Managing Redundant Content in Bandwidth Constrained Wireless Networks

Tuan Dao
University of California, Riverside
tdao006@cs.ucr.edu

Amit K. Roy-Chowdhury
University of California, Riverside
amitrc@ee.ucr.edu

Harsha V. Madhyastha
University of California, Riverside
harsha@cs.ucr.edu

Srikanth V. Krishnamurthy
University of California, Riverside
krish@cs.ucr.edu

Tom La Porta
The Penn State University
tlp@cse.psu.edu

ABSTRACT

Images/videos are often uploaded in situations like disasters. This can tax the network in terms of increased load and thereby upload latency, and this can be critical for response activities. In such scenarios, prior work has shown that there is significant redundancy in the content (e.g., similar photos taken by users) transferred. By intelligently suppressing/deferring transfers of redundant content, the load can be significantly reduced, thereby facilitating the timely delivery of unique, possibly critical information. A key challenge here however, is detecting 'what content is similar,' given that the content is generated by uncoordinated user devices. Towards addressing this challenge, we propose a framework, wherein a service to which the content is to be uploaded first solicits metadata (e.g, image features) from any device uploading content. By intelligently comparing this metadata with that associated with previously uploaded content, the service effectively identifies (and thus enables the suppression of) redundant content. Our evaluations on a testbed of 20 Android smartphones and via ns3 simulations show that we can identify similar content with a 70% true positive rate and a 1% false positive rate. The resulting reduction in redundant content transfers translates to a latency reduction of 44 % for unique content.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication; Distributed networks

Keywords

Challenged Networks; Image Similarity; Redundant Content Reduction

1 Introduction

A recent report estimates that there were around 350 million photos uploaded to Facebook and more than 50 million photos uploaded to Instagram on a daily basis in 2013 [1]. While new technologies attempt to increase wireless capacity (e.g., MIMO), users still find that their image/video uploads stall often when using the cellular infrastructure [2]. Moreover, the demand for wireless capacity is likely to be exacerbated when unforeseen events such as natural disasters occur. In such scenarios, the network further gets overwhelmed due to a combination of the physical destruction of the underlying infrastructure (which severely impacts both network capacity and coverage) [3][4] and users generating more content than usual [5]. For example, there was a sudden increase in the number of images related to the hurricane Sandy that were uploaded to Flickr during the hour when the hurricane made landfall in New Jersey [6]. Even disaster rescue teams may upload images/videos to a control center, to allow the center to appropriately distribute resources/help. The higher traffic demands combined with the strapped wireless infrastructure can significantly hinder information delivery in such scenarios [7].

The large volume of images/videos that users attempt to transfer during such events is likely to have significant redundancies in information (photos of the same event taken by different users). For example, Weinsberg *et al.* [7] study the images taken by people in the San Diego fire disaster in 2007 and the Haiti earthquake disaster in 2010. They found that 53% of the images in the San Diego set and 22% of the images in the Haiti set were similar to each other (and in essence contained redundant content). Suppressing transfers of such redundant content can ease the load on the network, and allow unique[1] information (possibly critical) to be transferred with low latency. Subsequently, the redundant content can be lazily uploaded when the network conditions are more benign. Content suppression and lazy uploading can also benefit users in more generic settings (e.g., a flash crowd scenario during a sporting event); users can save on their cellular data plans, as well as the energy on their smartphones. These are likely to be taxed when the available bandwidth is low.

Arguably, the biggest challenge in suppressing the transfer of redundant content is to determine whether or not content generated by disparate clients are similar (e.g., photos of the same event, cap-

[1]In the context of this paper, unique information refers to content in dissimilar images, which contain objects or surroundings that are not covered in other images.

tured almost at the same time). This is inherently hard since the service to which clients are uploading images (e.g, Flickr or a server at a disaster control center) must make this determination *before* a client uploads the image content. Even if any one among a set of similar photos has been previously uploaded, the service has to determine if a *second* photo [2] that is being considered for upload is similar to the one that has already been transferred; if the transfer of the first photo is underway, the process is even harder.

The computer vision community has studied the problem of identifying similar images largely in the setting where the two images being compared are at the same client/server. However, requiring clients to upload images before the service can check for similarity with previously uploaded images nullifies the utility of our framework. Therefore, to suppress the uploads of redundant image content, we leverage the *metadata* used by the computer vision techniques that detect image similarity. Specifically, we have the client first extract metadata from the image it wishes to upload, and upload this metadata to the service. The client then uploads the image content only if the service is unable find any similar images using the uploaded metadata.

This approach however presents a fundamental trade-off. On the one hand, the more fine-grained the metadata extracted by the client from its image, the better the service's ability to correctly identify whether similar images have been previously uploaded. It is important to ensure not only a low false positive rate so that clients do not miss uploading critical images but also a high true positive rate to reduce as much of the redundant content as possible. On the other hand, it is vital that the metadata extraction at the client, the metadata exchange between the client and the service, and the metadata-based lookup by the service all be lightweight. If not, high processing overheads at the client-side or server-side, or large delays incurred in transferring the metadata over the network, can render moot our goal of reducing image upload times by suppressing the uploads of redundant content; the same time can instead be spent to simply upload the image.

