A Measurement Study of the Wuala On-line Storage Service

Thomas Mager, Ernst Biersack, and Pietro Michiardi
EURECOM
Sophia Antipolis, France
{mager,erbi,michiardi}@eurecom.fr

***Abstract*—Wuala is a popular online backup and file sharing system that has been successfully operated for several years. Very little is known about the design and implementation of Wuala. We capture the network traffic exchanged between the machines participating in Wuala to reverse engineer the design and operation of Wuala. When Wuala was launched, it used a clever combination of centralized storage in data centers for long-term backup with peer-assisted file caching of frequently downloaded files. Large files are broken up into transmission blocks and additional transmission blocks are generated using a classical redundancy coding scheme. Multiple transmission blocks are sent in parallel to different machines and reliability is assured via a simple Automatic Repeat Request protocol on top of UDP. Recently, however, Wuala has adopted a pure client/server based architecture. Our findings and the underlying reasons are substantiated by an interview with a co-founder of Wuala. The main reasons are lower resource usage on the client side, which is important in the case of mobile terminals, a much simpler software architecture, and a drastic reduction in the cost of data transfers originating at the data center.**

I. Introduction

Storing personal data online using cloud-based storage services such as Amazon S3 [1], Google Docs [2], and DropBox [3] has become business and an effective solution for the needs of a growing number of users. As a complement to cloud-based solutions, online storage systems based on a peer-to-peer (P2P) design have been investigated in academia to cope with problems related to cost for long-term storage [4], security [5] and data lock-in [6].

In this work, we focus on a popular online backup and file sharing system, called Wuala[1], that has been successfully operated for several years. Wuala is particularly interesting because when it was launched in 2008 it had adopted a hybrid design, making use of servers in a data center as well as leveraging resources of the participating peers. However, not much is known about the design of Wuala. In this work we present some of the salient features of Wuala and discuss its recent evolution.

In summary, the goal of this paper is to

- Characterize the *infrastructure* of Wuala such as the number of servers involved in the operation of Wuala and estimate the number of peers
- Determine the *data placement* adopted by Wuala to understand the mechanisms used to decide where data is stored
- Understand if and how *coding techniques* are used to assure the availability and durability of the data in case of node failures
- Determine the *transport protocol* used to move data between peers and servers
- Describe the evolution Wuala has undergone between 2010 and 2012.

Our findings indicate that Wuala uses a simple, yet clever system design. Data *availability* – i.e., making sure that files are accessible at any time – and *durability* – that is ensuring that files are never lost – are achieved by relying on servers located in a data center, instead of using peers. This simplifies the software architecture and avoids costly maintenance operations to cope with peer churn. In Wuala, user data is first interleaved and encoded before it is transmitted via a UDP-based transport protocol, which allows for parallel data transfers to improve transfer performance. Our results also reveal that in Wuala peers are only used as distributed caches that off-load the servers when delivering frequently-accessed data.

Recent measurements show that Wuala has evolved towards a pure client-server model: peer resources are no longer used and data storage is offered as a cloud-based service addressing mainly business customers.

The remainder of this paper is organized as follows. In Section II we explain our experimental methodology used. In Section III we overview the Wuala architecture, followed by a discussion on how data is managed in Section IV. We describe the anatomy of uploading and downloading files, respectively in Section V and VI. In Section VII we describe the reliable transport protocol of Wuala. We finally give some details on recent changes to Wuala and discuss why peer-to-peer based systems seem to be getting out of fashion before we conclude the paper.

II. General Experimental Setup

We perform a series of experiments and measurements to elucidate the design and operation of Wuala. This is necessary since the Java bytecode is obfuscated [7], which prevents the analysis of Wuala by studying its source code. For our experiments, we run several Wuala clients in our Lab and capture all the network traffic between our Wuala clients and the rest of the Internet. We use two tools to analyze the traffic:

[1]See http://www.wuala.com/.

- Wireshark [8] to inspect and analyze the content of individual packets or a sequence of packets that are exchanged between two machines.
- Tstat [9] to compute the relevant information about all the connections observed such as the total number of bytes exchanged in each direction, duration of the exchange, round trip time, loss rate (in case of TCP connections), etc. Tstat allows us to identify all the other machines our Wuala client communicates with, the amount of data exchanged, and the transfer rates achieved.

The Wuala application creates some directories on the local client. We observe the content of these directories and record when files are created, modified, or deleted as well as their size.

Furthermore, by parsing the monitoring interface provided by the Wuala application on port 33333 we are able to obtain additional information on the application state such as the current on-line status as well as events related to incoming transmission blocks.

Our setup is summarized in Figure 1.

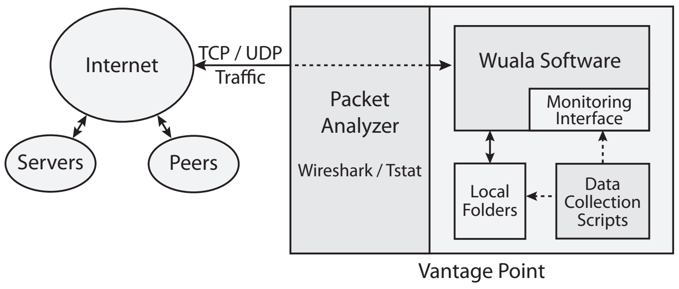


Figure 1. Experimental Setup

Our client machines are located in France. Most of the experiments were carried out in mid 2010. However, we performed new experiments in late 2011 and in early 2012 and we comment in Section VIII on the changes Wuala has undergone since 2010.

III. Wuala Network Architecture

As outlined in [10], the Wuala infrastructure consists of **servers**, running in a data center, and client machines, which we refer to as **peers**. Such a hybrid architecture attenuates the impact of the volatility of peers, which may join and leave the system at any time.

In Wuala, users can select two distinct modes of operation for their peers: **casual peers**, which do not contribute any storage to the network, or **storage peers**, that trade local storage for increased storage space within the application[2].

As we will see in Section V, whenever a file is uploaded from a client, a complete copy is first transferred to the Wuala servers in the data center. In addition, depending on the file size, some more data may be stored on storage peers.

In order to discover the Wuala network setup, we use three distinct machines:

[2]In this paper we do not study the incentive mechanisms used by Wuala.

- Two machines act as storage peers, each offering 100 GiB of local storage to Wuala. These machines stay on-line for a period of two months.
- A third machine acting as a casual peer downloading 452 public files (20 GiB in total) over a period of four days.

