Technical Report
KN–2013–DiSy–01

Distributed Systems Group

Utilizing Photo Sharing Websites for Cloud Storage Backends

Sebastian Graf **Wolfgang Miller**
Marcel Waldvogel
Distributed Systems Group
Department of Computer and Information Science
University of Konstanz – Germany

Abstract. Cloud Storages combine high availability with the unencessity to maintain any own infrastructure and all-time availability. A wide field of different providers offer a flexible portfolio for any technical need and financial possibility. Yet, the possibilities of different cloud storage providers have all one issue in common: Basic storage is cheap whereas the costs increase with the storage consumed adhering the pay-as-you-go paradigm. Photo sharing websites such as Facebook, Picasa-Web, and Flickr leverage from own cloud infrastructure and offer unlimited storage for less or no charge. Obviously pictures can be used to store information in, which has been used for steganography and watermarking at low data rates. We propose a general framework for storing large amounts of data, its data density and error-correcting mechanisms tunable to the properties of the photo sharing website of your choice. Our cost-performance-analysis shows that photo sharing websites compare favorably to professional cloud storage services such as Amazon S3. Thanks to the integration of our software as a backend to the widely-used *jClouds* framework, everyone can now use photo sharing websites as one component for low-cost purposes, including archival.

Table of Contents

Abstract. . . . . a
1 Introduction 1
2 Related Work 4
3 Storing Bytes in Images 5
3.1 Single-Layered Encoders 5
3.2 Multi-Layered Encoders 7
4 Hosting of data on Photo Sharing Websites 8
4.1 Picasa-Web 8
4.2 Flickr 9
4.3 Facebook 9
5 Robustness Measures 12
5.1 Composition of Modules 12
6 Results 15
6.1 Picasa 16
6.2 Flickr 16
6.3 Facebook 17
6.4 Comparison of the Performance of the Photo Sharing Websites . . 21
7 Conclusion 23
References 24
List of Figures 25
List of Tables 26

1 Introduction

Cloud storage has been the favorite storage for all kinds of users, ranging from end-user centric file storage like Dropbox, Skydrive, or Google drive to large-scale block and NoSQL storage for the professionals by the likes of Amazon and Google through dedicated HTTP-based APIs such as SOAP or REST. Cloud service providers typically mention reliability, availability, functionality, cost efficiency, and ease of use as the main selling points for their offers.

When the free service model is not enough, however, end users are presented with a fragmented market, inflexible pricing, and compatibility issues. Projects such as *jClouds* [1] are addressing compatibility issues by providing a uniform interface. This also helps reducing friction in the market. However, the market still remains small, with complex pricing models, where a small change in usage pattern can make a big difference in price.

Our goal is to open up the market and let users chose from a wider range of established providers, which is typically forgotten: Sharing sites. Photo sharing websites such as Yahoo's Flickr, Google's Picasa-Web, or Facebook, the site with the largest number of pictures[2], are typically forgotten. However, they provide large storage space and high availability for free or cheap.

Comparing the costs for professional cloud storages like Amazon S3 (denoted as AWS S3 in the rest of the paper), end-user storage systems and photo sharing websites, the price scales in different ways: Fig. 1 compares the costs of common cloud storages with the costs of photo sharing websites based on an exponentially increasing amount of storage including transfer of the data.

Professional cloud storage systems like AWS S3 bill not only the consumed space. All data to be transferred plus the related requests cost additionally to the storage. The price for AWS S3 in Fig. 1 bases on the assumption that the space consumed is also transferred once per month resulting in a linear scaling related to the amount of data stored. User-centric cloud storages like Dropbox, Microsoft Skydrive as well as Google Drive, are accessed by native clients by default. Nevertheless, all of these storage systems offer additional REST-based services.

The pricing normally includes low complimentary storage between 2GB (for Dropbox) and 7GB (for Skydrive). All further storage must be purchased whereas the billing is defined on thresholds in opposite to the exact billing within professional cloud storages. In Fig. 1, Dropbox bills starting 3GB whereas Skydrive offers 7GB complimentary storage. Google combines within its storage-plans its products resulting in one curve for Google Drive as well as Picasa-Web. While the costs for traditional cloud storages increase with the size of the data, photo sharing websites store data either at no charge or based on flat-rates. Picasa, Flickr and Facebook, as three examples, offer unlimited storage at no costs even though restrictions related to the traffic and the resolution of the images may apply. For example, the storage on Picasa-Web is complimentary as long as the images do not exceed a maximal resolution of 800^2 (respectively 2048^2 if the user subscribes to Google+ as well). Flickr, as another example, limits the transfer of any images to 300MB per month within its complimentary offer. The Flickr-Pro account offers unlimited transfer and access to the original upload images.

All denoted photo sharing websites utilize similar infrastructures like cloud storage providers and guarantee all necessary constraints of a professional storage system as well: Availability, accessibility as well as consistence of the data while

performing heavy load operations. For example, Facebook in 2010 stored 260 billion photos representing 20 PB of data where 60 TB of new photos were uploaded within one week [2].

We develop an adapter for utilizing the vast and cheap resources of these photo sharing websites as storage backends for no-sql interfaces. Based on an in-depth analysis of Flickr, Facebook and Picasa-Web, we present the features of these platforms including billing models and image-/storage-constraints. The different appliances of compressions, as well as the access to original data, result in different transformations of any byte-content into images. The transformations represent different trade-offs between robustness, storage and bandwidth consumption as well as image size. Additional error-correction codes applied to the data stored in the images further allows us to increase the data rate per image by adding only a constant overhead. The resulting adapter is included in the *jClouds*-API [1][1] and offers convenient, cheap, and scalable access analogical to AWS S3.

The benefit accessing unlimited storage comes at a price: Compared to common cloud storages like AWS S3, our approach generates a constant overhead, heavily depending on the respective hoster, as well as on the encoder utilized with respect to the amount of data. This upload consumes more time, based on the processing of the image on the photo sharing website before acknowledging the arrival of the data since the data is compressed and organized directly within the storage-process. Our approach is therefore especially usable for use-cases demanding for high data capacity without the necessity to regular update already existing data like e.g. archiving purposes.

Whereas most current approaches related to data encapsulation in images either focus on data retrievable over cameras (like QR-Codes) or hide information in existing photos (representing the field of steganography), we generate images directly out of the data processing the images directly on the pixels themselves. As a consequence, our images contain multiple times the information achievable within QR-Codes or steganography, and leverage from the free hosting service of photo sharing websites. Based on the direct processing of the images, we are able to generate images containing data-rates ranging from $8\frac{Pixel}{Byte}$ to $\frac{1}{3}\frac{Pixel}{Byte}$.