To address this trade-off between minimizing the metadata-related overhead and maximizing the accuracy of suppressing redundant content, we break up the photo uploading process into multiple phases. Our goal here is that, when a client is attempting to upload a photo, if no similar image was previously uploaded to the service, we seek to determine this with the least amount of metadata exchange, so that the upload of the image's content can begin at the earliest. For this, the first phase involves the exchange of a very small amount of coarse level metadata between the client trying to upload the photo and the service, which allows us to determine if a more careful comparison is even necessary; if there are no images that match the candidate image to be uploaded even at this level, the client can simply proceed with the upload.

If matches are found in the first phase, we employ a series of vision algorithms and exchange fine-grained metadata to increase the fidelity of the comparison. We first seek to minimize the amount of metadata/processing needed (we combine [8] and [9]). However, this does not adequately ensure that false positives are rare. Hence, we slightly increase the overhead by adding additional metadata using the approach in [10]. While this allows us to bring down the false positive rate, we are still unable to reach a reasonably high true positive rate. Hence, we incorporate a third phase which involves human feedback based on thumbnails (again, we increase the metadata by a small amount) returned from the service; this drives up the true positive rate without introducing additional false positives. In combination, these three phases are able to significantly reduce the amount of redundancy in transferred content and thereby the congestion, while ensuring very low false positive rates and low processing overheads at the client and the server.

[2]We use the terms image and photo interchangeably.

Our contributions: In this paper, we propose a framework for identifying and suppressing the transfer of redundant image content in bandwidth constrained wireless networks. A key component of our framework is the aforementioned three-phase approach for metadata exchange between the generators of content (smartphones) and the service which receives the images. The framework allows a client to estimate if the service is already in possession of content that is similar to that in an image being considered for upload. If the estimation suggests that this is the case, the image upload is suppressed and deferred for a lazy transfer at a later time when conditions are more benign; else the transfer proceeds.

We implement and evaluate our approach on a 20-node Android smartphone testbed in various conditions with the Kentucky [11], and an US cities image data set that we put together. We find that our multi-stage approach for uploading images correctly identifies the presence of similar images on the service with $\approx$70% accuracy, while ensuring a low false positive rate of 1%. More importantly, our framework's suppression of uploads of similar content enables the network to tolerate 60% higher load (for target delay requirements), as compared to a setting without our framework. We obtain similar results even at scale, when using ns-3 based simulations. Finally, we also show that the overheads imposed by our framework in terms of bandwidth and energy are very small, therefore making it viable for use.

2 Related Work

In this section, we briefly overview relevant related work.

Improving network performance and reliability during disasters or flash-crowd events: There has been research on the impact of flash-crowd events [12, 13, 14] and natural disasters [3, 4] on network performance and connectivity. Proposed solutions allow the network to adapt and survive in such scenarios [15, 16, 17]. From among these, the work that is closest to ours is CARE [7], which is a framework for image redundancy elimination to improve content delivery in challenged, capacity-limited networks. While the premise is similar, our work differs in terms of how image similarity is detected. In CARE, it is assumed that central infrastructure is unavailable and thus, content is transferred in a peer-to-peer fashion. Similarity detection takes place locally at a chosen node, where the images to be compared are first made available. This node transfers unique images when a DTN (delay tolerant network) relay with infrastructure connectivity is available. In our system, we assume that users have access to central infrastructure; only metadata that is extracted from the images is used for similarity detection. Our goal is to preemptively suppress the uploading of similar images on the bandwidth constrained wireless network. We acknowledge that the authors of CARE were the first to suggest the use of similarity detection in images to reduce content; we believe that our proposed work is complementary to CARE, both in terms of the setting considered and the actual approach itself.

Data deduplication in network services: Orthogonal to our work, data deduplication has been used to reduce storage capacity [18] and bandwidth [19, 20] requirements in systems which involve storing and moving large amounts of data. However, these efforts do not consider content semantics as we do here.

Image similarity detection: Our work leverages state-of-the-art approaches in computer vision for image feature extraction and object matching. Over the last decade, many algorithms have been proposed for robust extraction of global [9, 21] and local key-point [8, 22] features. The bag-of-words (BoW) approach, which had been originally used in text document classification, was applied in computer vision for image classification and matching by building a visual codebook from image local key-points [23]. The min-hash technique was proposed by Chum *et al.* [9, 24] to effectively estimate similarity between images represented in the BoW format. Recent work has been focusing on geometry verification to improve similarity detection accuracy [10, 25]. In section 3, we describe in detail how we effectively combine these techniques to create a lightweight, yet accurate image similarity detection system.

3 Efficient, lightweight detection of redundancies in images

Our goal is to determine if there are similarities between images that are to be uploaded by a plurality of spatially disparate uncoordinated clients. We seek to do so with a very low overhead while still sustaining a high accuracy for detecting similar images. We envision that these images are to be transferred over a wireless network to a central server. The server has access to all the images that were previously uploaded to it.