These three machines are connected via a 100 Mbit/s link to the Internet. The utilization of this link is low, so our results should not be affected by access bandwidth limitations.

Based on the output of Tstat, we find that servers use an immutable port in the range 50012, 51012, ..., 59012 while peers – in general – listen on a port in the range 7100, 7101, ..., 7600, chosen at random.

In 2010, we identified 295 different IP addresses of Wuala servers: 293 IP addresses were located in four subnetworks that belong to Hetzner Online AG [11] in Germany, which is an Internet hosting service, and 2 IP addresses were located in Switzerland. Recent measurements at the end of 2011 showed that the number of IP addresses has further increased: We find 412 servers in Germany, 3 in France, and 3 in Switzerland.

While it is beyond the scope of the paper to estimate the total size of Wuala, it is still interesting to see the geographical distribution of the *peers*. We can get a glimpse by geolocating all the IP addresses our three test clients had seen in 2010: We observed 19,078 unique IP addresses of peers running Wuala. Using Maxmind [12], we obtained the following geographical distribution for the IP addresses: Germany (36%), Switzerland (16%), United States (7%), France (7%), Spain (6%), Italy (3%), and Austria (3%); the remaining 22% were spread across more than 100 countries.

In the following we try to understand how data is stored in the Wuala system.

IV. Data Management in General

We now discuss several techniques related to data management in Wuala. This includes encoding and encryption of data as well as data placement.

A. Erasure Coding

Since Wuala uses error correcting codes to produce redundant data for larger files, we first provide a brief overview on the basics of erasure coding techniques [13].

Erasure coding has been widely used in storage systems to increase data availability when storing data on unreliable machines [14] and has also extensively been studied for peer-to-peer storage systems [15]–[17]. Erasure coding takes as input a set of k fragments, also referred to as **original fragments** or originals for short, and outputs a set of $k + h$ fragments referred to as **coding fragments**. The ratio $\frac{h}{k}$ is referred to as the **degree of redundancy**. A code for which the first k coding fragments are identical to the original fragments is referred to as **systematic code**. Systematic codes have the advantage that for reconstructing the original file, decoding is only necessary if some of the original fragments are lost. RAID disk arrays are a very prominent example for a storage system that uses erasure coding. RAID arrays typically use systematic codes with $h = 1$, which means a single coding fragment is produced by doing a bit-wise XOR over the k original fragments. When each of the $k+1$ fragments is stored on a different disk, one disk failure can be tolerated without affecting the data availability.

B. Encryption

Wuala uses encryption to protect user data and meta information. As stated by the team of Wuala [10], a cryptographic tree structure called Cryptree [18] is used.

In fact, we observe that a file enqueued for upload into Wuala results in a new file of equal size in a local folder. Since the binary representation of the new file is different from the original file, we conclude that the file has been encrypted locally.

Further, Wuala uses convergent encryption [19], which means that the key for encrypting a file is derived from the file content using hashing. If different users upload the same file, the encrypted file will be byte by byte identical, which allows Wuala to perform duplicate detection and store the file only once. We confirm this behaviour by uploading two identical files from two different machines and user accounts in sequence: the second upload is skipped.

C. Data Placement

Since the encrypted copy of the file is stored in a temporary folder on our local client, it is possible to compare the content of the encrypted file with the data that are uploaded, in order to see how a file is uploaded to the network. We upload files of many different sizes into Wuala and capture all the network traffic for analysis with Tstat and Wireshark.

Using this method, we observe that Wuala treats files differently depending on their size. Also, this treatment is independent of the operational mode, casual or storage peer, of the client.

For "**tiny files**" (0 KiB - 4 KiB), the whole file is embedded into metadata information and is transferred to a single server using TCP. Instead, "**small files**" (4 KiB - 292 KiB) are replicated twice and transferred to distinct servers via TCP. "**Medium files**" (292 KiB - 1 MiB) are first erasure-coded (using fragments of 10 KiB) and then transferred to a variable number of servers using UDP. Finally, "**big files**" (exceeding 1 MiB) are encoded with the procedure described in the following section, and transmitted to *both*, servers and peers using UDP as transport protocol.

We are also able to determine the placement strategy for all metadata (e.g. file names, folder structure, and file sizes): A newly installed Wuala client does not open any connection to another peer during log-in or while browsing directories in the user interface. From this we conclude that metadata are not stored in a distributed hash table on the peers but in a centralized repository on the servers.

V. Anatomy of an Upload Operation

This section focuses on the upload of big files, which are the only ones that are uploaded to both, Wuala servers and peers. Before we present the details, we sketch the high level steps that are executed when uploading a big file:

1) Store a local copy of the file, apply convergent encryption [18], [19] and organize the content in transmission blocks (see Section V-A);
2) Contact a coordinator server T to obtain a list of servers $\mathbb{A}$ and upload transmission blocks to a subset $\mathbb{S} \subseteq \mathbb{A}$ of servers (see Section V-C);
3) The uploaded file is indicated to the user as completely uploaded;
4) Contact a coordinator server T to obtain a list of peers $\mathbb{B}$ and upload additional transmission blocks to a subset $\mathbb{P} \subseteq \mathbb{B}$ of peers (see Section V-D);
5) Update T with the list of peers $\mathbb{P} \subseteq \mathbb{B}$ that were selected to store additional transmission blocks.

We see that Wuala prioritizes server uploads over client uploads: the operation is considered successful when a sufficient number of transmission blocks is safely stored on servers in the data center. The machines in a data center generally have higher availability and more bandwidth than remote peers.

In the following, we present in more detail some of the steps performed during the upload operation.

A. Generation of Transmission Blocks

In order to infer the coding procedure used by Wuala, we compare the binary representation of a temporary encrypted big file, which is stored locally, with the data finally transferred to different machines. The results of this comparison reveal the procedure presented in the following.

Files of size S KiB are first partitioned into $f = \lceil S/100 \rceil$ **segments** of 100 KiB size. Each of these segments is then encoded independently using a systematic code. For this purpose, a segment is broken into $k = 100$ original fragments of 1 KiB that are encoded to obtain $n = k + h$ **coding fragments** of 1 KiB each, which make up one **coding block**. In this way, segment $i, i \in \{1, \ldots, f\}$ results in n coding fragments $C_{i,1}, ..., C_{i,n}$. For transmission over the network, the coding fragments $C_{1,j}, ..., C_{f,j}$ for a given $j, j \in \{1, \ldots, n\}$ are all grouped into one **transmission block** $T_j, j \in \{1, \ldots, n\}$ according to a technique known as **interleaving** [20].