[1] Freely available under `https://github.com/disy/jclouds` as provider “imagestore”

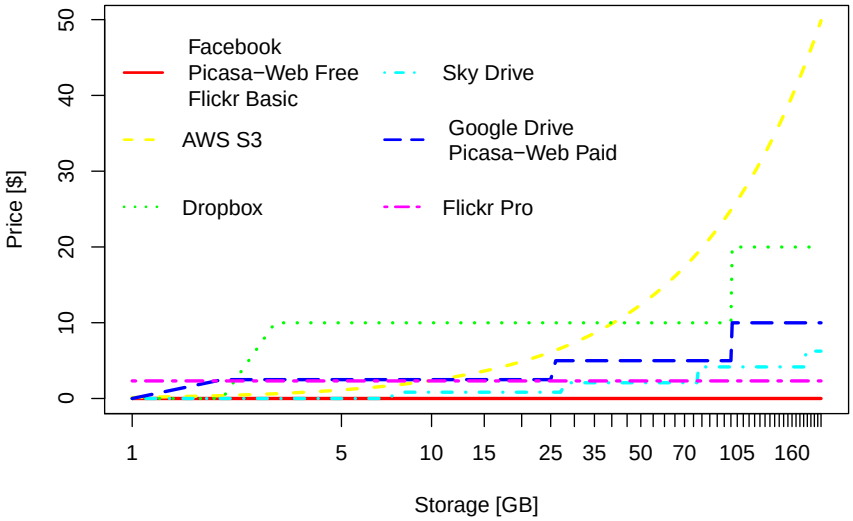


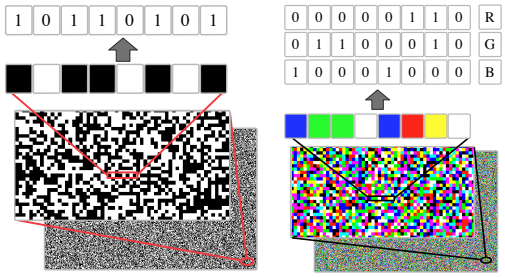
Fig. 1: Costs for Storage

2 Related Work

Utilizing images as containers for data represents one major field related to encryption. Xu et al.[3] describes a method of image-steganography that is extremely robust to JPEG compression. With this method, it is possible to extract the embedded information with zero bit-errors, even if the containing image was processed with maximum JPEG-compression. This aim is similar to ours even though we do not need to encapsulate the data in existing images. Instead, we generate the picture out of the data, enabling us to generate much higher data rates not achievable when utilizing existing images as the base for data encapsulation.

Besides steganography, the use-case of QR-Codes is quite similar to our approach. QR-Codes are not only widely distributed on any print-media, they still represent an active area of research. Langlotz et al.[4] for example introduces 4D-barcodes. The main idea behind this approach is to improve the capacity of regular cellphone-readable 2D-barcodes by adding the dimensions color and time representing a GIF, which can be processed by normal mobile phones. Even though the main purpose is similar to ours, we do not need the detour over any camera. Leveraging from the different levels within the RGB color-space, Dean and Dunn[5] propose a layered barcode where each layer contains an own barcode. One of our encoders utilizes the same technique for increasing the data-rate in images as described in Sec. 3. Kato[6] proposes a color selection schema providing robustness against color compression. We apply his findings to our encoding-schema as well.

Since we parse the image directly, the image does not need to be readable by any optical device resulting in much higher data rates. Some similar approaches for encapsulating data in images already exist mainly to leverage from the easy combination of image-compression and transmission-rates: PNGStore[7] as one example encapsulates CSS and JavaScript in PNGs to increase speed for web sites for very slow connections. This approach is very similar to our encoding-schemas even though we do not aim to leverage from any lossless compression at the moment. Nevertheless, we utilize similar mechanisms to bring any data to photo sharing websites.



(a) *SL* with $\gamma = 2$ (b) *ML* with $\gamma = 2$

Fig. 2: Examples of generated Images

3 Storing Bytes in Images

Despite most current approaches which are encapsulating data in camera-readable images or storing encrypted information in existing images, we generate images directly out of the data. Our approach generating images is thereby based directly on the pixels used as atomic units to store a variable number of colors. The number of colors per pixel is denoted as γ in the rest of the paper. The appliance of different colors to one pixel takes place in two different ways:

1. We interpret all colors as one single data-range and map different areas of this range on a variable numbers of bits. This approach is denoted as *Single Layered*-approach (*SL*-approach) in the rest of the paper.
2. We interpret each component of the RGB color-space as one single value range and map the resulting three values to each other. This approach is denoted as *Multi Layered*-approach (*ML*-approach) in the rest of the paper.

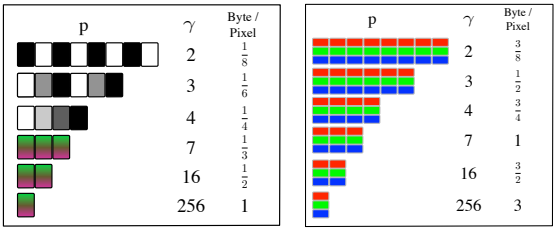
Examples for $\gamma = 2$ for both approaches, also denoted as encoders within the rest of the paper, are shown in Fig. 2 and described in the following section in more detail.

3.1 Single-Layered Encoders

Within the *SL*-approach, we interpret all colors as one single value-range. In the simplest case, this results in images consisting of black and white pixels only, as represented by Fig. 2a. In this case, we need eight pixels to store one byte denoted by the red area in Fig. 2a. The interpretation of the binary values represented by the pixels results in bits which are combined to one byte. Based on the following formula where γ is the number of values applied to one pixel, we are able to compute the number of pixels necessary to store one byte within the *SL*-approach:

$$\lceil \log_{\gamma}(256) \rceil = p \tag{1}$$

Taking this equation into account, we define different values for γ and compute the resulting number of pixels needed for an increasing number of γ. The



(a) *SL* Encodings (b) *ML* Encodings

Fig. 3: Encodings for Putting Bytes in Images

result is rounded up to the next natural number since one pixel is the finest-granular unit for painting. Fig. 3a shows the resulting encodings.

The chosen values for γ result in the best usage of the value-range for a given number of pixels. Any other choice of γ would result in the same number of necessary pixels with more information per single pixel. Since this information can not be utilized when storing bytes in the images, the additional value-range could not be used but would result in an higher fragility of the encoders since more colors are applied. As a result, the proposed 6 different values for γ are optimal with respect to the mapping of bytes to pixels. A switch of the value-range from bytes to any other base, mapped to the applicable pixels, would have the possibility to make use of these unused bits. Such an adaption is straightforward and out of focus in this paper, since we rely only on the storage of bytes.

