & \cdots & 0 \end{pmatrix} = \begin{pmatrix} a_0^{(0)} & a_1^{(0)} & \cdots & a_{N-1}^{(0)} \\ a_0^{(1)} & a_1^{(1)} & \cdots & a_{N-1}^{(1)} \\ \vdots & \cdots & \cdots & \vdots \\ a_0^{(L)} & a_1^{(L)} & \cdots & a_{N-1}^{(L)} \end{pmatrix} \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} \begin{array}{c} \begin{matrix} \mathbf{s^{(0)}} & \mathbf{s^{(1)}} & \cdots & \mathbf{s^{(K-1)}} \\ \| & \| & & \| \end{matrix} \\ \begin{pmatrix} b_0^{(0)} & b_1^{(0)} & \cdots & b_{K-1}^{(0)} \\ b_0^{(1)} & b_1^{(1)} & \cdots & b_{K-1}^{(1)} \\ \vdots & \cdots & \cdots & \vdots \\ b_0^{(L)} & b_1^{(L)} & \cdots & b_{K-1}^{(L)} \end{pmatrix} \end{array}$$

Figure 2: Solomon-Reed redundancy process. A string s is organized in blocks $\mathbf{a^{(0)}}, \dots, \mathbf{a^{(L-1)}}$ of N bytes each. Each block is interpreted as polynomial coefficients and the polynomial is oversampled to produce $\mathbf{b^{(0)}}, \dots, \mathbf{b^{(L-1)}}$ (each arrow represents a polynomial oversampling). All the corresponding components of each $\mathbf{b^{(i)}}$ are then reorganized into data samples $s^{(0)}, \dots, s^{(K-1)}$, each of which is stored on an independent node.

$$\begin{array}{c} \begin{matrix} \mathbf{s^{(i_0)}} & \mathbf{s^{(i_1)}} & \cdots & \mathbf{s^{(i_{N-1})}} \\ \| & \| & & \| \end{matrix} \\ \begin{pmatrix} b_{i_0}^{(0)} & b_{i_1}^{(0)} & \cdots & b_{i_{N-1}}^{(0)} \\ b_{i_0}^{(1)} & b_{i_1}^{(1)} & \cdots & b_{i_{N-1}}^{(1)} \\ \vdots & \cdots & \cdots & \vdots \\ b_{i_0}^{(L)} & b_{i_1}^{(L)} & \cdots & b_{i_{N-1}}^{(L)} \end{pmatrix} \end{array} \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} \begin{pmatrix} a_0^{(0)} & a_1^{(0)} & \cdots & a_{N-1}^{(0)} \\ a_0^{(1)} & a_1^{(1)} & \cdots & a_{N-1}^{(1)} \\ \vdots & \cdots & \cdots & \vdots \\ a_0^{(L)} & a_1^{(L)} & \cdots & a_{N-1}^{(L)} \end{pmatrix} = \begin{pmatrix} s_0 & s_1 & \cdots & s_{N-1} \\ s_N & s_{N+1} & \cdots & s_{2N-1} \\ \vdots & \cdots & \cdots & \vdots \\ s_{S-1} & 0 & \cdots & 0 \end{pmatrix}$$

Figure 3: Solomon-Reed retrieval process. N data samples $\mathbf{s^{(i_0)}}, \dots, \mathbf{s^{(i_{N-1})}}$ are recovered from N independent nodes. They are reorganized in L N-component subsets of the original $\mathbf{b^{(0)}}, \dots, \mathbf{b^{(L)}}$. Polynomial interpolation (as per Equation 3; each arrow represents a polynomial interpolation on the same Lagrange basis) is then used to reconstruct $\mathbf{a^{(0)}}, \dots, \mathbf{a^{(L-1)}}$, which can be reorganized into the original string s.

Polling overhead

Failure of a node can be caused not only by failure of its storage device, but also by damage to its single-board computer or by user-generated permanent disconnection. The latter causes cannot be detected by the node experiencing the failure and notified to the nodes that share the redundancy of files with it. Real-time, distributed monitoring of uptime and data availability must therefore be implemented via **polling** (namely, iterated active checking for the availability of a resource).

File size statistics In order to estimate the polling overhead caused by storing each file on a distinct set of nodes, we need to determine the average number of distinct files each node will be storing.

In order to do so, we implemented an automated online survey to anonymously scan the size distribution of personal files of 34 volunteers. Volunteers were prompted with an online form that allowed them to select one or more folders from their computer, and were asked to select any folder where personal files were stored

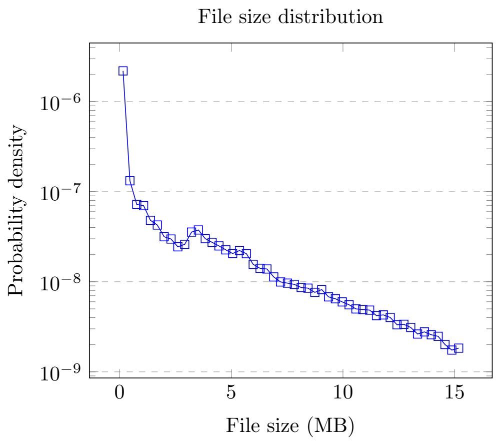

Figure 4: Experimental file size distribution generated by our scanner on the personal files of 34 volunteers. Bottom 98 percentiles are represented here (the top 2 were cropped to allow semilogarithmic representation.)

(e.g. `Desktop`, `Documents`, `Music`, `Pictures`, and so on). $1.01 \cdot 10^6$ files were scanned.

Figure 4 shows the resulting file size distribution. After a steeply decreasing regime, the distribution shows a dent around 3.8 MB followed by an exponential distribution for larger files.

The experiment shows an average file size

$$\langle S \rangle = 5.56 \text{ MB}$$

Polling bandwidth If we preliminarily underestimate $N = 8$, $K = 12$, the average data sample's size will be 695 KB, and the time needed to transfer it will be approximately 7 s. In order for recovery procedure time to benefit from the reduced concurrency induced by independently storing files, polling frequency must be in the same order of magnitude, e.g., 10 s. A full node with $C = 1$ TB storage capacity will contribute to store on average $1.58 \cdot 10^6$ distinct files, which would result in $1.89 \cdot 10^7$ polling connections. The size of a single, empty `UDP` packet is 52 B. The polling bandwidth required to monitor files distributed independently across the nodes would therefore be at least 98.3MB/s, far beyond the upload speed S_u (see Section B.1.2).

Daemons and villages

In order to reduce polling the overhead stemming from real-time, distributed data monitoring, we will divide the storage capacity of each node in partitions of Z bytes. Each partition will be managed by a **daemon**, namely, an independent instance of a data-managing software. Daemons will be organized in groups of K that we call **villages**. Each file in our network will be stored by a village, and each daemon will store the same components of each data block of each file.

Remark: Z must be the same for every daemon in the same village. Since every time a file is uploaded to a village the same space is occupied on every daemon (each daemon is storing a data block of the same size), any space allocated beyond the storage capacity of the smallest daemon would be wasted.

B.3.2 Redundancy parameters

Lazy recovery

Polynomial interpolation and re-sampling allows us to recover data samples that went missing. In order to recover any number of missing data samples, however, at least N other data samples need to be gathered on the same node (polynomial interpolation requires at least N samples to occur).

Due to the fixed networking cost of interpolation, it would therefore be more efficient to allow multiple data samples of the same file to be lost before performing a recovery procedure, thus amortizing the fixed networking cost of interpolation by re-sampling more missing data samples at a time.

We will model a village with the following *rules*:

- The village is always composed of K daemons. Whenever a daemon experiences a failure, it is immediately replaced by a new, empty one.

- Whenever the **redundancy level** of a file (namely, how many distinct data samples of the file are stored by the village) reaches a **recovery threshold** T ($h = T/N$ will be called **recovery ratio**), a recovery procedure is performed.

  Let us call **sources** the daemons that, at the beginning of the recovery procedure, have a data sample of the file, and **sinks** the daemons that don't. The procedure is implemented as follows:

  – One of the sinks downloads N chunk samples of the file from N sources.

  – The sink recovers the original file and re-samples all the missing chunk samples.

  – The sink stores one of the new data samples, and sends one of the other $T - N - 1$ to each other sink.

Parameters

To sum up, our redundancy strategy depends on four **redundancy parameters**, for which adequate values need to be determined:

- **Block size** (N): the number of coefficients in a polynomial.

- **Village size** (K): the number of daemons in a village, equal to the number of data samples per file. $r = K/N$ will be called **storage ratio**.

- **Recovery threshold** (T): the redundancy level that triggers a recovery procedure for a file. $h = T/N$ will be called **recovery ratio**.

- **Storage size** (Z): the amount of storage space managed by a daemon.

B.4 Redundancy performance

The efficiency and reliability of our redundancy strategy will be determined by the following values:

Data lifetime ($[L^*] = \mathrm{s}$): the average lifetime of a file stored by our network.