Therefore, the size of the transmission block is a function of the size of the file. A file of size S KiB will consist of $f = \lceil S/100 \rceil$ segments that will be transmitted to the servers as n transmission blocks of f KiB each. Since the first 100 transmission blocks contain the unencoded content of a file, we label them **original transmission blocks**, while successive erasure coded transmission blocks are called **additional transmission blocks**. Figure 2 illustrates this procedure in detail.

As the maximum file size in Wuala is currently limited to 14 GiB, the biggest transmission block can have a size of 140 MiB.

To reconstruct the content of a segment, the decoding operation requires any 100 out of n coding fragments that make up that segment. As we will see later in Section VII, this fact is used in Wuala to greatly simplify the data transfer protocol.

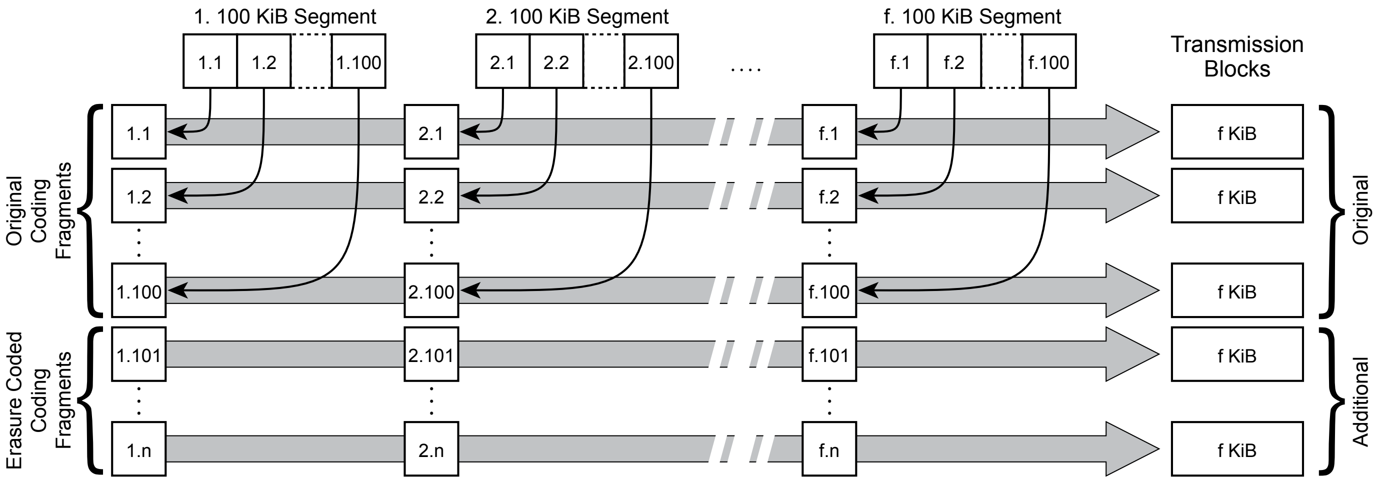


Figure 2. Generation of Transmission Blocks

B. Use of Obfuscation

Inspecting the content of the data packets exchanged between the client and the coordination server T shows that the IP addresses of potential sinks are obfuscated. However, since we know that many servers are located in the subnetwork 188.40.0.0/16, the first two bytes of the binary representation of the IP address should rarely change. This knowledge allows us to identify a simple obfuscation technique that consists in a bitwise XOR operation of the IP addresses with the sequence 0xb3b3b3b3 (as shown in Figure 3). This obfuscation scheme is evident since subsequent connections to servers match exactly the obtained IP addresses.

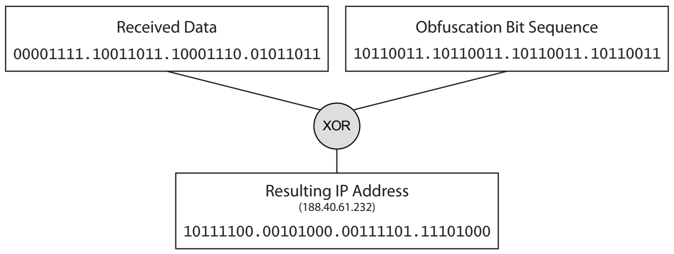


Figure 3. Obfuscating Procedure to Mask Server IP Addresses

C. Uploads to Servers

By inspecting the packet traces and repeatedly uploading the same files from different peers at the same time, we discover that the coordination server $\mathbb{T}$ uses – based on transmission block identifiers – a simple hashing scheme for the server selection. For this purpose, the uploading peer sends a batch of 20 sequence numbers to the coordinator $\mathbb{T}$, starting at 0. These sequence numbers each identify one of the transmission blocks to be uploaded. Subsequently, the server $\mathbb{T}$ replies with a list of 20 obfuscated server IP addresses. Since the server $\mathbb{T}$ deterministically chooses these servers, we assume that the transmission block identifiers are hashed together with the file hash. The resulting hash can be used for lookup in a hash table on server $\mathbb{T}$ to determine the placement of transmission blocks on servers. However, due to the limited number of servers, hashing can result in collisions, which in our case means that the coordination server will assign multiple transmission blocks to the same server. Storing more than one transmission block on the *same* server does not improve the availability as does storing each transmission block on a *different* server. Therefore, the upload of transmission blocks to servers is done in two phases, as described in the following.

Upload of Original Transmission Blocks: The inspection of datagrams transferred indicates that the upload to servers begins by uploading *original* transmission blocks in parallel: first two concurrent connections are established; if the available upload bandwidth permits, the client initiates more parallel connections until saturation.

The placement of the original transmission blocks is specified by the data center:

1) The peer sets up a reliable TCP connection to a Wuala coordination server $\mathbb{T}$ and sends 20 sequence numbers.
2) The server $\mathbb{T}$ replies with a list of 20 servers.
3) The peer uploads to each of these servers one original transmission block.
4) This step is repeated five times until all original transmission blocks are uploaded.

In this phase, the peer strictly follows the placement instructions given by the coordinator T and different original transmission blocks may be placed on same server, since the list of servers returned by $\mathbb{T}$ may contain servers that already store a transmission block belonging to that file.

Upload of Additional Transmission Blocks: Once all original transmission blocks are uploaded, the client uploads *additional* transmission blocks to the servers:

1) The peer sends 20 sequence numbers to the Wuala

coordination server $\mathbb{T}$.