For $\gamma = 2$, the *SL*-encoder works with only black and white as possible values per pixel resulting in an high robustness against JPEG-transformations. For $\gamma = 3$, the encoder contains one and for $\gamma = 4$, the encoder contains two further grey values.

Starting values of $\gamma = 7$, we apply colors to the pixels, derived from Kato et al.[6] as shown in Fig. 3a. Kato describes a robust choice of colors by ensuring maximal distances of the colors in the RGB color-space as well as in the YCbCr color-space. We take 7 out of the 10 defined colors for $\gamma = 7$ and utilize the approach to generate another 9 colors for $\gamma = 16$ and another 246 values for $\gamma = 256$.

By choosing $\gamma = 7$ as well as $\gamma = 3$, we are not able to make entire use from the value-range stored in the pixels but getting the highest distances between γ values per pixel. Within an increasing γ, the robustness of the images decreases as we describe in Sec. 4. As a consequence, $\gamma = 2$ represents, based on the largest distance between the applicable values per pixel, the most robust encoder whereas $\gamma = 256$ is vulnerable against all kinds of lossy compressions.

The price for this robustness is the size of the generated images: The number of bytes to be stored must be multiplied with p to get the number of necessary pixels which influence not only the generation of the image, but also the performance related to uploading and download any data including the processing from the photo sharing website before acknowledging the arrival of the picture.

3.2 Multi-Layered Encoders

To reduce p as far as possible, we extended our *SL*-approach by exposing the colors stored in the picture. By making use of the RGB color-space as three independent dimensions, we store up to 3 times more data per pixel than within the *SL*-approach. One example utilizing only two values per component is shown in Fig. 2b where each pixel is seen as composite holding up to 2^3 different values. The following equation applies to all values of γ within the *ML*-approach:

$$\frac{\lceil \log_\gamma(256) \rceil}{3} = p \tag{2}$$

Since each component represent the same value range from $[0 \dots 255]$, the same findings for the *SL*-approach apply for each component with respect to γ: As a result, only values for γ proposed in Fig. 3b generate images with the lowest applicable values per component per pixel.

Since we utilize all components independently from each other, the appliance of a robust color choice like the approach from Kato et al.[6] is obsolete. The *ML*-encoder is as a consequence vulnerable against any kind of lossy color compressions. On the other hand, the data-rate of the generated images is three times higher compared to the *SL*-encoder. The higher data-rate results in less pixels consumed. This lower number of pixels utilized, result in a lower creation time of the image, a faster up- and download of the data to the photo sharing website and a faster processing of the image before the data is acknowledged within the upload.

As a summary, the choice of the suitable encoder and the corresponding values of γ bases on the following aspects:

- The higher the supported resolution of the gallery provider is, the more data fits in the picture. We thereby aim to store images with the highest resolution possible not generating any size-based compression on the image. Resizing-operations applied by the photo sharing website harms our pixel-based encoding whereas the awareness of the highest retrievable resolution is mandatory for our approach.
- The *ML*-encoder is preferred against the *SL*-encoder since an higher data-rate results in smaller images and therefore in less consumption of upload- and downloading-resources. The appliance of the *ML*-encoder relies on the color-compression performed on the photo sharing website and can be hardened with the help of error-correction codes like proposed in Sec. 5.
- γ should be chosen as high as possible. Based on the compression applied by the photo sharing website, γ directly influences the size of the generated image represented by the $\frac{Byte}{Pixel}$-column in Fig. 3a and Fig. 3b. For performance reasons, the data-rate should be as high as possible, resulting in less resources consumed while uploading and downloading any data.

4 Hosting of data on Photo Sharing Websites

Yahoo (representing Flickr), Google (representing Picasa-Web) and Facebook are global players of photo sharing websites. All three provide free-of-charge and convenient ways to share photos. Within our approach, we extend the *jClouds*-API[1] to encapsulate bytes in images based on the encoders described in Sec. 3. The underlaying blob-model of *jClouds* is thereby mapped to images whereas containers are represented by albums or galleries. The convenient access to these photo sharing websites is provided by a REST-based API and described in more detail in Sec. 5.1.

Since the encoders represent a trade-off between robustness and size, the choice of the suitable encoder for each photo sharing website must be based on the attributes of the photo sharing website:

1. If the resolution of the hosting image provider does not match the image-resolution, the image is resized. The maximal resolution supported by the photo sharing website is mandatory. We define a fixed width based on the supported resolution und encode any upcoming bytes from top to bottom in the image. If the number of bytes to be encoded exceeds the resolution with respect to the height of the image, the bytes are split into multiple chunks resulting in multiple images to satisfy the maximal resolution of the photo sharing website.
2. The colors are transformed into the YCbCr color-space and transformed back to the RGB color-space cutting of some colors on fixed defined thresholds. Since the parameters of this transformation are applied by the providers individually, the choice of the applicable encoder depends directly on the hosting provider.

In the following section, we analyze Picasa, Flickr and Facebook based on these two attributes as well on their billing models to identify which approach is applicable as well as to define matching values for γ.

4.1 Picasa-Web

Hosting a photo on Googles infrastructure takes place over Picasa-Web. Tightly integrated into Google+ as social sharing mechanism and accessible with the help of provided APIs, Picasa-Web offers as only tested provider free and full access to original uploaded data. Based on our goal to utilize the encoder with the highest data-rate possible for performance reasons, the access to original data enables us to store our data with the *ML*-encoder and $\gamma = 256$ on Picasa-Web.

The storage is free for images with a maximal resolution of 2048^2 pixel per image if the user signed up for Google+ and 800^2 pixel otherwise. Since we aim to expose the storage as free storage, we assume a Google+ user and generate images with a maximal size of 2048^2.

In the case of larger images necessary, the limit for free storage is 1GB whereas additional storage is purchasable. The price for additional storage is 0.10$ per GB up to 25GB and 0.05$ per GB starting 100GB. Transfer and requests through the API are included in all cases. For comparison reasons, the traditional Google Cloud Storage costs 0.12$ per GB up to 1TB and includes neither requests nor traffic.[2]

[2] All prices apply to the 26^{th} of November 2012.

4.2 Flickr

Originating from the purpose of a professional photo sharing website, Flickr offers hosting for images as free and as paid service. The free service includes 300MB of traffic and the access to the images as JPEGs within the resolution of 2048^2 pixels. The paid service includes unlimited traffic, images within all resolutions restricted only by 50MB of file size and the ability to access the original uploaded data. The costs for the paid service vary between 1.87$ and 2.31$ per month.

Based on our motivation to utilize free storages only, we rely on images with a maximal resolution of 2048^2 and JPEG as retrievable format.

Fig. 4 shows the failure rates for those of our encoders generating errors by retrieving the information from the images hosted in Flickr.