Recovery bandwidth ($[B^*] = \mathrm{B/s}$): the average amount of bandwidth used by a daemon to perform the recovery procedures needed to maintain file redundancy.

Data downtime ($d^* \in [0,1]$): the average fraction of time a file is unretrievable due to temporary network- or power-related issues.

B.4.1 Data lifetime

Recovery time

Assuming 100% storage occupancy [2], whenever a node experiences a failure its village will be storing NZ worth of files. The files that will need recovery after a node n experiences a failure will be those stored by n and exactly other T daemons.

As we have seen earlier, the time a file spends in a redundancy level l is on average L/l. Therefore the probability of any file being hosted by $T+1$ daemons at any time is

$$\begin{aligned} p_c &= \frac{L(T+1)^{-1}}{L\sum_{i=T+1}^{K} i^{-1} + t_r} \leq \\ \frac{(T+1)^{-1}}{\sum_{i=T+1}^{K} i^{-1}} &\leq \frac{(T+1)^{-1}}{log(K+1) - log(T+1)} \end{aligned}$$

where the divisor of the right hand side of the second line stems from

$$\begin{aligned} L\sum_{i=T+1}^{K}\frac{1}{i} &\geq L\int_{T+1}^{K+1}\frac{1}{x}dx \\ &= L(\log(K+1) - \log(T+1)) \end{aligned}$$

The probability for a file whose redundancy level is $T+1$ to be hosted by any daemon is

$$p_h = \frac{T+1}{K}$$

therefore the average data that will need to be recovered when a node experiences a failure is

$$\begin{aligned} C &= NZp_cp_h \\ &\leq \frac{NZ}{K(log(K+1) - log(T+1))} \end{aligned}$$

[2] This is likely to be a large overestimate, as it would require each and every user in the network to occupy all of his/her storage space.

The average data transfer load during a recovery procedure is as follows:

	Upload	Download
Source	$\frac{C}{T}$	0
Sink	$\left(\frac{K-T-1}{N}\right)\frac{C}{K-T}$	$\left(1+\frac{K-T-1}{N}\right)\frac{C}{K-T}$

Most of the data transfer is carried out by the sinks. As a home-grade Internet connection has an asymmetric upload/download bandwidth, the time required to complete a recovery procedure is therefore (see Section B.1.2):

$$t_r = \max\begin{cases} \frac{C}{T}\,S_u^{-1} \\ \left(\frac{K-T-1}{N}\right)\frac{C}{K-T}\,S_u^{-1} \\ \left(1+\frac{K-T-1}{N}\right)\frac{C}{K-T}\,S_d^{-1} \end{cases}$$

Recovery failure probability

A file is permanently lost if its redundancy level goes under N before its recovery procedure is completed. The probability of any alive daemon surviving at least t_r is $p_s = \exp(-t_r/L)$. The probability of losing exactly k daemons out of T in t_r time is

$$p_l^{(k)} = \frac{T!}{k!(T-k)!}(1-p_s)^k p_s^{(T-k)}$$

and by the Chernoff bound the probability p_l of losing $T-N+1$ or more chunks is

$$p_l = \sum_{k=T-N+1}^{T} p_l^{(k)} \leq \exp(-T\tilde{D}(h, 1-p_s))$$

with

$$\tilde{D}(r,d) = D\left(\frac{r-1}{r}\,\middle|\middle|\, d\right)$$

and $D(a,p)$ is the relative entropy between a Bernoulli(a) and a Bernoulli(p) distribution:

$$D(a||p) = a\log\left(\frac{a}{p}\right) + (1-a)\log\left(\frac{1-a}{1-p}\right)$$

Lifetime

As we have seen in Section B.3.1, the average time for the redundancy level of a file to decay from K to T is

$$\begin{aligned} t_d &= L\left(\sum_{i=T+1}^{K}\frac{1}{i}\right) \\ \geq L\int_{T+1}^{K}\frac{1}{x}dx &= L(\log(K) - \log(T+1)) \end{aligned}$$

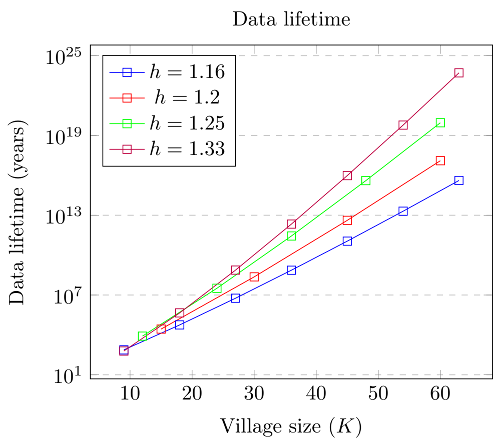

Figure 5: Data lifetime for different values of the recovery ratio h, as a function of the village size K. Here we have $r = 1.5$ and $Z = 100$ GB.

The time for the redundancy level of a file to decay from K to T and for the file to be recovered is $t_d + t_r$. The average number of these **redundancy cycles** is $1/p_l$. Therefore the expected data lifetime is

$$L^* = \frac{t_d + t_r}{p_l}$$

Figure 5 shows data lifetime for different values of the recovery ratio, as a function of the village size. Data lifetime can be made sufficiently large for any practical purpose by villages of manageable size, even when Z is as large as 100 GB.

Remarks

Reliability calculations depend critically on the assumption of independence. In a centralized storage paradigm, disk failures could be correlated for any number of reasons: disks coming from the same manufacturing lot, disks operating in the same physical environment, and so forth. Reliability estimates, therefore, tend to be inflated when the assumption of independence is unfounded [22] [23].

In a distributed paradigm, however, storage devices will be purchased from independent lots, and each will in general operate in an independent physical environment with independent conditions. Therefore, reliability estimates can be considered more trustworthy in a distributed context.

B.4.2 Recovery bandwidth

Recovery bandwidth can be easily computed from the results of Section B.4.1. Upon node failure, on average C worth of data will need recovery.

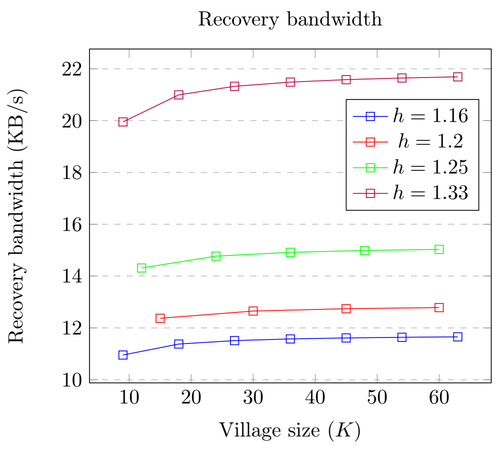

Figure 6: Recovery bandwidth for different values of the recovery ratio h, as a function of the village size K. Here we have $r = 1.5$ and $Z = 100$ GB

Upon recovery, T sources will transfer data to $K - T$ sinks, and from the table in Section B.4.1 we get the total amount of data Q that is transferred every time a recovery procedure is performed

$$Q = 2C\left(1 + \frac{K - T - 1}{N}\right)$$

and the recovery bandwidth

$$B^* = \frac{Q}{K}\left(\frac{L}{K}\right)^{-1} = \frac{Q}{L}$$

Figure 6 shows the recovery bandwidth for different values of the recovery ratio h, as a function of the village size Z. As expected, B^* is an increasing function of h. Recovery bandwidth seems to reach a plateau for large values of K, on which it does not strongly depend. For large villages and relaxed recovery ratios, B^* is in the order of 10 to 20 KB/s when Z is as large as 100 GB.

Remarks

Unlike data lifetime, whose value can be easily made high enough for any practical purpose, recovery bandwidth is small, but non-negligible. Our estimates, however, can be reduced by the following arguments:

Full usage: we assumed that *every* user of the village will use 100% *of its allocated storage space*, which is unlikely to be true in real-world scenarios.

Immortal files: we assumed that files are *never deleted.* Files whose lifetime is shorter than a redundancy cycle (which is in the order of several months) will be deleted before undergoing any recovery procedure.

No access: we assumed that files are *never accessed*. Whenever a file is downloaded by a user, its data samples are collected and interpolated by a client. Since the most costly part of a recovery procedure is, in fact, already carried out, missing samples can be re-sampled and uploaded, and the redundancy for the file restored.

Recovery procedures can also be designed to be carried out when network load is low. For example, when the redundancy level for a file reaches $T + 1$, an **early recovery procedure** could be scheduled for the next nighttime. In the likely event that no additional failure will occur in the next few hours, the recovery procedure will not significantly affect user experience, and make use of the routing infrastructure when it tends to be more idle.