2) The server $\mathbb{T}$ replies with a list of 20 candidate servers.
3) The peer uploads additional transmission blocks to those servers in the list to which the client has not already uploaded any transmission block related to the same file.
4) These steps are repeated a certain number of times.

The upload of additional transmission blocks is done to inject redundancy: not only servers may crash, but as mentioned, the same server may hold multiple original transmission blocks, which affects resiliency to failures.

Despite the many experiments we performed, we could not derive the exact number of additional transmission blocks h produced by Wuala. In our experiments h varied between 17 and 40.

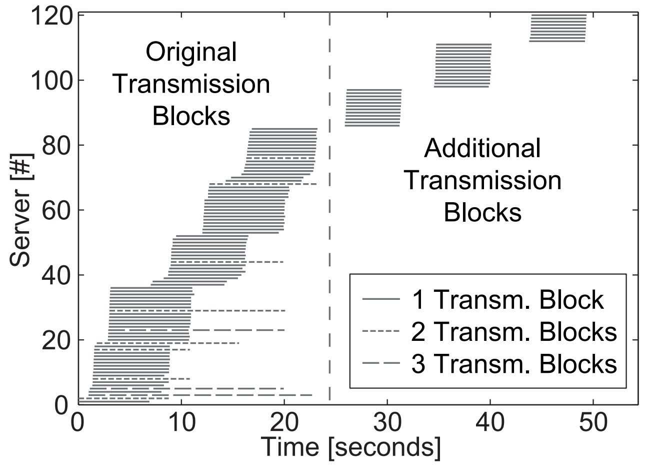


Figure 4. Upload Activity to Servers

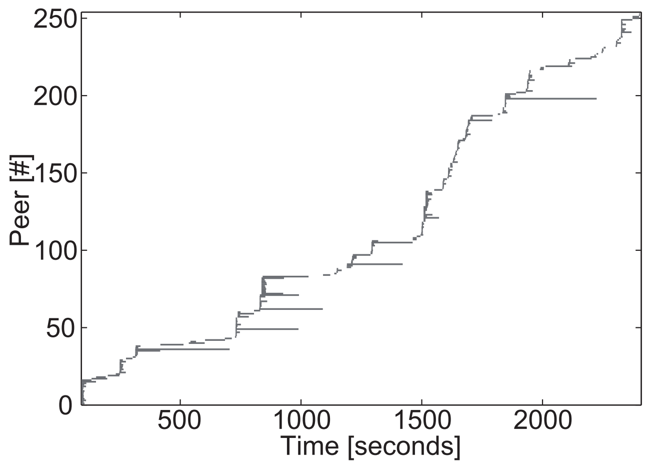


Figure 5. Upload Activity to Peers

Figure 4 illustrates the upload process to servers for a file of size 100 MiB from one of our measurement clients. On the y-axis we show the unique servers selected to store transmission blocks: a horizontal line indicates the time it takes for one or more blocks to be transferred; each transmission block is of size 1 MiB. The server upload results in 135 transmission blocks sent to 120 distinct servers. First the 100 *original* transmission blocks are uploaded, followed by 35 *additional* transmission blocks uploaded in 3 batches. In total, the upload consists of 144 MiB.

Once the server upload is completed, a file is safely stored and available, even if the Wuala client goes offline.

D. Uploads to Peers

Usually, the Wuala client will stay on-line after the file has been uploaded to the server. As our traces indicate, once a file has been transferred to the servers, the upload process continues by uploading *additional* transmission blocks to remote storage peers.

If we look at the trace of the single file upload we discussed in the previous section, we observe that once the server upload completes, another 250 remote peers are used to store additional 273 transmission blocks. Figure 5 depicts such upload activity to storage peers. It is interesting to note that only a few uploads are performed at the same time. As we will substantiate in Section VI, storage peers participating in Wuala are used as content caches. Uploading to peers therefore does not affect the availability and durability of the file.

In contrast to the server selection mechanism described above, data placement on peers is essentially *random*. Our traces indicate that a Wuala client obtains from the Wuala coordination server a random list of peers before uploading transmission blocks. We also find that remote peers can store more than one transmission block that is part of the same file, although this rarely happens.

Figure 5 shows the activity for the upload to different peers. As can be seen, the transmission rates to peers can vary a lot. We try to determine whether the selection of peers is biased. For this purpose we use our storage peer offering storage space to Wuala over a period of several months. We observe the number of incoming transmission blocks per day using the monitoring interface of Wuala. In Figure 6 we see that the number of incoming transmission blocks increases with increasing on-line time, which indicates that the coordination server has a bias in the peer selection towards storage peers with larger up-times. This approach is sensible since it enhances data availability on peers [21].

E. Maintenance

We want to understand the maintenance policy of Wuala. In fact, a pure P2P storage system requires frequent maintenance of stored files to achieve a given level of redundancy, which implies the generation and upload of additional transmission blocks. We want to understand if the peer that owns the file is involved in the maintenance, as is typically the case in a P2P storage system [17].

We carried out the following experiment, which involves one of our Wuala clients operating as a *casual* peer. From this peer we upload a 100 MiB file into Wuala. Over the next month we monitor the outgoing communication of our casual peer: While the encrypted representation of the uploaded file remains in the temporary folder of the casual peer, we cannot

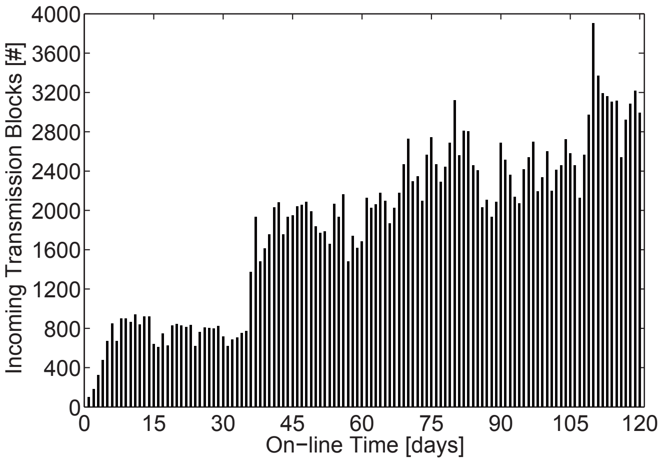


Figure 6. Incoming Transmission Blocks per Day

detect any further upload operations. In Section VI we will see that one week after the file was uploaded, its availability on peers is already fairly low. To keep it available, maintenance would already have been necessary.