3.1 Our framework in brief

Figure 1 presents an overview of our framework, which can be adopted by any service to which users uploads photos, such as Flickr or Facebook, or even a server at a disaster response control center. When a new image is considered for upload to the server, a small amount of metadata is first extracted from the image and transmitted to the server. The server compares this metadata with that from images that were previously uploaded, and determines if a similar photo is already available. If this determination yields a positive outcome, the photo upload is suppressed for the time being; else the device seeking to upload the image proceeds to do so.

This seemingly simple high-level approach has three phases, with the aim of reducing the overhead associated with identifying similar images. The first two phases form a hierarchical, automated approach for image similarity detection. First, when a client seeks to upload an image, it extracts certain coarse-grained global features and sends these to the server. If the server finds that there is (are) a previously uploaded image(s) with similar features, it invokes the second phase. In this phase, the client intelligently combines state of the art vision algorithms to extract fine-grained local features from the image. A compact representation of these features is then sent to the server. The server performs a further comparison of these features with those in its pre-existing set of images. If there is a further match, it is deemed that similar content exists, and the upload of the candidate image is suppressed.

For all images that pass the first check but fail the second check, the server sends back thumbnails of a small set of the closest matching images in its pre-existing set to the client. In fact, in the scenarios of interest, a small set of pre-existing images may turn out be the closest matching ones to multiple images (being uploaded by disparate users) that are being considered for transfer; in such cases, the server can simply broadcast these thumbnails. If a client device is in the possession of a human user (e.g., a smartphone), the user can look at the thumbnail and then make a final decision on whether or not to continue with the image upload.

Table 1 provides a summary of the techniques used in our framework. In the subsequent subsections, we elaborate on how these techniques are combined to efficiently detect image similarity.

Scope of our work: While our approach is applicable to different forms of rich content, we limit ourselves to image/photo uploads/transfers in this work. Extension of the work to video is possible [26] but will be considered in the future. Further, our focus is

Technique	**Usage**	**Goal**	**Section**
OCS color histogram	In Phase 1: Compare Euclidean distance between color histograms to determine server has candidate similar images	Lightweight, but coarse-grained similarity detection	3.2
ORB local key-points	In Phase 2: Capture distinctive patches on an image; these can be matched to find similar images	Facilitate highly accurate similarity detection; uses image local features	3.3.1
BoW representation	In Phase 2: Compute the Bag of Visual Words (BoW) representation of an image by mapping its key-points into pre-computed clusters	Provide the inputs for computing the min-hash values	3.3.2
Image min-hash values	In Phase 2: Convert a BoW representation into a fixed number of hash values	Reduce communication and processing overhead	3.3.3
Geometry visual phrases	In Phase 2: Add geometry information to reduce false visual word matches	Reduce false positive rate	3.3.4
Thumbnail feedback	In Phase 3: Feedback image thumbnails	Use user input to increase true positive rates	3.4

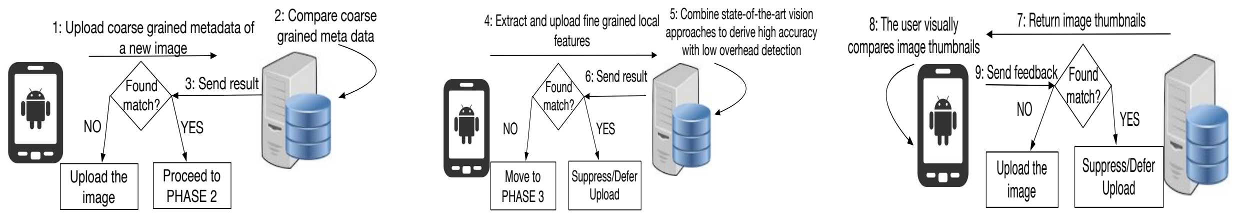
Table 1: Summary of techniques combined to form our framework

the identification/suppression of redundant content in this work. In scenarios such as disasters, it is conceivable that some images are more important than others (e.g., a human in need of rescue versus a damaged uninhabited vehicle). Thus, one could conceivably target prioritizing image uploads based on content; however, we defer studies of such possibilities to the future.

In this work, we assume that if there is no prior image that is similar to the one that is considered for transfer, the image is transferred; else it is suppressed. We do not take into account things like the quality of the image (e.g., resolution), or the coverage (e.g., close up versus wide angle) as criteria for the above determination. Accounting for these factors is a harder challenge; the service will need to delay image transfers, compare all metadata from images that are being considered for uploaded and explicitly pull images from a chosen client based on some criteria (e.g., HD quality image with a close up of a house). Furthermore, the vision algorithms that we use here will not provide such assessments.

Our work primarily targets public services that require the transfer of images (e.g. photos transferred during a disaster to facilitate rescue operations). Our approach can be potentially leveraged in flash crowd scenarios where bandwidth is scarce; for example, an image sharing service can use our approach to provide mobile users an *option* to temporarily point to a similar version of an image (that they seek to upload), which is already available on the server side. A seamless upload and replacement with the user's own image could be done lazily when the network is under less duress; from the user's perspective, such an approach would save both on the data usage (if WiFi was used instead of 4G later) and energy costs that could be heavy due to retransmissions when the bandwidth is poor.