B.4.3 Downtime

As we have seen, the probability of a file's redundancy level being k is

$$p_c^{(k)} = \frac{Lk^{-1}}{L\sum_{i=T+1}^{K} i^{-1} + t_r}$$

and the downtime of a file whose redundancy level is k is given by the binomial

$$d^{(k)} = \sum_{n=k-N+1}^{k} \frac{k!}{n!(k-n)!} d^n (1-d)^{(k-n)}$$

therefore

$$d^* = \sum_{k=T+1}^{K} p_c^{(k)} d^{(k)}$$

Figure 7 data downtime for different values of the recovery ratio h, as a function of the village size K. As expected, downtime is a decreasing function of h (as the recovery procedure is performed before the redundancy level allows a significant probability of enough nodes being simultaneously offline).

B.4.4 Conclusions

Parameters and performance

As a result of the tradeoffs defined by reliability, polling overhead and recovery bandwidth we chose the following parameters:

Block size (N)	24
Village size (K)	36
Recovery threshold (T)	28
Storage size (Z)	100 GB

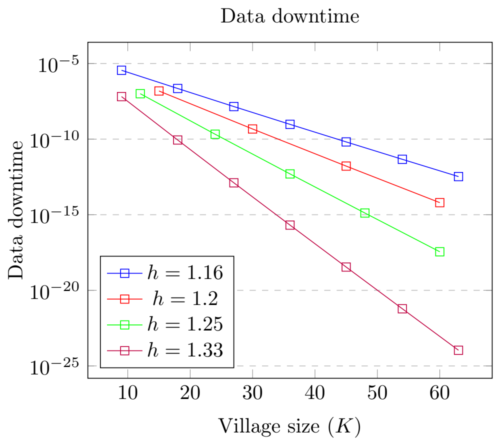

Figure 7: Data downtime for different values of the recovery ratio h, as a function of the village size K. Here we have $r = 1.5$ and $Z = 100$ GB

which result in the following performance:

Data lifetime (L^*)	$7.33 \cdot 10^8$ years
Recovery bandwidth (B^*)	11.5 KB/s
Data downtime (d^*)	$9.51 \cdot 10^{-10}$

Conclusions

In the previous sections, we described our redundancy strategy, designated four redundancy parameters (block size, village size, recovery threshold and storage size), and discussed a model that allowed us to derive analytical expressions for three key performance indexes (data lifetime, recovery bandwidth and data downtime).

Solomon-Reed based redundancy strategy with distributed, real-time data monitoring and recovery procedures proves to be reliable and efficient, and a viable strategy to store data using single board computers, consumer-grade storage devices, and home-grade Internet connections.

Part C
Architecture Challenges

Showing that an efficient and reliable redundancy strategy can be designed to store data on a distributed network of nodes is indispensable, but not sufficient to build a network that can actually be used to securely store and share files. In this section, we will provide an overview on some of the foreseeable challenges for the full development of RAIN.

C.1 Security challenges

As we discussed in the Introduction, the RAIN network should guarantee **confidentiality** (any unauthorized access to the content stored on the network should be impossible), **integrity** (it should be impossible to alter content stored on the network without the tampering being detected) and **availability** (it should be difficult for attackers with limited hardware resources to make some content permanently unavailable or temporarily unaccessible).

The security protocols of traditional client-server paradigms are designed under the assumption that servers will be in physically secure environments beyond the reach of attackers. Discretion over who is granted access to the data, however, is left to the provider of the service and not to the user: in the recent past, many of the largest Internet companies in the world, handling personal data of billions of citizens, have guaranteed access to government-grade attackers [24] [25].

Since each node of the network will be beyond our physical reach, and in the hands of potentially malicious users, RAIN will need radically different security protocols.

Attack model

RAIN's security protocols will be designed under the following **attack model** (namely, a set of axiomatic assumptions about the extent of an attacker's access to our network and its intents, see, e.g., [26]):

Access

1. The attacker will have *read-only* access to **all** the raw data persistently stored in our network.

2. The attacker will have physical access to up to **1**% of the nodes in our network.

3. The attacker will be able to gain physical access to any node in the network in **one day**.

4. The attacker will have at its disposal a computational power comparable to that of the whole world in the foreseeable future.

5. The attacker will **not** have access to unknown computational technology, or know cryptanalytic exploits to **NSA Suite B Cryptography** algorithms (namely, `AES`, `ECDSA`, `ECDH` and `SHA-256/384`).

Intents

1. The **primary objective** of the attacker will be to gain access to plaintext data stored by the network's users.

2. The **secondary objective** of the attacker will be to alter the data stored by the network's user.

3. The **tertiary objective** of the attacker will be to severely disrupt the network's service (e.g., by making large amounts of data unavailable) in order to undermine users' trust in the network and convince them to revert to a classical client-server paradigm.

C.2 Network organization

C.2.1 Honest Geppetto attacks

As we have seen, correlations among failures disrupt data lifetime. For example, whenever $T-N+1$ nodes in the same village will under the control of the attacker, an undetectable **Honest Geppetto attack** (here we use the same nomenclature as in [27]) can be delivered: malicious nodes can behave identically to honest nodes until some recovery threshold is reached. When the redundancy of some files they are hosting reaches T, malicious nodes can simultaneously wipe their content, thus making data permanently unavailable.

C.2.2 Proofs of Persistent Storage

It is easy to see that no distributed storage network can guarantee availability under a Google attack (namely, a variant a Sybil attack where the attacker has under its control an amount of hardware resources far larger than the size of the whole network). Especially in the case of an Honest Geppetto attack, where it is impossible to distinguish malicious from honest nodes before they perform the attack, there is no effective way of selectively storing data on honest nodes, and the probability of that happening by chance is vanishingly low. The only feasible defense against a Google attack is to have a large enough network [27] (or to make its size grow faster than the reward for an attacker to invest its resources in a Google attack).

As per Assumption 2, the attacker might have under its control an extensive amount of physical nodes, that we still assume to be a minority. A malicious node, however, might spawn more daemons than what allowed by its storage space, and go undetected until the actual occupancy of files overflows its capacity. By carefully *overbooking* its storage, without any hardware cost, the attacker might multiply the number of *daemons* under its control.

In order to prevent this from happening, we will make use of *Proofs of Persistent Space*, i.e., protocols that allow one party (*verifier*) to check if the other party (*prover*) is persistently occupying some committed space [14].

PoPS protocols exist with polylog(N) communication complexity [15] that, however, are vulnerable to tradeoffs that, at a cost of CPU, could allow the prover to maliciously save part of its committed space. In order to address this issue, we developed a provably secure, tight-bound PoPS that uses inherently sequential intertwined functions and timeouts to prevent the on-the-go recomputation of the storage commitment.

By forcing each daemon to commit the storage space it is making available, we will prevent the attacker from controlling a larger percentage of daemons than physical nodes.

C.2.3 Authenticated Distributed Hash Tables

Publicly verifiable systems exist (namely, blockchains [13]) that allow distributed bookkeeping on a public ledger of transactions without need for the nodes involved to trust each other, or a third party. Blockchains, however, require each node to keep a full copy of the whole transaction ledger, whose size grows in time.

As we will later see, in order to guarantee availability, we will need to perform some rare operations on the network, like the addition of a new node or the creation of a village, whose result needs to be verifiable by any other node.

In order to do this in a way that is scalable and does not require each node to keep a list of every global transaction ever occurred on the network (as in a blockchain paradigm), a protocol based on *authenticated hash tables* (for related work, see [12] and [28]) is under development.

Our strategy will be to organize the entries of a hash table in a Merkle tree where each value can be retrieved by navigating the tree along a path uniquely determined by the key.

This data structure can also be distributed (*authenticated distributed hash table*) among multiple nodes, each holding one or more entries, and their corresponding Merkle proofs. It is possible to show that non-conflicting Merkle proofs can be merged with each other, which allows for local edits that can be propagated without corrupting pre-existing entry proofs.

A consensus mechanism can then be devised to sequentialize updates based not (as in, e.g., [13]) on CPU-intensive proofs of work, but on the same PoPS in place to prevent Sybil attacks.

Our preliminary results suggest that it should also be possible to partition the set of nodes in interconnected communities bound to verify separate segments of the ADHT. Allowing for parallel validation of updates would be a significant breakthrough in the field of decentralized ledgers, as it would remove the limit to the number of operations per second that can be carried out on the ledger.

A paper on ADHT is under development.

C.2.4 Random beacon

As we will see, many of our protocols will rely on the possibility to generate globally verifiable random numbers. While in principle distributed random number generation could be performed by a Byzantine consensus protocol among all the nodes, its communication complexity would be as high as $\mathcal{O}(N^2)$ [29], which a planetary network, potentially with billions of nodes, cannot afford.

The development of a reliable *random beacon* (as in [30], but free from unilateral management) is therefore of paramount importance in the development of our network.