While we can conclude that the peers are not involved in data maintenance, we cannot tell if there is some maintenance that is *completely internal* to the data center. On this topic we could obtain more information by communicating with one of the co-founders of Wuala. Luzius Meisser from Wuala told us [22], that Wuala tries to have transmission blocks belonging to a single file stored on at least 116 servers. As soon as a file is spread across less than 108 different servers, a maintenance operation is triggered in order to have transmission blocks on 116 different servers. Using this information we can approximate a lower bound for the availability of stored files. If we assume for Wuala servers an availability of 0.99, the file availability is at least [15]:

$$A_{file} = \sum_{i=100}^{108} \binom{108}{i} 0.99^i (1-0.99)^{108-i} \approx 0.999998 \quad (1)$$

VI. Anatomy of a Download Operation

Similar to Section V, we now study the download operation. Our traces indicate that downloading a file involves the following steps:

1) Contact a coordination server $\mathbb{T}$ for a list of servers $\mathbb{S}$ and peers $\mathbb{P}$ that hold a transmission block related to the file;
2) Start transfer of transmission blocks from servers $\mathbb{S}$;
3) Start transfer of transmission blocks from peers $\mathbb{P}$;
4) Each established transfer from a peer $p \in \mathbb{P}$ replaces one transfer from a server $s \in \mathbb{S}$;
5) The encrypted file is reconstructed in a temporary folder;
6) Optional: Contact $\mathbb{T}$ again for a list of peers $\mathbb{P}'$ and upload additional transmission blocks to peers $\mathbb{A} \subseteq \mathbb{P}'$;
7) Send $\mathbb{T}$ the list of peers $\mathbb{A}$ storing additional transmission blocks.

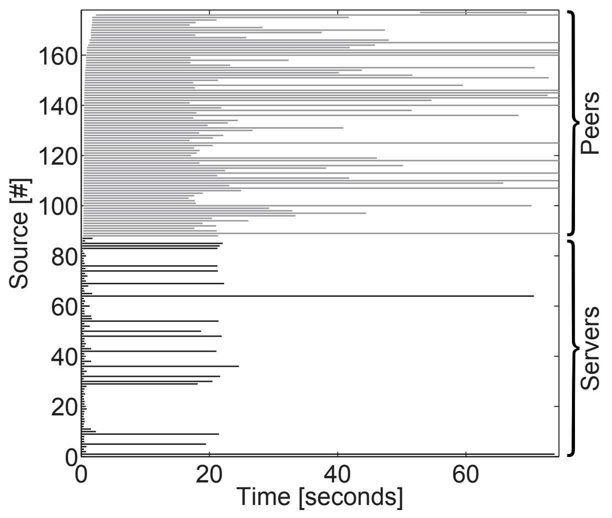
Our goal is to characterize the mechanism used to select the sources of transmission blocks (servers, peers) and to corroborate the findings of Section V-D, i.e., that peers are mostly used as content caches. This approach has already been investigated by [23]. In summary, our results indicate that data durability and data availability are achieved using servers in a data center. However, whenever possible, the burden of serving files – especially popular ones – is delegated to peers. Also, the download operation involves one more step that resembles to data maintenance: if a file is requested, its availability on peers is restored.

We now describe our experiments in detail. We instruct our Wuala client to download the same 100 MiB file that he had uploaded previously. One download is performed immediately after the upload operation was completed and another one is performed one week later.

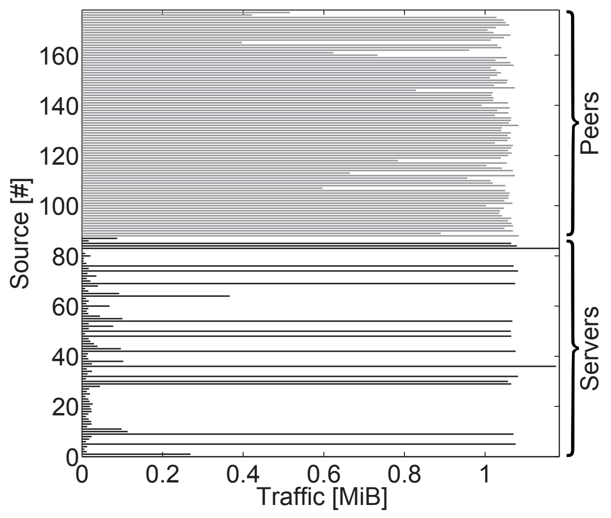
Figure 7(a) indicates, on the y-axis, the source identifier and type (server or peer): horizontal lines represent the amount of time required for attempts or full downloads of a transmission block. Similarily, Figure 7(b) shows the amount of traffic from each sink for the same transfer. Clearly, peers provide most of the fragments of the downloaded file in the immediate download case. Recall from Section V that our client uploaded 250 transmission blocks to remote peers. We also note that our client establishes many parallel connections [24], among them a substantial number to servers. However, within a short time after the start of the download, some data transfers are diverted to peers. This happens as soon as first connections to peers are successfully established. In this respect, the reliable transfer protocol described in Section VII plays an important role, as it is possible to switch sources for downloading a file at any time: missing coding fragments within a transmission block can be individually indicated to the source. We see that servers generally guarantee a fast startup of a file transfer, while subsequent transfers from peers take over the transfers as soon as possible. For this particular experiment, our measurements indicate that 87 servers provided 21 MiB worth of data, whereas 97 MiB were served by a total of 90 peers.

Figures 7(c) and 7(d) show how the situation changes when the download operation is initiated one week after the upload. In this experiment, 86 transfer sessions were opened to servers and only 26 to peers. The download resulted in 100 MiB worth of data from servers and 23 MiB from peers. Clearly, this indicates scarce data availability on peers. Barely 10% of the 250 peers that had received transmission blocks a week before contribute to the download. Since we certainly expect more than 10% of the same users still to use Wuala after one week we explain this by the fact that peer up-times are low.