Finally, we do not leverage device features (e.g., GPS location, geotags, camera orientation) to assess if two images could be similar; these features could be useful in reducing the search space at


1: Upload coarse grained metadata of a new image
2: Compare coarse grained meta data
3: Send result
Found match?
NO
YES
Upload the image
Proceed to PHASE 2
4: Extract and upload fine grained local features
5: Combine state-of-the-art vision approaches to derive high accuracy with low overhead detection
6: Send result
Found match?
NO
YES
Move to PHASE 3
Suppress/Defer Upload
8: The user visually compares image thumbnails
7: Return image thumbnails
9: Send feedback
Found match?
NO
YES
Upload the image
Suppress/Defer Upload

(a) Phase 1: Coarse grained metadata matching (b) Phase 2: Fine grained feature matching (c) Phase 3: Thumbnail feedback

Figure 1: Our framework for determining and suppressing images that contain similar (redundant) information

the server side (e.g., it can compare images that are taken by cameras in close proximity only). Leveraging such features is orthogonal to, and can be used in conjunction with our framework.

How do you determine if content in two images is similar?: Whether or not the content in one image is similar to that in another is a subjective matter; different human users may perceive things differently and with respect to different images as well. Moreover, a general user may choose to upload his image regardless of whether or not someone else has uploaded a similar image. We assume that (i) savings in terms of data usage and energy will incentivize users to suppress their image transfers, especially when the network is congested and, (ii) in scenarios such as disaster recovery, smartphones could be used by the relief crew, who will want to suppress redundant content to reduce congestion and thus, aid relief operations.

In this work, we seek to ensure that if it is highly likely that a typical human does not perceive that two images are similar, they are classified as dissimilar. In other words, our framework must minimize false positives when classifying images as similar. Keeping this primary goal of a very low false positive rate, we seek to eliminate redundancies via such similarity detection to the extent possible, using state-of-the-art computer vision algorithms. We use known data sets (discussed later) to get objective evaluations of our framework; these evaluations show that our framework is extremely effective in decreasing network congestion.

3.2 Phase 1: Use of a coarse-grained global feature

Global features capture the entire content in an image. Examples include the color pattern or the scene pattern in the image. A global feature is represented by a single feature vector. As color is an important image attribute, a histogram of the color distribution in an image is widely used as a global feature for determining if two images are similar.

To construct such a histogram, we use the opponent color space (OCS) [9] to determine image similarity. We use the OCS color space, since it is not very sensitive to illumination (brightness) variations, unlike the RGB space. In brief, there are three components in the OCS space: an intensity component and two opponent colors. The components in the RGB color space can be used to compute the the OCS color components using the following equation.

$$\begin{aligned} I &= (R+G+B)/3 \\ O_1 &= (R+G-2B)/4+0.5 \\ O_2 &= (R-2G+B)/4+0.5 \end{aligned} \tag{1}$$

The intensity component is quantized into 64 bins, while the other two components are quantized into 32 bins. The histogram vector is normalized so as to represent each component with 1 byte; thus, 128 bytes are used overall to represent the histogram. Once these 128 bytes are sent to the server, the server compares the bin values with those of the images that it has in its data set (previously uploaded). If the Euclidean distance of the histogram of any image on the server side and the histogram of the image about to be uploaded is less than a threshold τ_1, the system enters the second phase for similarity detection; otherwise, the client uploads the new image.

3.3 Phase 2: Using fine-grained local features

In the first phase, only the global distribution of colors and intensity were examined. If the server finds matching histograms, in the second phase, finer grained local features are extracted from the image and uploaded as metadata for further comparisons. Contrary to global features, local features are extracted from small patches in the image. When combined together, such local features (called key-points) represent the characteristics of the entire image. Fine-grained local features can be used to detect image similarity with high accuracy. Our approach for using local features consists of the following steps.

3.3.1 Extraction of key-points from an image

As mentioned above, the local features that we compare in order to assess the similarity of images are *key-points*. Key-points are small patches of an image that differ significantly from the surrounding areas (in the image). In computer vision, SIFT (Scale Invariant Feature Transform) is the most widely used algorithm for determining the key-points in images [22]. However, SIFT typically imposes a very high processing complexity and is thus, not a viable solution for resource (battery) limited devices like smart-phones. In our experiments, extracting key-points of a high resolution scenery image (approximately 2 MB of data) with SIFT requires about 30 seconds or even more.

Hence, we choose ORB [8] as our algorithm to extract image key-points, instead of SIFT. Experiments from other research groups have shown that ORB is about two orders of magnitude faster than SIFT while offering comparable results in many situations [27][28]. Each ORB key-point is described by 256 binary digits, whereas with SIFT, each key-point is described by a 128-dimensional vector. The number of key-points depends on the image size, the image resolution and the number of objects in the image. Normally, the amount of data associated with the key-points in an image is far greater than the size of the image itself! Therefore, directly comparing and matching key-points of images is not an option for our framework; this would violate our goal of exchanging a very limited amount of metadata for determining the similarity across disparate images.