Using time-hard functions

It has been shown [31] that inherently sequential, time hard functions exist (e.g., square root over $GF(p)$ with $p \bmod 4 = 3$, that requires at least $\log_2(p) - 2$ inherently sequential multiplications on the field) that, while being wallclock-time hard to compute, can be verified with very little computational effort (one multiplication in the $GF(p)$ square root case).

This allows us to implement a timed protocol where multiple nodes can seed the generation of the random number, but a malicious node is unable to adjust its seed to affect the final generated value, because doing so would require a wallclock time effort longer than the window it has to provide its seed.

Using again PoPS as a commitment mechanism to prevent a single node with limited hardware resources to spawn an arbitrarily large number of daemons [15], we can limit the fraction of the seeders pool (namely, the daemons that can contribute to seed the random number generation) that even a very large attacker can control.

Using ADHT, we can now publicly keep track of the number of nodes in the seeders pool, and monitor how many times each daemon contributed to the seed.

Now, if we program each honest daemon to contribute to one seed every, e.g., r/S, S being the number of seeders and r a limited number (e.g., 100), each number will be seeded on average by r independent seeds, and the probability of all of them being malicious (the outcome will be random if at least one of the seeds is random) will decay exponentially in r.

Daemons will be free to randomly choose what random numbers to contribute to, but if a daemon contributes too often (note how this can be monitored with a authenticated hash table), its contributions will be considered invalid and be filtered (this prevents flooding).

This mechanism (which we will describe in more detail in an upcoming paper) is similar to the process of *mining* [13], but with two major differences: the hardware commitment is in terms of storage space, and not computing power and multiple daemons will seed the same random number, thus improving the overall security of the mechanism.

Remark: even if this mechanism can sustainably generate a relatively small amount of random numbers per time unit (e.g., one random number every 30 minutes), a larger amount of random numbers can be generated by using the random beacon to select a limited random subset of nodes from the pool (the set will change every, e.g., 30 minutes), and use a traditional Byzantine consensus algorithm on that subset to generate random numbers at a higher frequency.

Assumptions 2 and 3 can be used to guarantee the security of random numbers generated using this protocol.

C.2.5 Village formation

Once the publicly verifiable generation of random numbers is guaranteed, a protocol to organize daemons into villages will be designed in order to verify the trustworthiness of daemons and to minimize time correlations induced by the attacker without compromising the performance of the network. Trustworthiness will be verified via distributed uptime and IP consistency tests. Memory-hard functions on committed storage can be also used to detect *botnets*. New daemons will be assigned as *freeloaders* to pre-existing villages. New villages will be formed by randomly selecting daemons from the freeloaders pool only when storage is saturated. This allows us to minimize the possibility of the attacker to introduce correlations in the formation of new villages.

C.2.6 Bootstrapping connectivity

In order for data to be stored and retrieved by users on different drives, each node of our network needs to be reachable by any other. This poses a challenge both in terms of networking and security, as on the one hand NAT-traversal techniques and bootstrapping techniques need to be reliable and scalable, and on the other the attacker must be prevented from undermining the connectedness of the network. Non-secure, worldwide network bootstrapping solutions, however, are already in place, developed, e.g., in the scope of trackerless torrent protocol [9], and IPFS protocol [32].

An *exponential network topology* can be designed to guarantee $O(\log(N))$ connectivity and $O(\log(N))$ crossing time, and ADHT can be used to perform timely global bans on the bootstrap network.

C.3 Data

C.3.1 Storage table synchronization

Once daemons are successfully organized into villages, each daemon will have some amount of storage allocated by its fellow villagers. A village-wide ledger will be designed to keep track of new files uploaded and their hashes. Since each villager will be in charge of updating only its part of the ledger, no concurrency protocol needs to be implemented. Incremental synchronization protocol (e.g., `GIT`) already exist to minimize communication overhead and optimize reliability of updates distribution when nodes have a non-negligible downtime.

Villagers will be in charge of monitoring the legality of each other's operations, and a fair bidding protocol will be devised to ban villagers that misbehave or deny access to their data.

C.3.2 Distributed data monitoring

As we discussed, villagers will monitor each other's data availability. This will be done by means of Merkle trees [11] and signatures. This will be non-trivial, as we want to make sure that space involved in a ADHT commitment can still be used to store actual data: PoPSs should serve the purpose to prove that a daemon has some hardware available and uniquely committed to the network, but they should not interfere with how that space is used.

A bidding protocol will be designed to trigger a recovery procedure, along with secure, global ways to find substitute nodes from the freeloaders pool whenever nodes become permanently unavailable.

C.4 Metadata

C.4.1 Synchronization

Data is uploaded by users on their nodes' villages, using the storage separately allocated by each village to their account. When data is organized in file trees and shared with other users, however, synchronization becomes an issue of relevance. As any other piece of data, drive metadata will be encrypted but, unlike data, nodes will be able to edit it so as to make changes to the file tree. In the context of a shared drive, multiple users can simultaneously try to push their edits to the drive's metadata, and atomicity needs to be guaranteed.

The problem is widely discussed in literature, and addressed by *pessimistic algorithms* [33] (that involve acquiring multiple locks before pushing an update to multiple nodes, useful when inconsistencies are either very dangerous or very frequent) and *optimistic algorithms* [34] (that allow for temporary inconsistencies among the copies and focus on quickly resolving them).

C.4.2 Efficient ban

Metadata in a drive is encrypted with a symmetric key, which, when a new user is invited to join the drive, is encrypted with his/her public asymmetric key and shared. When a user is banned, however, a new symmetric key needs to be generated in order to exclude the banned user from access to the data stored in the drive. When the amount of users in the drive is large, however, sharing the new key to all of them is a costly procedure. Similar problems are found, e.g., in the field of premium television channels, where new keys need to be distributed appropriately only to subscribers.

Algorithms exist in literature efficiently addressing the problem [35]. On our side, an $O(\log(N))$ protocol is under development to efficiently share new keys to all the users except one.

C.4.3 Data-metadata synchronization

While atomicity will be guaranteed in the context of metadata synchronization, data and metadata are in principle independent. A procedure needs to be devised to prevent orphan data, i.e., data that is not referenced by any metadata. This can be done by allowing villages to communicate. Whenever data will be stored on a village, it will be marked as temporary until a metadata village confirms to be referencing it. When a reference is removed from a drive, the village storing its metadata will notify the village storing the data to delete the data to save space.

C.5 Credentials

C.5.1 High-entropy authentication mechanisms

In order to prevent the attacker from pursuing its primary and secondary objective (see Assumptions 1 and 2), end-to-end cryptography will be implemented using asymmetric keys generated at the moment of signup. Since, as we said in Section: User Experience, users should be able to access their data from anywhere, a user's private key will need to be persistently stored on the network, encrypted with a key that can be generated by an authentication mechanism. In a client-server context, the encrypted key would shared by the server only when the user proves to be able to decrypt it, thus acting as a bottleneck for any brute-force or hash table attack (for example, a server can lock an account for a few minutes if too many incorrect trials are made). This protects low-entropy passwords from attacks (or, in the case of an attacker in control of the server, completely undermines the security of an end-to-end encryption based on weak passwords). Due to Assumption 1, however, there is no place in the network where we can store an encrypted private key without the attacker being able to retrieve it. Due to Assumption 4, the attacker can perform a brute force or a hash table attack on the key, of which it has the ciphertext.

This issue can be addressed in two parallel ways. On the one hand, the decryption function can be made hard to compute [36]. Even without acting as effectively as a server bottleneck, this will slow down brute force attackers, but it comes at a cost in terms of user experience: the login process to the service should not take more than a few seconds on a standard machine. On the other hand, we will try to increase the entropy of our authentication mechanism. In order to do so without making its solution difficult to memorize, we plan to enforce the use of **passphrases**, rather than passwords, by design. By leveraging on associative memory, this can be done in two ways.

Image-word associations Research has been already carried out on the possibility to use images as a part of an authentication mechanism [37]. Here we propose an example of an image-based authentication mechanism. Upon signup, the user is prompted with a collection of images. Images are representational, but without a uniquely identifiable semantic content. The user is asked to pick a certain number of images (e.g., 10) that are easy to associate to a personal memory, and to associate one or more words to each. During signing, the user is prompted with the images that he/she selected, and asked to enter the corresponding words. If words were uniformly picked among the 2000 most common words in the user's native language, a 10-word passphrase would be approximately as secure as a 110-bit random key. In order verify the hypothesis that word-image associations are easier to memorize and have a higher entropy than passwords, we developed an online experiment where password entropy and memory persistence are measured. Preliminary results are encouraging as far as memorization is concerned, with some users remembering with 100% efficiency up to 286-bit equivalent passphrases. However, some users tend to choose words that describe the image, rather than a memory associated with it. This immensely reduces the final entropy of the passphrase.