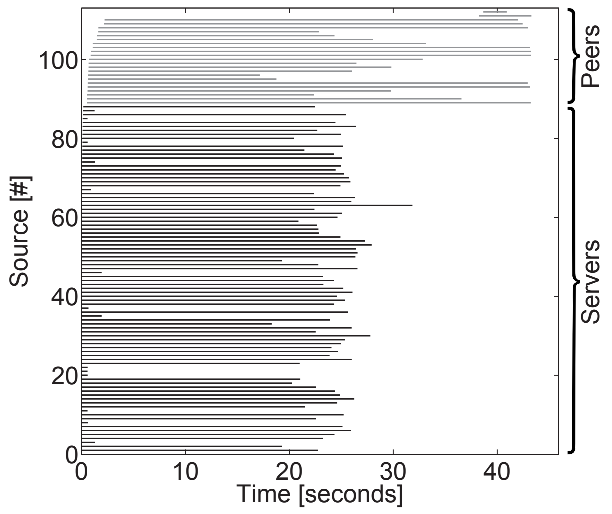
Our traces also indicate that downloading the file after one week triggers the upload of new transmission blocks from our peer to some randomly selected remote peers: A total of 112 transmission blocks have been uploaded. These additional uploads serve to *refresh the peer caches*: subsequent downloads for the same file will be served predominantly by peers. Especially for flash crowds, when a file is requested in a short timespan by a large number of users, this approach can help reduce the outgoing traffic from the servers in the data center.



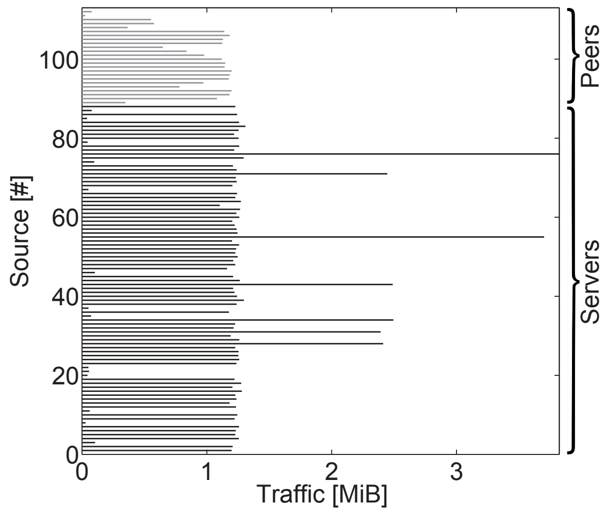
(a) Download Right after Upload: Activity per Source over Time



(b) Download Right after Upload: Data Traffic per Source



(c) Download One Week after Upload: Activity per Source over Time



(d) Download One Week after Upload: Data Traffic per Source

Figure 7. Download at different points in time

VII. Transfer of Coding Fragments

We now focus now on the transport protocol used to reliably transfer big files. Our traces indicate that Wuala uses UDP, which does not offer any loss recovery. Therefore, it is the application that must cope with lost, duplicated, or reordered datagrams. We first introduced packet loss using `iptables` to see how much loss Wuala can tolerate: For a 100 MiB file, the upload completed as long as the loss rate was not above 10% packet loss, whereas the download completed even for loss rates up to 90%.

We now explain our methodology to infer from our traces how Wuala performs error detection and recovery. First we note that the sender of a transmission block transmits a burst of datagrams with a size from 1,040 to 1,047 bytes. In fact, each of these datagrams, which we shall further call **content datagrams**, contains 1,024 bytes of payload data that exactly correspond to one *coding fragment* (see figure 2). The receiver occasionally sends small **status datagrams** of 36 bytes. The number of content datagrams sent in response to a status datagram depends on a variable window size. The window size starts with 1 and is increased until it reaches a maximum window size of 16. To analyze each datagram type more closely, we use a Chi-Square like test as proposed by Finamore et al. [25]. This test reveals how the values taken by a group of bits deviates over several datagrams from a theoretical random distribution. This information helps us to determine the boundaries of fields, identify fields that stay constant or are incremented. We use 4 bit groups in the following tests.

First, we analyze *content* datagrams by taking a set of 1000 datagrams from one transfer of a transmission block from a single source. In the header of the content datagram we can identify a sequence number increasing by one for each content datagram. For a fully transferred transmission block, this number covers exactly the number of coding fragments the transmission block consists of.

Performing the Chi-Square test on 500 datagrams of one transfer session mixed with 500 datagrams of another transfer session allows to identify the header field that contains the identifier that uniquely indicates the transfer session. Given this information, a receiver is able to assign each incoming coding fragment to the proper position in the transmission block. Since UDP datagrams are delivered atomically, a coding fragment is either delivered completely or not at all. Duplicate datagrams or delivery in wrong order are, due to the identifiers in the header, easy to detect. The protocol also needs to deal with lost datagrams. To find out how this is done we also need to analyze the content of the status datagrams sent by the receiver.

Applying the same procedure as above to the analysis of the *status* datagrams, we also find a unique identifier for each transfer session. We are able to identify a mechanism in the status datagrams to address missing coding fragments within a transmission block. An offset in combination with a bitmap of 7 bytes is used to tell the sender which coding fragments to transfer next. Since the bitmap is able to address only 56 successive coding fragments, the offset is used to shift the bitmap during the transfer in order to be able to cover all fragments that are part of the same transmission block. Whenever needed, this technique also allows the receiver to switch to another data source for missing coding fragments. Remember that thanks to the erasure coding used (see Figure 2), for a segment to be correctly received the receiver can use *any* set of 100 different coding fragments.

This procedure corresponds to a pull-based approach [26]: it not only allows to request the missing coding fragments individually, but also allows the receiver to individually adapt the rate at which new fragments arrive.

VIII. Recent Changes to Wuala

The results presented so far are based on measurements made in mid 2010. In the mean time, significant modifications have been made to the Wuala architecture [27]. In this section we first present an outline of the new system design, followed by a discussion on the reasons for this modification.

A. New Storage System

The most important change made in the release issued at the end of 2011 is that Wuala no longer stores data on peers (see Section V): peers are not used as content caches anymore, and the Wuala architecture is now a purely centralized one. In this Section, we present measurement results that elucidate the data management in general and the new storage system called **blobs**, which is still in a beta stage; we also focus on upload and download operations. In the following, we adopt the same experimental setup as introduced in Section II.

Data Management: In order to reassess the data management strategy used in the new release of Wuala, we proceed by uploading a number of files with different sizes, as we did in Section IV. Our traces show that all files in the new storage system are indeed uploaded to servers only, corroborating the fact that Wuala is now a pure server-based file sharing and backup system. Again, the size of a file determines how data is transferred to and stored on the servers. Our traces also indicate that a new file fragmentation scheme is used.