The input for the images is an exponential increasing amount of random generated data with the size of $2^x \mid x \in \mathbb{N}, 10 \leq x \leq 20$. The resulting dataset is the base for all benchmarks within this paper and containing random bytes where the size of the dataset ranges from 1KiB to 1MiB in steps of powers of 2: $[1024 \dots 1048576]$ bytes.

The data was encoded by our encoders with the defined values for γ from Fig. 3, uploaded, downloaded and compared.

The *SL*-encoder fails for $\gamma = 256$ and the *ML*-encoder fails for $\gamma = 7$, $\gamma = 16$ and $\gamma = 256$ as represented by Fig. 4. The relative failure rate for $\gamma = 256$ applied to the *SL*-encoder and the *ML*-encoder (73.14% receptively 86.81%) make both encoders with such a γ unusable in combination with Flickr as free hosting instance. The *ML*-encoder with $\gamma = 7$ and $\gamma = 16$ generate only small failures respectively $9.537 * 10^{-5}\%$ and 3.14%. These small errors can be compensated by the appliance of error-correction codes. We equip our approach with a basic Reed-Solomon-Code[8] to compensate such small errors. The appliance of the error-correction code to our approach is discussed in Sec. 5.

4.3 Facebook

Facebook is nowadays the largest photo sharing website in the world[2]. Entirely free, Facebook offers unlimited storage for photos for all registered users including commenting and sharing functionalities. The aim of Facebook thereby is not the hosting of original images but the social interaction on base of the hosted images.

Facebook supports a resolution of 2048^2 pixels at most with unlimited storage and traffic whereas images must have a minimal height of 5 pixels.

Unfortunately, the color compression makes it impossible to utilize Facebook as storage backend with colored images as represented by the failure rate shown in Fig. 5. The lowest failure rate generated, is produced by the *ML*-encoder with $\gamma = 2$ (66%) making any choice of γ resulting in colored images inapplicable on Facebook even if the data would be guarded by our error-correction extension. The *SL*-encoder combined with $\gamma = 2$, $\gamma = 3$ and $\gamma = 4$ generates no errors since it relies on shades on grey only, making it applicable on Facebook.

Tab.1 shows a summary of the appliance of our proposed encoders to the evaluated photo sharing websites. The check-marks denote the applicability of an encoder with a defined γ.

Table 1: Applicable Painters on Photo Sharing Sites

Encoder		Facebook	Picasa-Web	Flickr
SL	$\gamma = 2$	✓	✓	✓
	$\gamma = 3$	✓	✓	✓
	$\gamma = 4$	✓	✓	✓
	$\gamma = 7$	⊠	✓	✓
	$\gamma = 16$	⊠	✓	✓
	$\gamma = 256$	⊠	✓	⊠
ML	$\gamma = 2$	⊠	✓	✓
	$\gamma = 3$	⊠	✓	✓
	$\gamma = 4$	⊠	✓	✓
	$\gamma = 7$	⊠	✓	(✓)
	$\gamma = 16$	⊠	✓	(✓)
	$\gamma = 256$	⊠	✓	⊠

Facebook offers least possibilities for applicable values for γ based on their restrictive color model. Only the SL-encoder with $\gamma = 2$, $\gamma = 3$ and $\gamma = 4$ are applicable on Facebook.

Picasa-Web enables users to access even original uploaded data making the ML-encoder with $\gamma = 256$ applicable.

Even though Flickr hosts the images in their original format as well, the access to this data is restricted as paid-service only. Since we rely on free services only, we are able to encode data without any error-correction extension only on the base of the SL-encoder and $\gamma = 16$. If we utilize error-correction-mechanisms like proposed by Sec. 5, we are able to use the ML-encoder up to $\gamma = 16$ on Flickr.

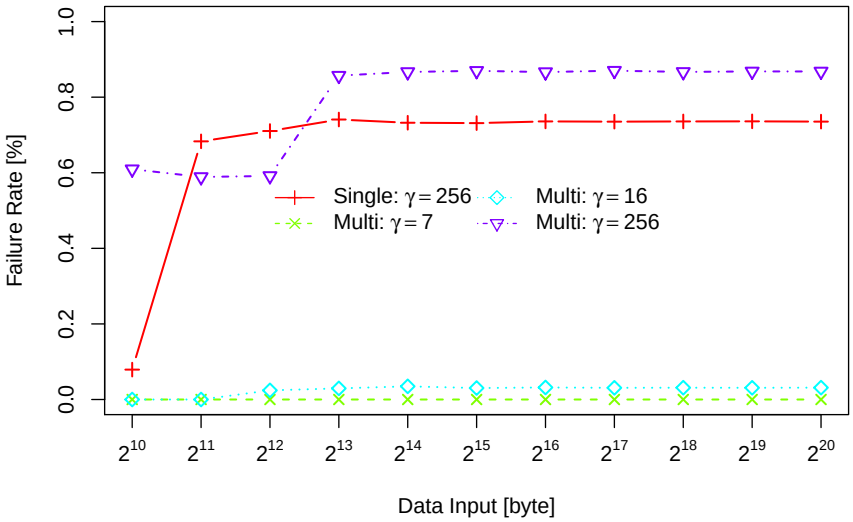


Fig. 4: Failure Rate on Flickr

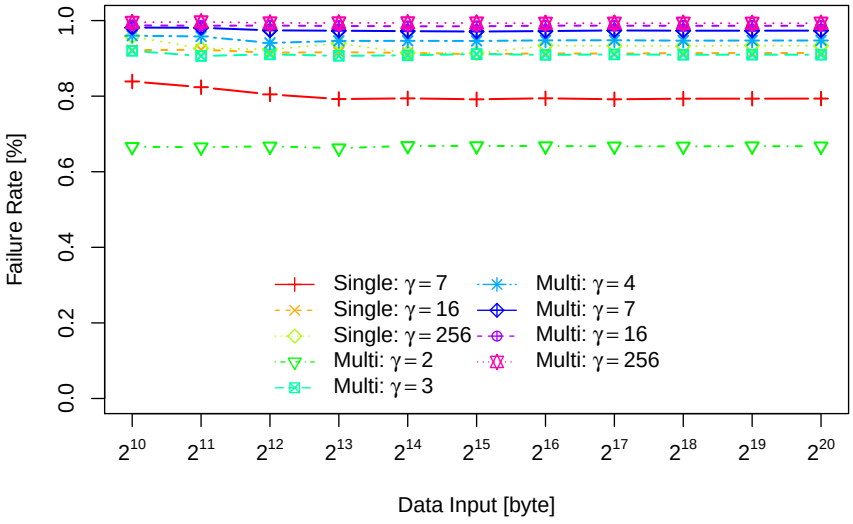


Fig. 5: Failure Rate on Facebook

5 Robustness Measures

The interfaces of photo sharing websites are not designed to handle requests as flexible as interfaces from cloud storages: First, the upload and download performance always includes some processing time on the photo sharing website. Second, put and removal operations on albums occur not as frequently as container-modifying operations on cloud storage providers. As a consequence, the client must guarantee the stability of the data transfer. We therefore implemented a multi-try approach falling back on the last request in the case of an unsuccessful upload or download of the data. This approach further harms the performance as we will see in Sec. 6 but is necessary to ensure consistency of the hosted data.