3.3.2 Bag of Words (BoW) representation

Instead of directly working with image key-points, we use the bag-of-words (BoW) approach [23] to build what is called a "visual codebook." Any image can be represented as a bag of *visual words*, which is much more compact than simply representing the image via key-points. We describe below how the visual words are determined. For now, we point out that a visual word in the ORB representation is simply described by a 256 bit-vector; each element of the vector is called a dimension. A comparison of the visual words representing two images could be used to determine if the two images are similar. Representing an image by a bag of visual words is performed as follows.

Determining the visual words: First, the ORB key-points of a large set of representative images are extracted. These key-points are all grouped into a pre-defined number of clusters using any good clustering algorithm. In our approach, we use a modified version of the k-means clustering algorithm to partition and group binary vectors [29]. With this algorithm, the input key-points are mapped onto k different clusters based on the Euclidean distance between the key points and the cluster centroids. However, the Euclidean distance is not suitable for binary data such as the ORB key-point descriptors. Therefore, in our framework, we use the Hamming distance instead of the Euclidean distance. The Hamming distance between two binary vectors is simply the number of bits that are different in the two vectors.

The centroid of each cluster is randomly chosen first but is iteratively refined, as key points are added to the cluster; details are available in [30]. With the binary vector representation, in order to determine the centroid of a cluster, we count the number of zeroes and ones in each of the 256 dimensions, for all the data points (key points) that are associated with the cluster. If the number of zeroes is greater than the number of ones, the value of the corresponding dimension for the centroid is a zero, else it is a one. If there is a tie between the number of zeroes and ones, the value of the that dimension for the centroid is randomly assigned as either a “0" or a “1". Each such cluster centroid is then considered to be a visual word in the aforementioned codebook.

When an image is considered for transfer, each ORB key-point in the image is mapped on to the closest cluster centroid in terms of the Hamming distance. With such a mapping, each image is now represented by a histogram of visual words; the number of key-points mapped on to a cluster reflects the *value* of the corresponding visual word in the histogram.

We wish to point out here that the codebook can be pre-loaded onto the clients when our software framework is installed (under benign conditions of connectivity); thus, there is no need to exchange it each time a comparison is to be done across images. In the BoW approach, the number of clusters is generally chosen between tens of thousands to hundreds of thousands.

3.3.3 Reducing detection overhead using min-hashes

Transferring the histogram of visual words constructed as above will incur significant overhead since it would require at least $n * k$ bytes if each component in the histogram can be represented by n bytes and we have k such components. For example, even with n = 2 and k = 50000, this corresponds to 100 KB. To adhere to our goal of having very little overhead of exchanging metadata, we use the min-hash approach proposed by Chum *et al.* [9, 24] in conjunction with the BoW representation.

To define the min-hash function of an image in the BoW representation we do the following. First, if the value associated with a visual word in the histogram is greater than 0, the word is simply considered to be included in a set that is associated with the image. Simply accounting for whether or not a visual word is present in an image (as above) is a weaker representation of the BoW vector, since the number of occurrences of a word is not taken into account. Now that each image is simply represented by a set of words, the similarity of two images with sets of visual words I_1 and I_2 respectively, is defined by the following equation.

	A	B	C	D	E	F	I_1=ABC	I_2=BCD	I_3=AEF
permutations	3	6	2	5	4	1	2	2	1
	1	2	6	3	5	4	1	2	1
	3	2	1	6	4	5	1	1	3
	4	3	5	6	1	2	3	3	1
	Label assignments						Min-hash values		

Table 2: An example of min-hash functions: Four permutations of label assignments are shown.

$$sim(I_1, I_2) = \frac{|I_1 \cap I_2|}{|I_1 \cup I_2|} \quad (2)$$

The min-hash function approximates the similarity in Equation 2 between two images as follows. Let h be a hash function that maps all members (visual words) of set I (or I') into distinct integer numbers (called labels). The min-hash value of an image I is the minimum from all the hash values associated with the visual words in that image. Formally, for each visual word X, a unique hash value $h(X)$ is assigned. The min-hash value of image I is $H(I)$ = $min\{h(X), X \in I\}$. The client uses M different assignments of labels to the visual words. Specifically, let us say there are N visual words; each is assigned a unique label value $\in \{1, N\}$ at random. This is referred as one assignment or permutation. The client can perform M (different) such permutations and compute the min-hash value in each case. The number of identical min-hash values between two images can be used to assess the similarity level between the two.

We use an example from [9] to demonstrate how the min-hash approach works. Consider a vocabulary of six visual words A, B, C, D, E and F and three different images. Each image contains three of these visual words, specifically, I_1 = {A,B,C}, I_2 = {B,C,D} and I_3 = {A,E,F}. For each image, 4 min-hash functions are generated by using different permutations as shown in Table 2. For example, the min-hash of I_1, corresponding to the first permutation (row 1) is the value associated with C and is thus equal to 2. As I_1 and I_2 have 3 identical min-hash values out of 4, their similarity is $\frac{3}{4}$=75%; for the same reason, the similarity between I_1 and I_3 is $\frac{1}{4}$=25%.