Word-word association Word-word associations can be used instead of images [38]. Upon signup, the user is asked to enter a certain number of words (e.g., 10), and associate to each another word with a personal, easily recallable meaning. During signing, the user is prompted again with the words he/she chose, and asked to enter again the corresponding words. In order to filter out trivial associations, we plan to run a crowdsourced free word association experiment, where a network of more common associations is built depending of nationality. This would allow us to provide the user with an *estimate* of his passphrase entropy, and filter out then less robust ones.

C.5.2 Distributed hash tables on GBN

When signing in from a new client, a user will be only asked to enter his username and his passphrase. It will be necessary, therefore, to store distributed login information at a global level on the network, in a way that allows information to be retrieved from anywhere in a limited amount of steps. This can be done by organizing nodes in large redundancy groups (namely, *Distributed Hash Tables* like [9]), and storing login information for each user in a group that can be algorithmically determined from his/her username.

C.6 Environmental impact

RAIN's architecture is distributed and relies on nodes whose hardware is limited both in number of transistors and power requirements. A key milestone will therefore be to study how a large scale transition from a centralized to a distributed paradigm could affect the environmental impact of the Internet.

Here we report preliminary estimates and qualitative arguments for this infrastructure's energy efficiency.

C.6.1 Efficiency arguments

Low-energy nodes The power dissipation for a capacitor C over a voltage V that oscillates with a frequency of ν is given by

$$P = \frac{1}{2}CV^2\nu$$

which defines an essential part of the dissipation for a digital computer: the dissipation of the creation and removal of a bit of information. Single-board computers (like `Raspberry Pi`), on which we plan to develop our infrastructure, are built from components optimized for low-energy performance (e.g., `ARM` processors are often used in cellular phone technologies (see e.g. [39] [40]), where battery life is of paramount importance and cooling is passive). The reduced clock speed and size (`RISC` processors require a smaller number of transistors) of the nodes on which we plan to develop our architecture could therefore make them more energy efficient than faster servers, the overall complexity of each task being equal.

Embedded energy Hardware manufacturing processes significantly contribute to the energy budget of Internet-mediated transactions. For example, [41] shows that the production process of a server represents nearly the 20% of its lifetime overall energy cost. Moreover, as suggested in [42], servers are replaced at an increasing pace to supply the performance needed to offer up-to-pace services, with an expected lifetime currently around 3 years.

Low-energy nodes have a smaller number of transistors and often rely on larger-scale technology (e.g., 14 nm `Skylake` microarchitecture currently used to produce `Intel Xeon` series vs 40 nm technology for `Broadcom BCM2837`, currently in use in `Raspberry Pi 3`). These two effects, along with the reduced size of their components, could have a significant energy saving impact on the manufacturing process.

Datacenter infrastructure It is estimated [43] that approximately 50% of the total power consumption of a datacenter is due to servers and storage, where the rest of the balance mainly includes cooling $\tilde{2}5\%$ and powering of the involved distribution network hardware $\tilde{2}0\%$.

Due to its distributed architecture, and to the low energy nature of its nodes, our infrastructure would be passively cooled, and use an infrastructure that is already in place to service citizens and organizations.

Long-range routing infrastructure The geographical distribution of the nodes in our network would be significantly more fine grained than that of a traditional, centralized paradigm. By storing data on nodes that are geographically close to its users, it could be possible to reduce by orders of magnitude the average routing distance to provide access to the data.

For example, accessing Facebook from Bologna, Italy probably involves a communication with one of its datacenters in Europe, the closest being in Ireland [44], at a linear distance of more than 1500 km. Conversely, for a sufficiently deep service penetration, it is not unreasonable to assume that most of the personal data for a user in Bologna could be stored by a node in Bologna, thus reducing the routing distance to a few kilometers.

It is not easy, however, to estimate the energetic impact that this change of paradigm would have on the current routing infrastructure, which is optimized to service a centralized architecture. For example, it has been shown [45] that a significantly larger expenditure of energy in communication networks is due to *access networks* (i.e., the part of the routing infrastructure that directly provides access to the users) than to *core networks* (i.e., the faster routing infrastructure that carries data in long-range communications). Shorter range communications could therefore be less optimized, not due to inherent properties in the routing technology, but to optimization for a centralized paradigm.

Intensive data processing A centralized architecture needs a constant stream of revenue in order to be sustainable. As we have seen in the Part A, some of the largest owners of datacenters make intensive use of the data collected through their services to generate profit. Independently of the privacy issues that this practice raises, it is easy to argue that it comes at a significant cost in terms of computing power.

A distributed architecture could be sustained locally by the same community that uses it. This would make it unnecessary to find different, privacy-invasive and CPU-consuming ways of processing data to, e.g., provide better targeted advertisement services.

C.6.2 Node efficiency comparison

In order to determine an order-of-magnitude energy efficiency ratio between a single-board computer and a server, we performed a set of cryptographic benchmarks on a `Raspberry Pi 3` single-board computer and a `Dell PowerEdge R815` server, mounting 4 `AMD Opteron 6328 8-Core 3.2 GHz`.

Remark: we chose to benchmark cryptographic operations because of the static nature of the content stored and distributed by our network: in a real case scenario where the CPU cost of secure delivery of content over the Web is reduced to the cost of an `HTTPS` transaction, it was shown that 70% of the CPU cost was due to `SSL` processing [46].

Experimental procedure The benchmarks were run on the `Raspberry PI` on a clean installation of `Raspbian Jessie`, and on the `PowerEdge` on a clean installation of `Springdale Linux 6`.

We used the benchmark software integrated with the `OpenSSL` library. Both platforms used `OpenSSL 1.0.1e-fips`. `AES` encryption was benchmarked with 256 bit keys in `CBC` mode.

The benchmark was ran both in single-thread mode and in multi-thread mode, using all the available cores (4 and 32, respectively). 15 single-thread and 20 multi-threaded benchmarks were ran on each platform.

Remark: during the test we observed a significant dependency of the speed of the `Raspberry Pi` on its temperature. No cooling measure was taken, and the multi-threaded tests were looped repeatedly and the values taken when the speed stopped showing a decreasing trend.

Our results are as follows (all the values are in MB/s):

	1 thread	> 1 thread
`RPI`	36.90 ± 0.02	96.9 ± 0.2
`PEdge`	175.26 ± 0.08	3981 ± 5

Values for the maximum loads power consumption of `Raspberry Pi` and `PowerEdge` were obtained from [47] and [48] respectively:

	Maximum load power
RPI	3.7 W
PEdge	563 W

Which result in an encryption efficiency of **26.2**MB/J for the Raspberry Pi, against **7.1**MB/J for the PowerEdge.

Obviously, more detailed investigations are needed to determine the energy and environmental impact of the proposed distributed datastorage architecture as well as how it compares to the current centralized datastorage paradigm.

Part D
Potential Impact

A distributed, open-source, community-owned infrastructure for the storage, distribution and manipulation of information has multiple applications beyond the storage of personal and other sensitive user data. Here we propose some in the fields of *Services*, *Finance* and *State Administration.*

D.1 Services

RAIN's first goal is to reliably and securely store private data. This service can be offered in a scalable way, as its design involves connecting more nodes to the network as new users start using the service. A wide variety of critical services, however, could be hosted on RAIN's infrastructure once its acquisition is deep enough. Here we propose some examples.

D.1.1 Messaging platform

Due to the simultaneous scaling of demand and offer (the more users use the service, the more nodes are connected to it) and to the relatively low bandwidth requirements (whose bottleneck is often determined by mobile phone connection speeds), messaging services are among the easiest additional services that could be implemented on top of a pre-existing storage and distribution infrastructure for personal data.

Indeed, classical storage villages could be used as a means to store outgoing messages, available for any recipient to be retrieved. Messages could be end-to-end encrypted using the same strategy that guarantees the security of shared files. In principle, this could guarantee an uncircumventable secrecy of communications, while allowing our users to exchange messages and media without having to alter their usual user experience.

Real-world scenario: WhatsApp

Throughout 2016, one billion of WhatsApp users has sent an average of $64 \cdot 10^9$ messages per day, and $1.6 \cdot 10^9$ images [49].

Using the online interface WhatsApp Web and a network inspector, we selected 50 random images from a WhatsApp conversation to determine how images are compressed when dispatched, and we determined an average image size of (111 ± 8) KB.

Overestimating a text message size to 1 KB, we therefore obtain a total service bandwidth of 241.6 TB/day, or 2.79 GB/s.

As a rough estimate, a secure messaging service equivalent in size to WhatsApp could therefore be hosted on RAIN's infrastructure by dedicating 1% of the upload bandwidth of $1.40 \cdot 10^6$ of its nodes (here we used $S_u = 0.2$MB/s, recall Section B.1).

D.1.2 Content Delivery Network

A personal data storage service involves storing a large amount of data with a rapidly decaying distribution of popularity: files are accessed only to those users to which they were explicitly shared. Publicly available content follows a more heavy-tailed distribution of popularity [50].

In order to provide access to very popular data a one-to-one redundancy strategy, where the number of copies for each file is tuned only to prevent data loss, is not sufficient.