To guard the integrity of the data on the photo sharing website against any upcoming JPEG-compressions, we apply optionally the Reed-Solomon-algorithm[8] on the data before uploaded. Related to the failure rates on Flickr and Facebook shown in Fig. 4 and Fig. 5, we choose to add 10% more data for compensating at most 5% failures. The appliance of this error-correction code makes the usage of the *ML*-encoder with $\gamma = 7$ and $\gamma = 16$ usable on Flickr whereas the other failure rates of over 50% on Flickr and Facebook can not be compensated with the help of Reed-Solomon codes.

Besides this optional appliance of error-correction codes to the data, we store the meta-data of the encapsulated bytes with the help of the *SL*-encoding and $\gamma = 2$ only. The successful retrieval of this meta-data is mandatory to handle the downloaded in an appropriate way. Fig. 6 shows a schema of an image generated by the *ML*-encoder with $\gamma = 2$. The meta-data is encoded in the first 42 pixels of the image. The size of the image is stored in the first 32 pixels resulting in 4 bytes. Since the image is constructed from top to bottom based on a defined width of the image, the length of the encapsulated data can not be defined when an image is retrieved due to the fact that only entire lines of pixels are generated. The next 8 pixel determine the value for γ utilized to encode the image whereas the concrete encoder is stored in the next pixel. The last pixel of the meta-data stores the flag if the error-correction was applied while generating the image.

The blue dotted area represents the actual data. The error-correction code is represented by the appended 8 pixels surrounded by the green dotted area. Since we always paint the images from top to bottom based on a fixed length, we often have an unused area at the lower, right corner of the image in this case denoted by the yellow dotted area. The first 42 pixels within our encoded images are always reserved in the described way whereas the number of the pixels used for the data and for the error-correction-appendix may vary.

5.1 Composition of Modules

The proposed robustness measures play together in different components as presented in Fig. 7.

Within the upload of any data, represented by Fig. 7a, the data origins from any front-end utilizing the *jClouds*-API denoted by the blue area. The described error-correction approach is optionally applied on the received blob namely the inlying bytes. We split the resulting bytes into multiple junks if the generated image would not adhere to the maximum resolution of the photo sharing website. The chunks are afterwards encoded into images utilizing an

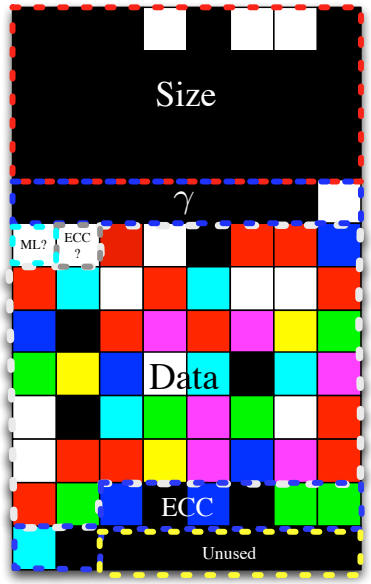
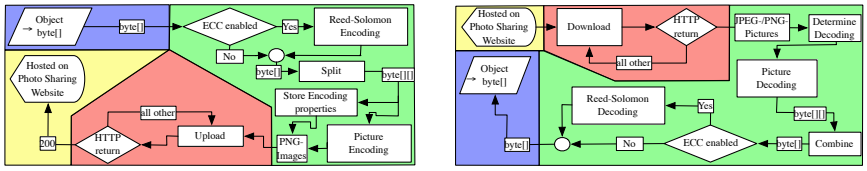


Fig. 6: Areas of Image



(a) Upload Workflow (b) Download Workflow

Fig. 7: Upload and Download Workflows

encoder and a suitable value for γ whereas the value of γ, the size of the image, the flag if error-correction is applied and the flag, what encoder was utilized, is encoded at the beginning of the generated image. All of these components are represented by the green area within Fig. 7. The resulting images are afterwards transferred to the photo sharing website- denoted by the yellow area - with the help of specific APIs translating the REST-dialect of the different photo sharing websites into Java-Method calls. These APIs are represented as the red areas in Fig. 7a and Fig. 7b. Our module is extensible enough to utilize any photo sharing website as long as the upload and download can occur automatically over any kind of open API.

The workflow of the download basically works the other way around: The file is downloaded including possible retries by specific APIs again denoted as red areas within Fig. 7b. After awareness of the encoder utilized and the value of γ, all retrieved from the beginning of the retrieved image, the image is decoded. The resulting byte chunks are combined and, if applicable, decoded by our optional

error-correction approach represented by the green area in Fig. 7b. The result is afterwards returned as blob to the front-end of our *jClouds*-utilizing program denoted again by the blue area.

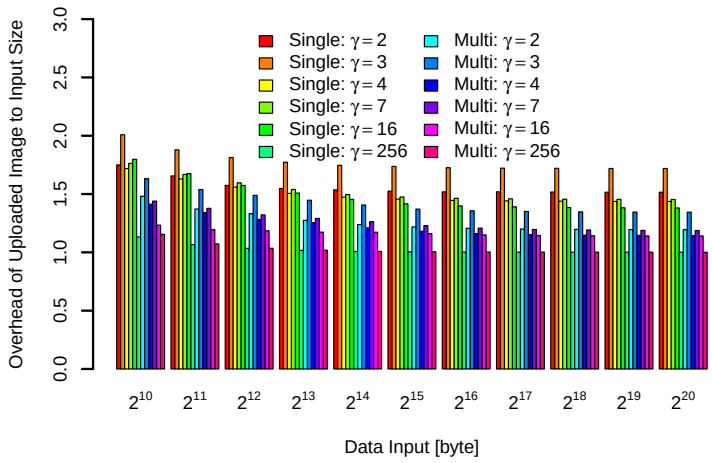


Fig. 8: Size of Image Files

6 Results

The encoding of the bytes in pixels is straight-forward and scales linear to the input data. Important for the performance is the size of the resulting image as well as the complexity related to any processing step on the photo sharing website.

Fig. 8 shows the file sizes for the defined test-data mapped on the *SL*- and *ML*-encoder as well as on different values of γ.