Proof sketch: The skeleton of a simple proof for why min-hash approach yields the similarity between two images is as follows. Let $\pi(S)$ be a random permutation on a set S, and let X be the element which has the minimum hash value in $\pi(I_1 \cup I_2)$. Because π is a random permutation, the probability that X is *any* element in the set $I_1 \cup I_2$ is equal for all elements. If $X \in (I_1 \cap I_2)$, then obviously, $H(I_1) = H(I_2) = h(X)$. Otherwise, without loss of generality, assume $X \in I_1 \setminus I_2$; then, $H(I_1) < H(I_2)$. To summarize, two images have the same min-hash value if and only if the element X, which has the minimum hash value, is included in both of them. It is easy to see that, as a consequence, the probability that two images I_1 and I_2 have the same min hash value is equal to their similarity, i.e., $sim(I_1, I_2)$, as defined in Equation 2.

3.3.4 Improving detection accuracy by using geometry visual phrases (GVPs)

In the bag of words representation and the inferred min-hash information, the geometry information of each key-point (for example, the location of a key-point in the image) is lost. To reduce the likelihood of false matches because of the above, we use geometry visual phrases (GVP) in combination with the min-hash values. This reduces the false positive rates significantly.

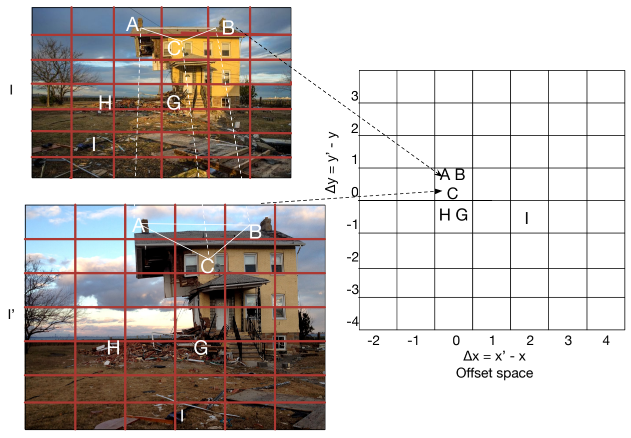


Figure 2: An example of visual phrases between two images

We describe in brief how GVPs are computed and used; more details are in [10]. Towards determining the GVPs between two images, each image is divided into a fixed number of bins (same for both images regardless of the size of the image). In each image, each key-point corresponding to a visual word is then mapped into this offset space and is represented by the co-ordinates $\{x, y\}$, of the bin index to which it belongs; this is referred to as the geometry information. Each pair of equal min-hash functions identify a visual word that occurs in both images. For each of these visual words, the differences in geometry information of the key-points (denoted by Δx and Δy) are computed. If these "difference" values for say L visual words are the same, this implies that these L key-points are likely to be mapped onto the same (corresponding) objects in the two images and thus, are said to form a *co-occurring* GVP of length L. To illustrate, let us consider the example in Figure 2, which shows two different images of a house (found online) that was damaged due to hurricane Sandy. The Δx and Δy values for the key points A, B and C are all $\{0, 0\}$. Thus, these three points together could potentially map onto something common in the two images and this forms a GVP of length 3. Similarly, for the the two key points G and H, the Δx and Δy values are 0 and -1 respectively; thus, these two key points could potentially map on to identical constructs in the two images. This is a GVP of length 2.

Given two images I and I', the similarity score based on visual phrases of length L is defined in equation 3.

$$sim^L(I, I') = \sum_s \frac{\binom{M_s}{L}}{\binom{M}{L}}, \tag{3}$$

In equation 3, M_s is the number of key-points in bin s of the offset space, and thus $\binom{M_s}{L}$ is the number of GVPs of length L in that bin. For example, in bin {0,0}, there are 3 key-points viz., A, B and C; if L = 2, the number of length-2 GVPs in the bin is 3, corresponding to AB, AC and BC. Thus, the numerator on the RHS of equation 3 is simply the total number of GVPs of length L between two images. The denominator, $\binom{M}{L}$, corresponds to the maximum possible number of GVPs of length L that can be formed between the two images, given that M min-hash functions are used. Hence, the similarity score between two images is the number of GVPs of length L that are common between them, normalized by the maximum possible number of GVPs length L that can be created by using M min-hash functions.

If the similarity scores between the user's image and a candidate image is greater than a threshold (say τ_2), the images are deemed similar.

3.4 Phase 3: Thumbnail feedback

If at the end of phase 2, if the server finds no matches for the image considered for upload, it invokes an optional phase 3, seeking user input for finally making a decision on whether or not to have the image uploaded.

Upon failing to find a match in phase 2, the server rank orders the images in the candidate set based on the similarity scores with respect to the image being considered for upload. For each of the top k images in this ordered list, a small thumbnail is sent back to the client device; photo sharing services typically generate a thumbnail for every image at the time it is uploaded [31]. The user of the client device can visually compare her image with the received thumbnails and assess whether or not similar images are already available at the server; based on this, she can decide whether or not to transfer the image.