Files smaller than 4 KiB in size (c.f. tiny files, Section IV) are embedded into metadata and uploaded to a single server, as in the previous version of Wuala. Instead, **Short files** with sizes between 4 KiB and 128 KiB are now simply transferred reliably via TCP to a single server. All files exceeding 128 KiB are now broken into 128 KiB **chunks**: we label such files **composite files**. Chunks related to a composite file are further grouped into **blobs** of 4 MiB. Figure 8 illustrates an example of such a composition.

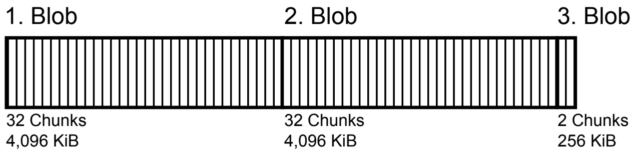


Figure 8. Example of a 8,400 KiB sized file resulting in 66 chunks grouped into 3 Blobs

Uploading Files: We now focus on file upload and describe how this operation has changed. A file is simply encrypted at the client and uploaded *once, to a single server* in the data center. Clients do not upload any additional redundant data to the servers, which means that clients are not involved in applying erasure coding to their files. The amount of traffic for uploading a short file is roughly the same as its file size. Instead, for larger composite files, data are uploaded as a multiple of 128 KiB, which is the chunk size. We also observe that composite files are uploaded using HTTP POST requests sent to an application server [28]. In fact, for each blob of a composite file a new HTTP POST request is sent by the client, resulting in up to three concurrent TCP connections to the same server.

Deduplication of Blobs: The new file fragmentation scheme of Wuala allows performing data deduplication on blob objects, as opposed to a file-level deduplication used in the previous version of Wuala.

As a consequence, data deduplication is now more fine-grained: Altering a few bytes in a composite file only triggers the upload of the blobs affected by the modification. For example, appending some bytes at the end of a composite file results in the upload of the last blob, or, if the number of blobs increased by one due to the append operation, only the last two blobs are uploaded to servers. Instead, when prepending some bytes at the beginning of a composite file, the whole file will be uploaded again.

Downloading Files: When downloading short files, the data is sent from a single server. However, when repeatedly downloading the same short file, the identity of the server is not constant: we recorded that more than 10 different machines delivered the file to our measurement client. Since we know that data is not replicated within the Wuala data center (see Section IX), we conjecture the presence of a number of proxy servers that take care of data delivery.

The download of composite files results in up to three TCP connections to different servers. The amount of data received from the three servers involved in the download operation is a multiple of the chunk size (128 KiB), from each server: the total download traffic accounts for the file size rounded up to the next multiple of the chunk size.

B. Conversation with the Designer of Wuala

It is very instructive to look into the reasons underlying the recent changes in the architecture of Wuala. One may ask why the designers of Wuala abandoned the idea of an hybrid architecture that relied on peer resources to deliver data. In discussions with one of the co-founders of Wuala [22] we were learned that

- An hybrid architecture, where peers participate in serving transmission blocks, makes the implementation and maintenance of the Wuala software much more complicated than a pure client/server approach.
- The price of outgoing bandwidth from the data center has dropped dramatically in the past few years. As of spring 2012, the price charged by the provider that Wuala uses for storing data [11] decreased by an order of magnitude as compared to the rates in 2008 (1 Euro as opposed to 12 Euros). This means taht the economic incentive to outsource data delivery to peers lost its attraction.
- As our measurement results showed (c.f. Section VI), the contribution of peers to download operations was marginal. After just one week, a file will already predominantly be downloaded from servers, instead from peers. This fact reinforces the idea that an hybrid architecture – albeit interesting from a research point of view – does not bring substantial benefits to compensate for the additional efforts due to a more complex software architecture.
- Wuala is increasingly used on smart phones, which have more stringent energy and bandwidth constraints than fixed clients. It is therefore preferable not to produce redundant data on the client in order to reduce CPU and bandwidth utilization. Instead, data redundancy is best introduced at the server side, allowing clients to upload only the amount of data they want to store, and not more.
- The target user group of Wuala has evolved from home users, mostly interested in the social aspect of content sharing inherent to a P2P approach, to business customers who's main concern is security and durability.

An additional technical detail we learned in our discussion is that once a file has been uploaded to the new Wuala system, erasure coding is still used to produce redundant coding blocks that are then stored on other servers for improved availability. However, this operation is now completely invisible to the client since it is internal to the data center.

IX. Discussion

In this paper, we have seen that Wuala has evolved from a hybrid system where peers can be used to serve contents, to a purely centralized system. In the context of Internet applications based on a P2P architecture, there is at least another prominent example that is similar to the Wuala case, namely Internet file-sharing. Without digging into the whole history of P2P file sharing applications, it is well known that BitTorrent, which represents a successful application based on a P2P paradigm, has – and continues to, albeit with country-specific nuances [29] – largely dominated Internet traffic, an indication of its popularity. However, in recent years, we have witnessed the advent of an equally popular alternative to P2P file-sharing applications, this time based on a purely centralized approach. Besides considerations on the usability of a stand-alone client versus an easy-to-use Web application, and on user privacy – which certainly contributed to the raise of one click hosters – centralized services such as Mega Upload and equivalent have flourished also because – similarly to what we discussed for Wuala – the costs associated to a client-server architecture have significantly dropped.

Which lessons can we learn, and what do the two examples discussed above imply for the research in P2P systems? We think there are two aspects that need to be clearly distinguished

- Research: In research, one is free to make its own set of assumptions under which a given problem is then addressed. It is important to realize that these assumptions do not necessarily need to incorporate the current real-world constraints. In fact, when identifying new research directions, one should not confine itself by what it is technically feasible today. Instead, one may ask how a problem could be solved if a given constraint that exists today would disappear. When such a design then becomes "practicable" at a later point of time, it may in fact be seen as major breakthrough. Therefore, the fact that P2P based designs seem to fall "out-of-fashion" does not mean that they are irrelevant. In fact, some of them start already finding their use in large scale data centers with thousands of machines, see e.g. Dynamo of Amazon [30].
- Service: When developing a service, on the other hand, one must understand well the current constraints and the environment in which the service will be used, in order to develop the most cost effective and efficient solution. In the case of Wuala, important constraints are limited upload bandwidth of peers (particularly in the case of smart phones), low peer availability, limited battery power (particularly in the case of hand-held devices) and continuously falling prices for data center storage and data center bandwidth.