The y-axis denotes the relative overhead of the file size of the generated images related to the input size represented by the x-axis. The file sizes of the images generated by our encoders scale with the input size of the data. The *ML*-approach with all values of γ scales better than the *SL*-approach except for $\gamma = 256$. Writing 1024 and 2048 bytes, $\gamma = 256$ performs within the *SL*-approach better than within the *ML*-approach. The overhead of $\gamma = 3$ and $\gamma = 7$ against $\gamma = 2$, $\gamma = 4$, $\gamma = 16$ and $\gamma = 256$ is originated from the overhead of the applied value range based on the choice of γ: Based on the base 3 and 7, more values are applied per pixel than actually needed, resulting in this overhead against the encoders to the bases of 2. All encoders stabilize their relative overhead against the input data with an increasing amount of data.

The benchmarks for the photo sharing websites focus on two aspects:

1. The performance of uploading and downloading data to/from each photo sharing website is evaluated:
   - The test-data is generated randomly and consists of $2^{10} \dots 2^{20}$ bytes.
   - The plotted curves base on the mean of 50 download-/ and upload requests.
2. The size of the consumed storage on the photo sharing website bases on the data downloaded including all applied JPEG-transformations.

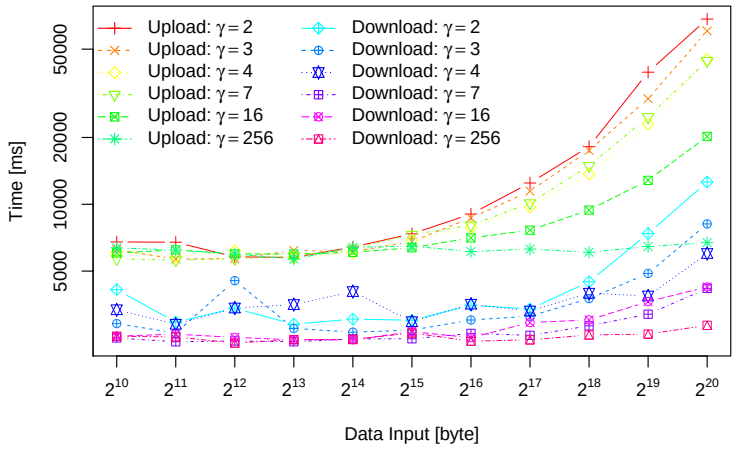


Fig. 9: Picasa Performance, *SL*-approach

6.1 Picasa

Picasa offers as only evaluated photo sharing website direct access to the original uploaded PNG enabling the *ML*-encoder even with $\gamma = 256$ as described in Sec. 4.1. The access to the original files makes the appliance of error-correction codes unnecessary.

Fig. 9 shows the performance of the *SL*-encoder on Picasa. Besides minor disturbances related to the processing of the images on Picasa while requesting, the performance depends on γ: The file size of the image has direct impact to the processing of the image and therefore to the performance. This applies to download-requests as well as to upload-requests. As a consequence, the *ML*-approach scales better due to the lower file size as shown in Fig. 10 whereas $\gamma = 256$ performs best.

The file size of the stored data is the same as the one of the uploaded data referenced in Fig. 8 based on the access to the original uploaded PNGs.

6.2 Flickr

Flickr offers free storage of all original data even though the access to this data is available as paid-service only. Since we rely on Flickr as free service only, only access to JPEG-transformed images is provided. As a consequence, images generated by the *SL*-encoder are storable on Flickr for all values for γ except $\gamma = 256$ based on our findings in Sec. 4.2. Fig. 11 shows the performance of uploading and downloading the test-data on Flickr for the *SL*-encoder. Again, the size of the generated images directly influence the performance of downloading and uploading any data especially related to the upload. This assumption is seconded by investigating the performance of the *ML*-encoder represented by Fig. 12.

Since Flickr generates errors on images encoded with the *ML*-encoder combined with $\gamma = 7$ and $\gamma = 16$, this combination is only usable when combined

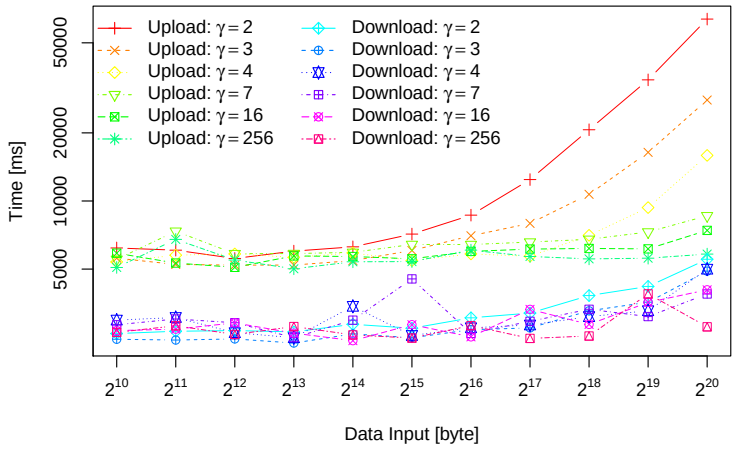


Fig. 10: Picasa Performance, *ML*-approach

with error-correction-measures like described in Sec. 5. The overhead for computing the additional data based on the Reed-Solomon Code is negligible related to the upload/download performance: The time consumed for uploading the test-data with the help of the *ML*-encoder and $\gamma = 4$ scales similar, independent if the error-correction is applied or not. As a consequence, Flickr is able to handle any data encoded with the *ML*-encoder and $\gamma = 16$ if equipped with the described error-correction-measures.

The size of the resulting images on Flickr is important since Flickr restricts the traffic to 300MB per month. The size of the test-data encoded by our applicable encoders is represented by Fig. 13. While the *SL*-encoder performs worst with $\gamma = 4$, the corresponding *ML*-approach with $\gamma = 4$ encodes the data in the lowest file size. The sizes of the images corresponds with the performance of the download of the images in Fig. 12 where all painters perform similar.

Related to the traffic restriction, the *ML*-encoder with $\gamma = 4$ seems to be the encoder of choice based on the best overhead of the image stored on Flickr, even if the upload-performance is not scaling as good as with $\gamma = 7$ and $\gamma = 16$.

6.3 Facebook

Facebook compresses the picture, like denoted in Sec. 4.3, resulting in the applicability of only color-less encoders namely the *SL*-encoder with $\gamma = 2$, $\gamma = 3$ and $\gamma = 4$.

Fig. 14 shows the performance of uploading and downloading our test-data with the help of the *SL*-encoder. Besides minor disturbances, resulting from the handling of the requests on Facebook, all approaches scale with the size of the data as expected. The size of the resulting images seems to appeal the performance since the *SL* with $\gamma = 4$ performs best related to upload and download.