3.5 Handling parallel transfers of similar content

Thus far, we implicitly assumed that an image being considered for upload is compared with images that were already uploaded previously and stored in the server database. However, it is quite possible that in our scenarios of interest, multiple user devices attempt to upload similar images almost at the same time (close to when the event is occurring). Due to the shared access to the wireless medium, these attempts could be proceeding in parallel. If bandwidth is limited, it becomes important to reduce the load especially in such settings; for example, multiple such critical events (people needing to be rescued) could be ongoing at the same time and it is desirable to have (unique) information associated with all such events. The challenge here is to essentially compare such parallel uploads and determine if such attempts are towards transferring similar content.

When a client sends an OCS histogram of a new image, the server inserts an entry (with this information) for the image, into a queue. When there are other parallel uploads, the server not only compares the histogram of a candidate image with that of the images in its database, but also with the histograms of entries in this queue. If there are similar histograms (in either the database or in the above queue), the client is instructed to upload the local features as before. When the server receives the local features of an image, it associates them with the proper entry in the queue. It then compares these local features with the images from the database that were classified to be likely candidates with similar content *as well as* the local features of entries in the queue that are already available (with matching histograms). If there is a match with either, the image transfer is suppressed and the corresponding entry is deleted from the queue; else, the client is instructed to upload the image. After an image is completely received, the corresponding entry is deleted from the queue and the image is added to the server database. We point out that we only use the first two phases in determining if images that are considered for upload almost simultaneously, are similar. When this determination is taking place, only metadata of the images is available at the server side (the thumbnails of such images are not yet available). Since we desire that the decisions on whether or not to upload be quick, we avoid waiting for the complete information towards generating thumbnails and providing subsequent user feedback; this would cause delays and affect user experience. However, recall that the thumbnail feedback in phase 3 is mainly to help increase the true positive rate; thus, for the images considered for upload almost simultaneously, our system still achieves a low false positive rate.

4 System Implementation

In this section, we describe the prototype implementation of our framework. Our prototype consists of a central server which accesses a database where a set of previously uploaded images are stored. A number of mobile client devices generate new images and attempt to upload them to the server. We use the Kentucky image data set (described later in Section 5) to learn the appropriate values for the parameters in our implementation.

4.1 Image server

The server stores the images that it receives in a central database. For each image, it extracts and stores the image's 128-byte OCS histogram and the image's min-hash values as described in Section 3. At the server side, we choose to construct 512 permutations (recall Section 3.3) and determine the corresponding hash values for each image; each hash value is stored using 2 bytes. As one might expect, the larger the number of permutations, the higher the accuracy in similarity determination. However, Zhang *et al.* [10] showed that when more than 512 hash functions are used, the gain in accuracy is at a point of diminishing returns due to an increase in the imposed processing overhead.

For each hash value, the server also stores the geometry information of the visual word that corresponds to that min-hash function (the visual word which is assigned the minimum value by the hash function). Specifically, for each visual word, we store the x,y indices of the bin to which the key-point associated with that visual word belongs, as described in Section 3.3. For each image, we use a 10x10 bin-space as in [10]; thus, the geometry information of a min-hash value consists of 1 byte, including 4 bits for the horizontal bin index and 4 bits for the vertical bin index. Therefore, the total byte count to capture the local features of an image is 1536; this includes 1024 bytes for the min-hash values and 512 bytes for the geometry information. In total, we impose only 1664 bytes overhead for each image, together for both the global and local features. This is less than 1% of the size of a normal quality image taken with a modern smartphone. Our server application is implemented in C++ and uses the OpenCV library [32] to extract global and local features of the images.

Server operations: When a client application is about to upload an image, it sends the OCS histogram of the image first. The server searches its database to find images with similar histograms (the component values are within a threshold τ_1 of the incoming image's histogram) based on Euclidean distance. If such similar histograms are found, the corresponding images are considered as candidates for containing the content in the image to be uploaded; then, the client is asked to transfer min-hash values and the geometry information of its image. If no similar histograms are found, the client is instructed to upload its image.

In the next phase, when the local features of the image are received, the server calculates the geometry similarity score of that image with respect to each of the images in the candidate set according to Equation 3. Here, the scores are based on GVPs of length 2; it has been shown that this provides a good enough detection accuracy when compared to using GVPs of other lengths [10]. If any of these similarity scores is $\geq \tau_2$, the server deems that the content is similar and notifies the client application to suppress the image upload.

Otherwise, the server chooses those images that have the highest similarity scores and sends back thumbnails of these, to the client (a delayed multicast is possible to reach a plurality of clients, but we don't implement this). The human user of the client device can then check to see if the images are similar, and only choose to upload the image if she feels that they are not. We assume that users

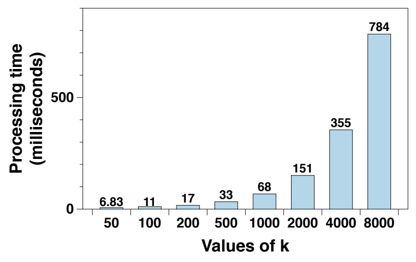


Figure 3: Search time for different values of k with ***knn***

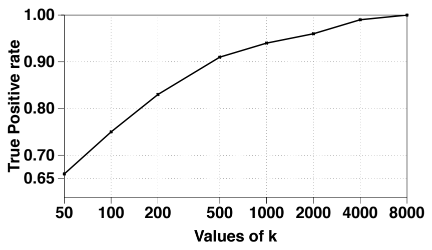


Figure 4: Percentage of images similar to an incoming image found

are objective and suppress an upload if a thumbnail is indeed of a similar image (we discuss our data sets and their objective usability for determining the similarity of images in Section 5).