X. Conclusion

We saw that there are several impediments in adopting a P2P approach: Low peer availability and peer churn, which if dealt with properly, lead to much more complex software systems as compared to a mere client/server based architecture [22]. Dealing with "unreliable" peers also results in a large amount of control traffic, as pointed out first by Blake and Rodriguez [31]. Another factor that is typically ignored is the programming effort required to build such distributed service as compared to simply using centralized components of high availability.

We saw that Wuala, a widely deployed "peer-assisted" storage system, offers a service to a large community, so ease of operation and cost efficiency are important. Therefore, in the beginning, Wuala used peers to save transmission bandwidth but not to save storage space.

Our measurement study revealed that Wualas hybrid approach overcomes most of the well known problems related to peer churn. We showed that Wuala stores all the metadata on servers and uses erasure coding in conjunction with a flexible transport protocol as the key for efficient parallel data transfers that exploit excess bandwidth capacity at the edge to serve popular content. Data availability and durability is achieved by storing data on servers hosted in data centers. To reduce the operational overhead, smaller files are managed differently from big files. Further, keeping a full copy of each file at the data center simplifies considerably the maintenance, no assumption is made about the original copy at the originating peer being available for maintenance [17].

In the last years, we have seen the advent of large scale data centers that offer services such as storage and computation at a very low cost that continues to decrease. On the other hand, mobile devices with limited processing power and bandwidth resources are increasingly used to access on-line storage systems.

Given these trends, it should be less of a surprise that a system such as Wuala

- whose design in 2007 [10] was presented as being pure P2P
- was first implemented as a hybrid system with the key components being already centralized in data center
- has been re-implemented recently as a fully centralized and server based system with the peers being relegated into the role of simple clients of the service.

Today, Wuala adopts a purely server based approach, as do alternative services such as Dropbox [3], with one important difference that in Wuala data are encrypted before upload and the password never leaves the client [10]. Dropbox encrypts data as well but admits that data is accessible to their employees [3].

For the future, our goals are to study and compare alternative designs that rely on different facets of data center economics. Questions related to the main driving factors that differentiate Wuala from Dropbox, for example, require a deeper understanding of the cost models behind these two popular services.

Acknowledgements We would like to thank Luzius Meisser from Wuala for very useful discussions and helpful comments. This work was partially supported by the FP7 project Figaro (grant n. 258378).

References

[1] Amazon Web Services, 2012, http://aws.amazon.com/s3/.

[2] Google Inc., 2012, Google Docs. http://docs.google.com/.

[3] Dropbox Inc., 2012, Dropbox. http://www.dropbox.com/.

[4] Amazon Web Services, 2012, Pricing. http://aws.amazon.com/s3/#pricing.

[5] J. Kincaid, 2011, Dropbox security bug. http://techcrunch.com/2011/06/20/dropbox-security-bug-made-passwords-optional-for-four-hours/.

[6] R. Chow and et al., "Controlling data in the cloud: outsourcing computation without outsourcing control," in *Proc. of ACM CCSW*, 2009.

[7] M. Batchelder and L. Hendren, "Obfuscating java: The most pain for the least gain," in *Compiler Construction*, ser. Lecture Notes in Computer Science, S. Krishnamurthi and M. Odersky, Eds. Springer, 2007, vol. 4420, pp. 96–110.

[8] Wireshark Foundation, 2012, Wireshark. http://www.wireshark.org/.

[9] M. Mellia, A. Carpani, and R. Lo Cigno, 2012, Tstat. http://tstat.tlc.polito.it/.

[10] D. Grolimund, 2007, Google Tech Talk. http://www.youtube.com/watch?v=3xKZ4KGkQY8.

[11] Hetzner Online AG, 2012, Internet hosting provider. http://www.hetzner.de/.

[12] Maxmind Inc, 2012, Geolocation database. http://www.maxmind.com/.

[13] I. S. Reed and X. Chen, *Error-Control Coding for Data Networks*. Kluwer Academic, 1999.

[14] D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," in *Proc. of SIGMOD*, Chicago, IL, Jun. 1988, pp. 109–116.

[15] R. Bhagwan and et al., "Total recall: system support for automated availability management," in *Proc. of USENIX NSDI*, 2004.

[16] A. Duminuco, 2009, Data redundancy and maintenance for peer-to-peer file backup systems. Ph.D. thesis, Eurecom, France.

[17] L. Toka, M. Dell'Amico, and P. Michiardi, "Online data backup: A peer-assisted approach," in *Proc. of IEEE P2P*, 2010.

[18] D. Grolimund and et al., "Cryptree: A folder tree structure for cryptographic file systems." in *Proc. of IEEE SRDS*, 2006.

[19] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proc. of ICDCS*, 2002.

[20] J. Proakis and M. Salehi, *Fundamentals of Communication Systems*. Pearson Education, 2007.

[21] J. W. Mickens and B. D. Noble, "Exploiting availability prediction in distributed systems," in *Proc. of NSDI - Volume 3*, ser. NSDI'06. Berkeley, CA, USA: USENIX Association, 2006, pp. 6–6.

[22] Personal Communication with Luzius Meisser from Wuala, Feb. 2012.

[23] N. Leibowitz, A. Bergman, R. Ben-shaul, and A. Shavit, "Are file swapping networks cacheable? characterizing p2p traffic," in *Proc. of WWW Caching Workshop*, 2002.

[24] P. Rodriguez and E. W. Biersack, "Dynamic parallel access to replicated content in the internet," *IEEE/ACM Transactions on Networking*, vol. 10, 2002.

[25] A. Finamore *et al.*, "Kiss: Stochastic packet inspection," in *Proc. of TMA*, 2009.

[26] M. Zhang, Q. Zhang, L. Sun, and S. Yang, "Understanding the power of pull-based streaming protocol: Can we do better," *IEEE JSAC*, vol. 25, 2007.

[27] LaCie AG, 2012, Wuala Changelog. http://bugs.wuala.com/changelog_page.php.

[28] Eclipse Foundation, 2012, Jetty Web Server. http://www.eclipse.org/jetty/.

[29] Sandvine, Global Internet Phenomena Report, Spring 2011 . http://www.sandvine.com/downloads/documents/05-17-2011_phenomena/Sandvine%20Global%20Internet%20Phenomena%20Report.pdf/.

[30] G. DeCandia *et al.*, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[31] C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: pick two," in *Proceedings of the 9th conference on Hot Topics in Operating Systems - Volume 9*. Berkeley, CA, USA: USENIX Association, 2003, pp. 1–1.