A large enough network already deployed to store personal data could make use of the spare storage resources of its users and aid e.g. the distribution of web pages with an unprecedented geographical extensiveness. Locally cached data would be faster and less expensive to distribute, and long-range routing infrastructure would see its load reduced.

A peer to peer Web content delivery network is already under investigation by the IPFS project [51], relying for the distribution of content on the personal computers of users accessing the content. A similar protocol could be implemented on top of RAIN's infrastructure with the following advantages:

- **Uptime**: dedicated, permanently online embedded computers would have a higher uptime than personal computers. This would significantly reduce the redundancy needed to make a file available to large audiences, as each node would be providing the same content for a more extended amount of time.

- **Storage space**: the protocol could make use of a node's spare storage space to cache Web content. That space would already be allocated by the user to mass storage purposes, therefore the protocol would not be in competition for resources with the rest of the user's environment.

- **Computational resources**: the protocol would run on dedicated machines instead of personal computers. The user experience would therefore be unaffected by the protocol.

Real-world scenario: Wikipedia

Throughout February 2017, the online encyclopedia Wikipedia had $585 \cdot 10^6$ pageviews [52], for a total size (late 2014) of 23 TB [53]. By using the *Random article* feature, and a network inspector, we sampled the data transferred to load 300 random Wikipedia articles. The average data transfer per article is 243 KB, with a standard deviation of 195 KB. Under the approximation of Gaussian distribution of sizes, the error on the average is 11 KB.

The above sums to 142 TB/month, or 54 MB/s. As a rough estimate, Wikipedia could therefore be hosted on RAIN's infrastructure by dedicating 1% of the upload bandwidth of $54 \cdot 10^3$ of its nodes (here we used S_u as per Section B.1).

D.1.3 Social network

RAIN's infrastructure is designed to guarantee privacy-preserving sharing of files among its users while organizing data in a way consistent to that of a social network. Preserving the network's cryptographic architecture could be seen just as an extension to the software already in place.

Having to manage only a limited number of accounts, each node in the network could store and process information dispatched by others, related to the users to which its owner is connected. Moreover, while a universal protocol could be established to distribute data across the network, the way contents are organized and offered to the user would depend on software running on each user's node. This would allow on the one hand the transparency, and on the other the possibility to personalize the organization of content, as each node could organize its user's content in a different way depending on the software it is running.

Real world scenario: Facebook

As of 2014, the total storage capacity of Facebook, the largest social network in the world, was 300PB [54]. User base growth data is available [55] and shows an approximately linear growth from $1.276 \cdot 10^9$ users during the first quarter of 2014 to $1.860 \cdot 10^9$ at the end of 2016. Under the assumption of a steady production of data by its users, we can extrapolate Facebook's current storage capacity by multiplying the old figure by the square of the users' ratio, obtaining approximately 650 PB.

Under the assumption of each node having a storage capacity of 1 TB, and dedicating 1% of the storage capacity of each node to hosting an equally large social network, approximately $65 \cdot 10^6$ nodes could host the same $1.86 \cdot 10^9$ social network accounts, while providing privacy, transparency and customizability to its service.

D.1.4 Search engine

Multiple projects (e.g., [56] [57]) exist to implement distributed crawling, indexing and data mining to provide independent, open-source search engine services. Distributed crawling would be a task especially suited for a network already providing CDN services, as the crawling and indexing could be performed in-place by the same nodes hosting the content, saving the bandwidth needed to download the content and index it on a separate server. Multiple computing infrastructures [3] already implement similar mechanisms, sending code to be locally executed on the machine hosting the data rather than moving the data to enable its processing on a separate service.

Real-world scenario: Google

As of 2016, Google's index comprises $130 \cdot 10^{12}$ distinct pages, for a total size of 100 PB [58]. Under the overestimate of a 10% daily update rate, a distributed network of $100 \cdot 10^6$ non-overlapping crawlers could keep the database updated with as few as 1.5 page requests per second per node. Storing the index in a Distributed Hash Table with $10\times$ redundancy, the same amount of nodes could store the whole index with an occupancy of 10 GB per node.

Such a large-scale distributed database, however, would likely require the design of new indexing and routing algorithms optimized for distributed systems to make the performance

[3]For example, the WLCG (Worldwide LHC Computing Grid), which stores and processes the data produced by CERN's Large Hadron Collider, runs data analysis programs directly on the nodes storing the data, as it is easier to move programs than the data they need.

of a query comparable to that of a centralized paradigm. As of 2009, a query to the Google index involved an order of magnitude of 10^3 distinct servers [59], resulting in an extensive use of an internal datacenter bandwidth whose performance cannot be matched by that of a distributed system.

D.2 Cryptocurrency

As we said in Section C.2.3, a large enough set of nodes with a relatively high uptime, organized in a network where the number of daemons per node is limited by a hardware commitment, and managed for some critical aspects by a trustworthy or verifiable publicly available random beacon, can be used to implement verifiable bookkeeping of a distributed ledger by means of authenticated hash tables.

By means of verifiable hash tables, a set of account balances can be efficiently and securely stored, thus implementing a cryptocurrency with the following advantages:

- Due to the low communication and CPU complexity of the process, it could address the issue of the environmental impact of cryptocurrencies (like `Bitcoin` [60]), whose security relies on the constant solution of CPU-intensive cryptographic problems [13].

- Its constant persistent space complexity would guarantee scalability: while on the one hand the most commonly used cryptocurrencies require each user to store the list of all the transactions ever occurred on the network, the memory requirement for this one would not increase with time. Indeed, authenticated hash tables would store the *state* of the ledger, rather than the list of all its updates: while T transactions involving, e.g., the same two accounts in a blockchain-based cryptocurrency would result in a permanent storage requirement of T new entries in the blockchain for all the nodes in the network, in an AHT-based cryptocurrency those updates would just successively change the value of the same field.

- Cryptocurrencies based on mining encourage the formation of *mining pools* (namely, groups of nodes that join their computational power in order to solve cryptographic problems more efficiently). However, it has been shown [61] [62] that a coordinated group of users controlling the absolute majority, e.g., of the `Bitcoin` network's computing power could disrupt its security. This is far from being a theoretical weakness, as, for example, a single mining pool has reached 50% of the total `Bitcoin`'s mining power in June 2014 [63]. A cryptocurrency where mining pools are not rewarded would help avoid this critical scenario, while addressing the democracy issue inherent in the high entry point of cost-efficient mining hardware [64].

D.2.1 Transaction syntax

Authenticated hash tables allow any node with $O(1)$ previous knowledge on the status of the hash table to verify that an edit has been done to it. By grouping one or more edits together, it is possible to define a set of *legal transactions* and, by recursion, a *legal hash table* as an empty hash table or any hash table obtained from a legal transaction on a legal hash table.

For example, a basic cryptocurrency protocol could involve the following transactions:

Signup: a new entry is added to the hash table. The key is a user name, the content includes an initial balance of zero and the user's public key. The transaction includes the proof of the addition of the entry to the hash table, and a signature with the user's public key.

Payment: a user A is paying another user B some amount. The transaction involves a proof of the existence of A, a proof of the existence of B, the balance of A, the balance of B, the public key of A and a signature with the public key of A describing the transaction. The proof also includes an edit of A's entry (reducing its balance by the amount of the transaction) and an edit of B's entry (increasing its balance by the amount of the transaction). The transaction is accepted if both the users exist, if A has enough money in its balance, and if a valid signature is provided.

In this paradigm, the initial currency distribution process could be uniform in time (e.g., every account receives some amount per time unit, the time being measured e.g. in number of transactions) or exponential (i.e., each account receives proportionally to its balance). This eliminates the need for energetically expensive mining by replacing it with *minting*: currency is initially distributed to all the nodes that take part in ensuring the security of the network, e.g. by storing proofs of persistent storage and double-checking the stream of updates received by the network. Note how both activities are not CPU and energy intensive.

Since authenticated hash tables can be used to provably add, retrieve and remove entries from a hash table, depending on the set of legal transactions one defines, it is possible to implement a wide variety of financial constructs in the cryptocurrency.

D.2.2 Universal transaction syntax

As we have seen, a globally accepted transaction syntax could allow a set of peers offering a hardware commitment to check a stream of updates to a distributed ledger. This allows us to easily define *universal sytanxes*. Consider the case, for example, of a syntax where:

- A set of fields in the distributed ledger define a finite set of *rules*, e.g. in the form of regular expressions.
- All updates are sent along with the proof of the rule they are applying. Every node can verify that the rule provided actually lies in the ledger, and check that the rule is respected by the update.
- Rules can exist to update every field, also those that store rules. For example, one or more rules can define a distributed voting mechanism to update the fields storing the rules themselves. This would allow users in the network to define, and arbitrarily update the transaction syntax without having to alter the software that checks that they are respected.