Fig. 15 represents the overhead of the stored image against the encapsulated data. The ratio scales with an increasing amount of data and depends on the

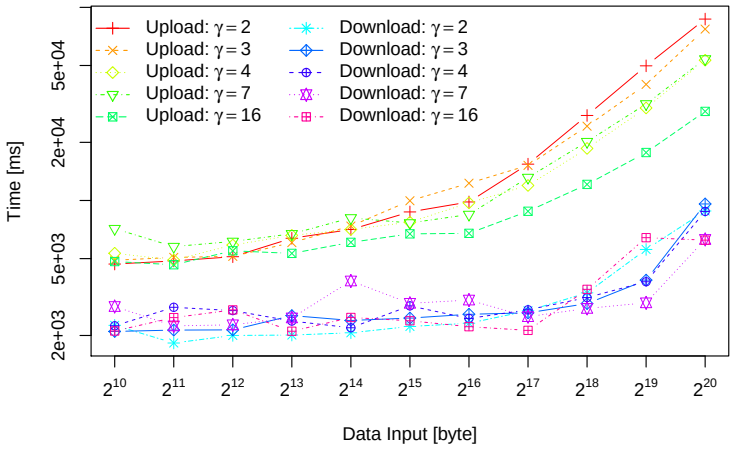


Fig. 11: Flickr Performance, *SL*-approach

Table 2: Applicable Painters on Photo Sharing Sites

Hoster	Encoder	γ
Picasa	*ML*	256
Flickr	*ML*	4
	ML	16 + ECC
Facebook	*SL*	4

encoding as well: The applied compression within Facebook increases the file size of the downloaded image resulting in an increased download overhead.