D.2.3 Taxation system

Some cryptocurrencies already implement, or plan to implement, transaction fees as a form of redistribution of wealth after the mining resources have been exhausted.

Instead of implementing a fee for accepting a block, thus splitting the value of the fee among multiple accounts, one or more *community accounts* could be included in the cryptocurrency. The syntax of a transaction could involve, for example, paying a small fraction of the amount to a community account.

Since each transaction would be verified by the whole community, tax evasion would be unfeasible: our cryptocurrency would implement taxes in the very language used for transactions. Instead of putting the community accounts under the control of any trusted third party, they could rather be managed by distributed, verifiable algorithms.

Sponsoring FOSS, Creative Commons & free knowledge

In Section D.1.2, we described how a Content Delivery Network could be hosted on top of a large enough personal data storage network like RAIN. Integrating that possibility with a taxable cryptocurrency integrated in the same network could have remarkable implications in terms of how knowledge is produced and made available.

For example, authors of Free and Open Source Software, Creative Commons Media and free knowledge in general could make their content available via RAIN's distribution infrastructure. A distributed, verifiable algorithm could then monitor how much each content is accessed and used, and use funds from the community account to appropriately reward its authors.

The possibility for authors to be sponsored by the community in the production of freely accessible content would significantly contribute in moving past a paradigm where content is either sold (and mostly brokered by progressively useless third parties that make profit on its distribution) or made available for free on a volunteer basis.

D.2.4 Privacy vs taxability

Among the arguably most significant factors that limited the large-scale acquisition of cryptocurrencies as a viable financial instrument is the concern that the anonymity they provide would make it simpler to evade taxes [65] [66] and to fund illegal [67] [68] and terrorist [69] activities.

In particular, in the case of cryptocurrencies where all accounts are anonymous and new accounts can be easily opened, the issue of fair taxation is highly significant. Most taxation systems, in fact, implement a progressive mechanism where those who earn more have to pay a larger percentage of their income in tax. In a context where there is no way to trace accounts back to their users, there is no way to implement this, as any user can easily split its resources across a large amount of anonymous alias accounts under his/her control.

On the other hand, we want to make sure to protect the privacy of the transactions for each user. In order to address the problem posed by the trade-off between anonymity and taxability / traceability, we have developed a toy model cryptocurrency based on ADHT (Authenticated Distributed Hash Tables) that, if proven to be secure, could offer a good compromise between the two.

Goal of the model In this model, we have a set of citizens that make use of an ADHT to implement a cryptocurrency, and an *authority* (their government). Citizens want their privacy safeguarded from mass surveillance programs, but they also want to make sure that everyone pays a fair tax, and that the under certain conditions (e.g., provided with a warrant) the authority will be able to uncover the transactions of any specific user.

The transaction syntax Our model will be simplified in that the amount of tax paid by each account will be a function of the amount of currency received by that account during, e.g., the previous month.

We will use the following *transaction syntax* for our cryptocurrency (see D.2.1) in order to provide all the properties described in the previous paragraph:

Signup: in order to open an account, a user will need to first receive an identifier from the authority. The authority will require the user to provide his/her personal details in order to receive an identifier. The identifier will be signed by the authority and will not publicly disclose the identity of the user. The authority, however, will know which identifier belongs to which user.

Splicing: two users can create two *ephemeral accounts* and transfer to them a matching amount of currency. The transaction will only be accepted if signed by both the users and a set of *witnesses* (namely, owners of non-emphemeral accounts), randomly selected by a global random beacon.

A splicing transaction will occur in this form:

$$(a, b) \xrightarrow[(a,b,w_0,\ldots,w_{K-1})]{n} (c, d)$$

where a and b are accounts, n identifies an amount of currency and those indicated under the arrow are the signatories of the transaction. Both a and b will spend n units of currency and two new accounts, c and d, will be opened with n units of currency in them. Only the signatories will know which ephemeral account belongs to which user (namely, if a's owner is also c's owner or if a's owner is also d's owner).

An ephemeral account cannot receive money, and it can only make one operation involving its whole balance before being closed, i.e., it can be used to create a new ephemeral account or to proceed to a payment to a non-ephemeral account (see next transaction). Ephemeral accounts also expire (if they are not used for longer than a certain time limit, they are locked and absorbed in a community account).

Payment: as in our first cryptocurrency example, any account can make a payment, but only non-ephemeral accounts can receive a payment. When a non-ephemeral account receives a payment, it will publicly add the corresponding amount to a log that will be then used to pay the necessary tax. This allows progressive taxations: accounts *are* nominal, but transactions are obfuscated: an account that receives more can now be subject to paying more.

Ratting out: when provided with a warrant signed by the authority, a witness to a splicing transaction will publicly log the correspondence between input and output accounts involved in the transaction, encrypted with the authority's public key.

In this model, while indeed the authority has a complete correspondence table between citizens and accounts, each account can make anonymous transaction by mean of a sequence of splicing transactions.

Let N be the total number of accounts in the cryptocurrency. At each splicing transaction, a new ephemeral account is opened and the indetermination on its owner doubles, as, in the limit $2^n \ll N$, each ephemeral account can be tracked back to $O(2^n)$ non-ephemeral accounts, n being the number of splicing transactions have contributed to the generation of the ephemeral account.

The existence of witnesses, however, guarantees that under certain circumstances (e.g., provided with a warrant) the authority can force a witness into uncovering which account generated which during a splicing transaction. This allows it, if needed, to successively track a transaction back to the non-ephemeral account that generated it. However, due to the fact that the result of the operation is publicly logged by the witness, the authority will have no way of implementing an undercover mass surveillance program, as each ratting out transaction is visible to all the citizens, that can then decide to limit the power of the authority, or select a new one.

D.2.5 Peer to peer lending & microcredit

Lending has traditionally been mediated by financial institutions that would lend private citizen's financial resources stored in bank accounts.

It is easy to see, however, that loans can be implemented in the syntax of a cryptocurrency based on authenticated hash tables. A peer to peer lending and microcredit platform could therefore be integrated in the cryptocurrency without need for third party mediation. A similar paradigm is implemented by *crowdfunding platforms*, that still act as lending mediators, but grant the user control over what one's money is used for.

In order to address the limited economic dynamicity that could stem from relying on private citizens to actively loan their resources, publicly verifiable, democratic lending pools could be formed to cooperatively select whose loans to grant.

Automatic relending options could also be provided by the cryptocurrency, since not only *data*, but also *scripts* can be stored in a authenticated hash tables and executed in a verifiable way at appropriate times.

D.3 Discussion

We have introduced a distributed communication and data storage architecture that provides a viable alternative to the growing centralized data storage solutions.

Initially (Part B) we demonstrate the feasibility of the proposed architecture. (a) We show that expected data lifetimes of the same order of magnitude as the age of the Earth can easily can be obtained using Solomon Reed redundancy strategies ($\sim 7 \cdot 10^8$ years) with a redundancy factor of 1.5 using 35 data storage nodes. (b) Based on measurement of end-user grade Internet connectivity and `Raspberry Pi` data storage nodes, we demonstrate that the RAIN architecture performance easily match or exceed that of the central data storage paradigm. We measured average data upload speed ($\sim$17Mbit/s) and download speed ($\sim$34Mbit/s) as well as estimated data recovery bandwidth ($\sim$12 KB/s) and downtime probability ($\sim 10^{-9}$) exceeding the existing centralized data storage paradigm. (c) Finally, the end-user costs for getting access to the proposed RAIN data storage architecture are significantly lower than the costs based on the centralized data storage paradigm.

Secondly (Part C) we define the scientific and technical challenges for designing data privacy and security, network organization as well as data, metadata and credential handling. In this part we also sketch the energy and environmental issues for the proposed distributed architecture and how it differs from large scale server centers. This part outlined the necessary steps involved in developing the RAIN data storage architecture.

Finally (Part D) we provided an overview and discuss the potential disruptive impact of a distributed, privacy and security by design, community owned digital infrastructure. We believe a discussion of potential impact is also an important component of the architecture design. In reflecting on the impact we in particular need to evaluate the critical connection between, on the one hand, cyberprivacy and security, and on the other hand, the potential political power structure within a modern society.

RAIN could support the development of communitarian services including telecommunication, content delivery, cryptocurrency, and distributed administration (nation state and regional governmental), which currently are services managed in a centralized manner through trusted third parties. Implementation of a RAIN style architecture could thus distribute the power from global centralized trusted third parties to local citizens and business, while at the same time presumably reduce the significant energy requirement and resulting CO^2 impact of centralized data storage.

Acknowledgements

We are grateful for partial financial support from the European Commission sponsored SYNENERGENE project as well as for constructive discussions with and suggestions from Sofia Farina, Per Odling, Leif Rasmussen and Piper Stover. Lucinda Voldsgaard is acknowledged for proof reading of the manuscript.