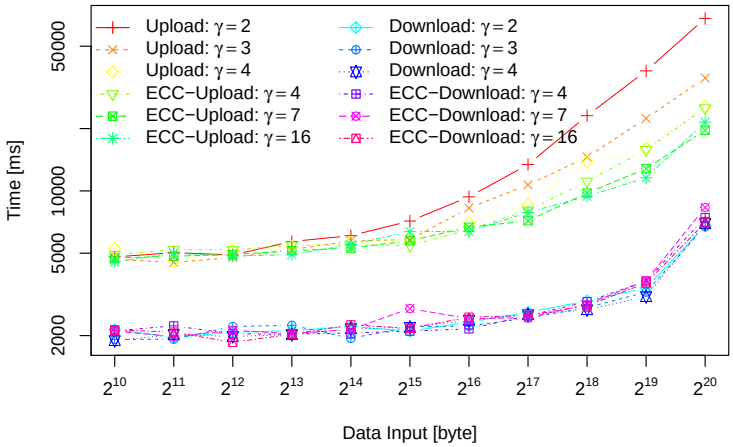


Fig. 12: Flickr Performance, *ML*-approach

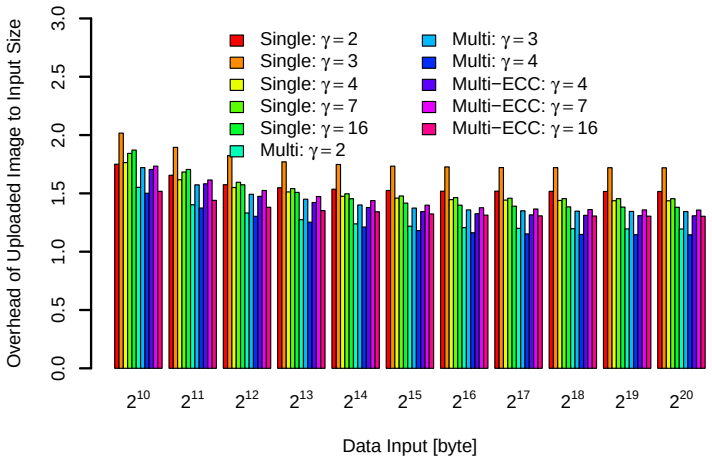


Fig. 13: Size of the uploaded Images on Flickr

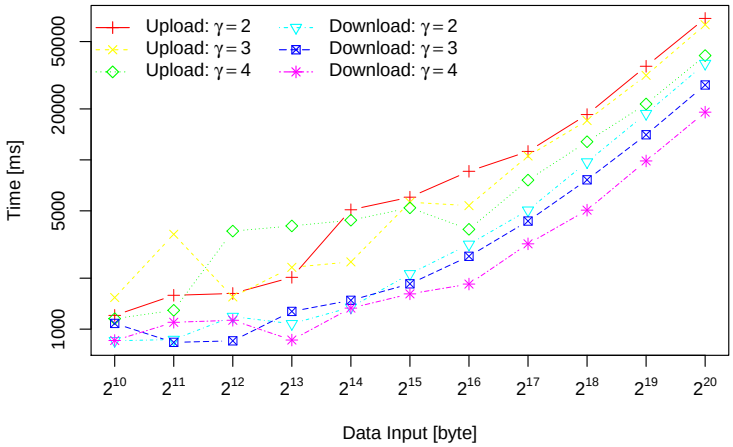


Fig. 14: Facebook Performance

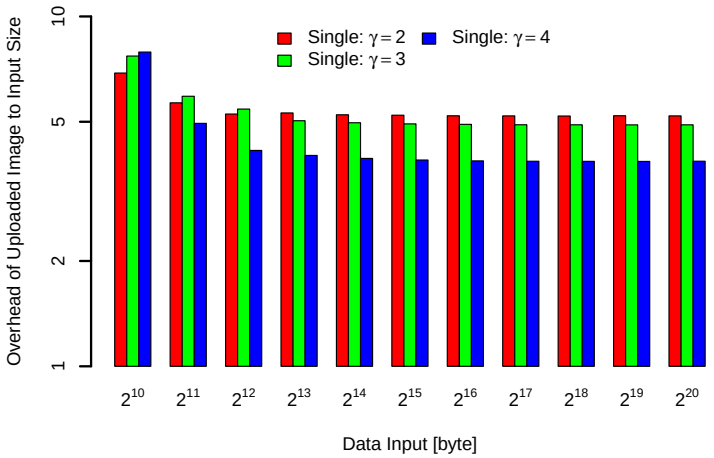


Fig. 15: Facebook image size

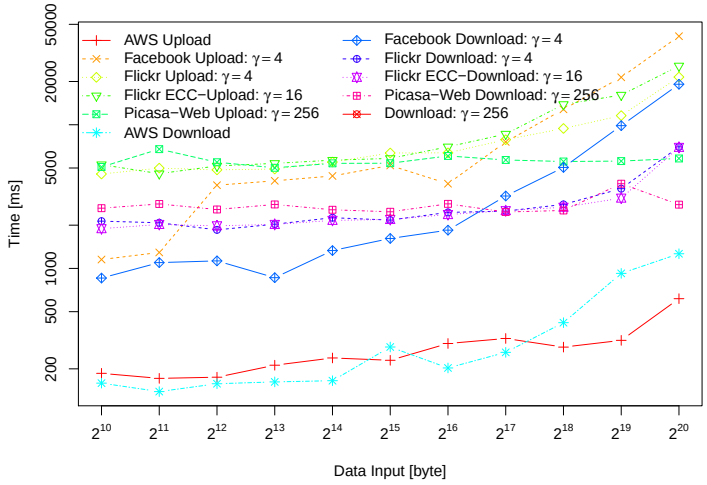


Fig. 16: Performance Comparison

6.4 Comparison of the Performance of the Photo Sharing Websites

We compare the performance of the analyzed photo sharing websites with our *SL*-approach and *ML*-approach where we rely on the values for γ defined in Tab. 2 based on our performance findings: The defined encoders including the values for γ are compared against AWS S3 as typical opponent to our approach.

Fig. 16 shows the absolute comparison related to the performance of the upload and download where the y-axis scales logarithmically. The overhead of uploading any data to photo sharing websites in our approach is generated by the hosters based on the immediate processing of any incoming data.

The download-performance scales similar and is more based on the access of the original data on the one hand and on the size of the data to be transferred on the other hand. As a consequence, the *ML*-encoder with $\gamma = 256$, applicable on Picasa-Web, is only twice as slow as AWS S3 including the extraction of the data out of the image.

The connection between the performance and the data size is represented by the comparison of the file sizes in Fig. 17. The size of the generated images scales with size of the underlaying data whereas the uploading and downloading performance relies on this size.

The images stored on Facebook scale at an overhead of 3.85 for larger data resulting in a worse download performance than the *ML*-encoder with $\gamma = 256$ applicable on Picasa-Web. This size of the generated images of this encoder scales with no overhead for larger data sizes resulting in download-performances comparable to normal cloud storage systems.

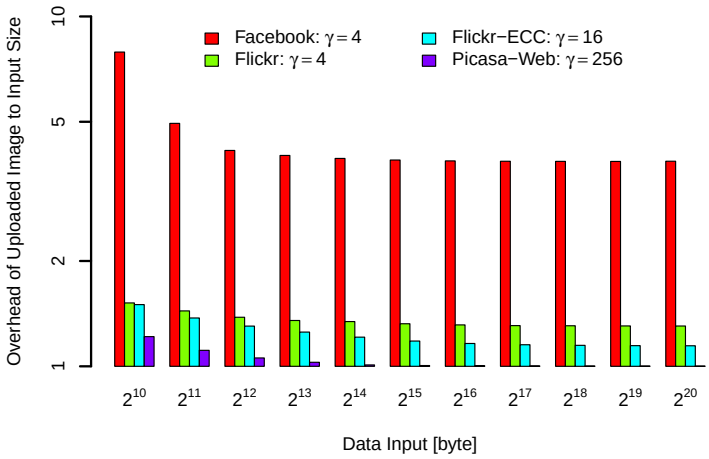


Fig. 17: Size Overhead

7 Conclusion

Photo sharing websites represent a cheap alternative for common cloud storages commonly accessible over similar APIs. Even if the accessibility and availability is comparable to normal blob-storages, the utilization of photo sharing websites as storage backends comes at a price: The processing of the images especially related to the upload to the hoster generates a constant overhead compared to dedicated blob-storages. Since the pricing of these blob-storages relies on the resources utilized including not only the consumed space but also the transfer of the data, photo sharing websites are an affordable alternative when it comes to large datasets which are irregular updated like existing in archiving purposes. To satisfy this use-case, the different values for γ allows an adaptive utilization of our *SL*-approach and *ML*-approach with different photo sharing websites adhering to the underlaying processing of the images with respect to robustness and performance. The measures to ensure robust handling of the images including separate encoded meta-data and optional applicable error-correction codes make our approach usable for all different kinds of photo sharing websites. The data rates of the generated images exceeds common approaches based on QR-codes and steganography and represent a new field of data encapsulation in images for direct processing. Our extension included in the *jClouds*-API[1] allows anyone utilize the described photo sharing websites out of the box while the design of our framework allows any utilization of other photo sharing websites as long as a related API is provided.

References

[1] jClouds, "Java API for accessing cloud services," http://jclouds.org, 2012. 1, 1, 4, 7

[2] D. Beaver, S. Kumar, H. C. Li, and J. S. an Peter Vajgel, "Finding a needle in Haystack: Facebook's photo storage," in *"USENIX OSDI"*, 2010. 1, 1, 4.3

[3] J. Xu, A. H. Sung, P. Shi, and Q. Liu, "Jpeg compression immune steganography using wavelet transform," in *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2 - Volume 2*, 2004. 2

[4] T. Langlotz and O. Bimber, "Unsynchronized 4d barcodes: coding and decoding time-multiplexed 2d colorcodes," in *Proceedings of the 3rd international conference on Advances in visual computing - Volume Part I*, 2007. 2

[5] T. Dean and C. Dunn, "Quick layered response (qlr) codes," Deparment of Electrical Engineering, Stanford University, Tech. Rep., 2012. 2

[6] K. T. T. Hiroko Kato and D. Chai, "Novel colour selection scheme for 2d barcode." in *International Symposium on Intelligent Signal Processing and Communication Systems*, 2009. 2, 3.1, 3.2

[7] C. Henderson, "PNGStore - Store JS/CSS in compressed PNGs," http://iamcal.github.com/PNGStore/, 2012. 2

[8] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," in *Journal of the Society for Industrial and Applied Mathematics*, 1960. 4.2, 5

List of Figures

1 Costs for Storage . . . 3
2 Examples of generated Images . . . 5
3 Encodings for Putting Bytes in Images . . . 6
4 Failure Rate on Flickr . . . 11
5 Failure Rate on Facebook . . . 11
6 Areas of Image . . . 13
7 Upload and Download Workflows . . . 13
8 Size of Image Files . . . 15
9 Picasa Performance, *SL*-approach . . . 16
10 Picasa Performance, *ML*-approach . . . 17
11 Flickr Performance, *SL*-approach . . . 18
12 Flickr Performance, *ML*-approach . . . 19
13 Size of the uploaded Images on Flickr . . . 19
14 Facebook Performance . . . 20
15 Facebook image size . . . 20
16 Performance Comparison . . . 21
17 Size Overhead . . . 22

List of Tables

1 Applicable Painters on Photo Sharing Sites 10
2 Applicable Painters on Photo Sharing Sites 18