References

[1] A. Vaughan, "How viral cat videos are warming the planet," *Independent.* https://www.theguardian.com/environment/2015/sep/25/server-data-centre-emissions-air-travel-web-google-facebook-greenhouse-gas.

[2] T. Bawden, "Global warming: Data centres to consume three times as much energy in next decade, experts warn," *Independent.* http://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html.

[3] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers Inc., 1999.

[4] "Boinc - open-source software for volunteer computing." http://boinc.berkeley.edu/.

[5] M. M. Gaber, F. Stahl, and J. B. Gomes, *Pocket Data Mining - Big Data on Small Devices*, vol. 2 of *Studies in Big Data*. Springer International Publishing.

[6] S. A. et al., "Mobile-edge computing – introductory technical white paper." https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf.

[7] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, pp. 300–304, 1960.

[8] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books.

[9] A. Loewenstern and A. Norberg, "Dht protocol." http://www.bittorrent.org/beps/bep_0005.html.

[10] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in p2p systems," *Commun. ACM*.

[11] R. C. Merkle, "A digital signature based on a conventional encryption function," *Advances in Cryptology - CRYPTO 1987*, p. 369, 1988.

[12] A. Miller, M. Hicks, J. Katz, and E. Shi, "Authenticated data structures, generically," *SIGPLAN Not.*, vol. 49, no. 1, pp. 411–423.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." https://bitcoin.org/bitcoin.pdf.

[14] L. Ren and S. Devadas, "Proof of space from stacked expanders," *Cryptology ePrint Archive*, 2016.

[15] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," *Advances in Cryptology - CRYPTO 2015*, pp. 585–605, 2015.

[16] B. Beach, "How long do disk drives last?." https://www.backblaze.com/blog/how-long-do-disk-drives-last/.

[17] G. Gasior, "The ssd endurance experiment: They're all dead." http://techreport.com/review/27909/the-ssd-endurance-experiment-theyre-all-dead.

[18] M. M. et al., "State of the internet report / q3 2016 report." https://www.akamai.com/us/en/our-thinking/state-of-the-internet-report/.

[19] "Speedtest market reports." http://www.speedtest.net/reports/.

[20] V. Strassen, "Gaussian elimination is not optimal.," *Numerische Mathematik*, vol. 13, pp. 354–356, 1969.

[21] R. Williams, "Matrix-vector multiplication in sub-quadratic time: (some preprocessing required)," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pp. 995–1001, Society for Industrial and Applied Mathematics, 2007.

[22] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *5th USENIX Conference on File and Storage Technologies*, FAST 2007, 2007.

[23] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you?," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST 2007, USENIX Association, 2007.

[24] G. Greenwald and E. MacAskill, "Nsa prism program taps in to user data of apple, google and others." https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data.

[25] G. Greenwald, E. MacAskill, L. Poitras, S. Ackerman, and D. Rushe, "Microsoft handed the nsa access to encrypted messages." https://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data.

[26] "Attack model." https://en.wikipedia.org/wiki/Attack_model.

[27] S. Wilkinson, T. Boshevski, J. Brandof, J. Prestwich, G. Hall, P. Gerbes, P. Hutchins, and C. Pollard, "Storj - a peer-to-peer cloud storage network." https://storj.io/storj.pdf.

[28] S. A. Crosby and D. S. Wallach, "Authenticated dictionaries: Real-world costs and trade-offs," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 2, pp. 17:1–17:30.

[29] J.-P. Martin and L. Alvisi, "Fast byzantine consensus," *IEEE Trans. Dependable Secur. Comput.*, vol. 3, no. 3, pp. 202–215.

[30] "Nist randomness beacon." https://www.nist.gov/programs-projects/nist-randomness-beacon.

[31] A. K. Lenstra and B. Wesolowski, "A random zoo: sloth, unicorn, and trx," *IACR eprint archive.*

[32] "libp2p - modular peer-to-peer networking stack." https://github.com/libp2p.

[33] P. A. Bernstein and N. Goodman, "The failure and recovery problem for replicated databases," in *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pp. 114–122, 1983.

[34] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Comput. Surv.*, vol. 37, no. 1, pp. 42–81.

[35] Y. Dodis and N. Fazio, *Public Key Broadcast Encryption for Stateless Receivers*, pp. 61–80. Springer Berlin Heidelberg.

[36] C. Percival, "Stronger key derivation via sequential memory-hard functions."

[37] M. Gibson, M. Conrad, and C. Maple, "Evaluating the effectiveness of image-based password design paradigms using a newly developed metric." http://perisic.com/preprints/EvaluateImagePasswordMetrics.pdf.

[38] R. Pond, J. Podd, J. Bunnell, and R. Henderson, "Word association computer passwords: The effect of formulation techniques on recall and guessing rates," *Computers & Security*, vol. 19, no. 7, pp. 645–656, 2000.

[39] "iphone 7 - technical specifications." http://www.apple.com/iphone-7/specs/.

[40] Samsung Galaxy S7, "Samsung galaxy s7 — Wikipedia, the free encyclopedia," 2017. [Online; accessed Apr. 7, 2017].

[41] K. Craig-Wood and P. Krause, "Towards the estimation of the energy cost of internet mediated transactions," *Preliminary version of a technical report of the EEC (Energy Efficient Computing) SIG of the ICT KTN (Knowledge Transfer Network)*, 2013.

[42] J. Edwards, "New technologies mean shorter server life cycles."

[43] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.

[44] "Clonee data center - facebook." https://www.facebook.com/CloneeDataCenter/.

[45] S. Lambert, W. V. Heddeghem, W. Vereecken, B. Lannoo, D. Colle, and M. Pickavet, "Worldwide electricity consumption of communication networks," *Opt. Express*, vol. 20, no. 26, pp. B513–B524.

[46] L. Zhao, R. Iyer, S. Makineni, and L. Bhuyan, "Anatomy and performance of ssl processing," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2005*, ISPASS '05, pp. 197–206, IEEE Computer Society, 2005.

[47] "Power consumption." https://www.pidramble.com/wiki/benchmarks/power-consumption.

[48] "Power efficiency comparison of the dell poweredge r815 and hp proliant dl585 g7 rack servers." http://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Dell-2u-R815-versus-HP-4u-DL585.pdf.

[49] "How many text and photo messages are sent using the top messaging apps globally each day?." https://askwonder.com/q/how-many-text-and-photo-messages-are-sent-using-the-top-messaging-apps-globally-each-day-57738fa9b3159159003df760.

[50] L. A. Adamic and B. A. Huberman, "Zipf's law and the internet," *Glottometrics*, vol. 3, pp. 143–150, 2002.

[51] J. Benet, "Ipfs - content addressed, versioned, p2p file system." https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf.

[52] "Page views for wikipedia, both sites, normalized." https://stats.wikimedia.org/EN/TablesPageViewsMonthlyCombined.htm.

[53] Wikipedia, "Wikipedia:Size of Wikipedia — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Wikipedia%3ASize%20of%20Wikipedia&oldid=775138473, 2017. [Online; accessed 13-April-2017].

[54] P. Vagata and K. Wilfong, "Scaling the facebook data warehouse to 300 pb."

[55] "Number of monthly active facebook users worldwide as of 4th quarter 2016 (in millions)." https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/.

[56] "Majestic-12: distributed search engine." https://www.majestic12.co.uk/.

[57] "Yacy - the peer to peer search engine." http://yacy.net/en/index.html.

[58] "How search works - the story."

[59] J. Dean, "Challenges in building large-scale information retrieval systems."

[60] K. J. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," in *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, pp. 280–285.

[61] *Andes*, "Bitcoin's kryptonite: The 51% attack.." https://bitcointalk.org/index.php?topic=12435.

[62] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," 2013. https://arxiv.org/abs/1311.0243.

[63] R. Gill, "Cex.io slow to respond as fears of 51 http://www.coindesk.com/cex-io-response-fears-of-51-attack-spread/.

[64] S. Valfells and J. H. Egilsson, "Minting money with megawatts," in *Proceedings of the IEEE*, vol. 104, pp. 1674–1678, 2016.

[65] O. L. Bateman, "Bitcoin might make tax havens obsolete," *Motherboard*. https://motherboard.vice.com/en_us/article/bitcoin-might-make-tax-havens-obsolete.

[66] E. Zwirn, "No, you can't avoid taxes by investing in bitcoin," *New York Post*. http://nypost.com/2017/04/08/no-you-cant-avoid-taxes-by-investing-in-bitcoin/.

[67] J. Bearman, "The untold story of silk road, part 1," *Wired*. https://www.wired.com/2015/04/silk-road-1/.

[68] J. Bearman, "The untold story of silk road, part 2: The fall," *Wired*. https://www.wired.com/2015/05/silk-road-2/.

[69] S. Higgins, "Isis-linked blog: Bitcoin can fund terrorist movements worldwide," *Coindesk*. http://www.coindesk.com/isis-bitcoin-donations-fund-jihadist-movements/.