Fast histogram matching in Phase 1: For each OCS histogram that is received (from the clients), the server needs to find the set of images in its database with similar histograms. The database could potentially contain a large set of images, and brute-force checks are thus not viable. To achieve an efficient search, we utilize the fast "k nearest neighbor" search (***knn***) to find a good approximate set of the candidate images. We use the FLANN library [33] which is freely available for knn search. The histograms of all the images on the server side are grouped into hierarchical clusters. Specifically, based on the histograms, all the images are first grouped into N clusters; each such cluster in turn is recursively partitioned into N sub-clusters and so on, up to a maximum number of iterations. Here, we choose a default value of N=32, as suggested by the FLANN library.

Choosing the key parameter for the **knn** *search:* One important parameter when using ***knn*** search is the value of k, the maximum number of nearest neighbors the library should return. The higher the value of k, the library explores a larger number of branches in the cluster-tree and is thus able to find a larger set of similar images. However the search time also increases.

To determine a good value for k, we conduct an experiment using the Kentucky image set with 10200 images. This set contains groups of images that are similar; each group contains four images (ground truth). We build a test set that consists of 500 images, such that no image is similar to any other image in the set. For each image, we execute FLANN with different values for k, and record the processing times and the number of similar images found on the server. Figure 3 shows the search time (for query processing) for different values of k, and Figure 4 shows the accuracy of the search results in terms of the percentage of similar images that are found for a candidate image (as compared to the ground truth). These graphs show that, if we set k=500, we are able find over 90% of similar images with an expected processing delay of only about 33ms. Thus, we choose this to be the default value in our server implementation.

4.2 Client application

The client app is implemented on Android smart-phones using Java and native C++ code (JNI). Upon capturing an image, our client app is invoked for attempting an upload of the image to the server. The Java code is only for the graphical interface; the image processing code is written in C++ and is linked with the OpenCV library for feature extraction.

Client operations: First, the client application extracts the OCS histogram of the image and sends it to the server. It then awaits a server notification with regards to whether or not there are images with similar histograms in the server's database. If such candidate images exist, the client app extracts the local features, i.e., the min-hash values and their corresponding geometry information, from the image. To calculate the min-hash values, first the ORB key-points of the images are extracted. Next, the client application converts the image into the BoW presentation by mapping the key-points into visual words based on a vocabulary file. The vocabulary file contains the book of visual words (codebook) that is pre-built and preloaded from a set of training images (these are also available to the server). We choose to use a codebook of 20000 clusters. With this, the processing time at the client for each image is approximately 1.2 seconds. With a larger codebook, (e.g., with 50000 clusters) the processing time is around 3.5 seconds. Thus, choosing this larger codebook will degrade the performance of our framework (increased latencies). Furthermore, with 20000 visual words, we only need to use 2 bytes to represent each cluster and this limits the metadata overhead; larger numbers of clusters will increase these overhead costs.

Next, the client reads 512 different hash values (permutations) from a pre-built data file; for each permutation, a unique label is assigned to a hash value and thus, each visual word. It identifies the label of the visual word with the min-hash value for each permutation, and computes the geometry information for the visual word associated with that value. It sends both the min-hash values and the geometry information back to the server (local features).

Pre-installed data on client side: We pre-install a vocabulary file for the BoW processing and a permutation file for determining the min-hash labels for images on the client side. Thus, this information does not need to be exchanged for each image. With 20000 clusters, the size of the vocabulary file is only 640 KB; each cluster centroid of an ORB key-point is just 256 bits (32 bytes). The permutation file contains 512 permutations; each permutation in turn contains 20000 assignments which map each cluster on to a unique 2-byte label value. To reiterate, each permutation essentially reorders the identification labels to be assigned to the clusters. Thus, the size of the permutation file is $\approx$ 20MB. These files are also available to the server (which essentially provides the software at install to each client). The only time that these files need to be rebuilt is if there are large changes to the image database maintained by the server.

5 Evaluation

In this section, we describe our evaluations of our framework. We perform experiments on a testbed of Android phones, as well as simulations using ns3 to showcase the performance as well as the benefits of our approach.

5.1 Training and test image sets

We begin with describing our image data sets and how we use them to evaluate the accuracy with which our framework can identify similar images. We use the Kentucky image set, which has been widely used in computer vision, primarily because of the availability of ground-truth information. We also use an image set of US cities that we collected on the Internet as described below.

The Kentucky image set [11]: The image set consists of 10200 images forming 2550 groups. In each group, there are 4 images of the same object taken from different angles; such images match our requirement/definition of similar images.

The city image set: We use the Bing image search service to find one image each for 5000 popular US cities. As these pictures are taken from different cities, there are no pairs of similar images in the entire image set.

We resize and increase the file size of all the images to $\approx$ 700 KB using ImageMagick [34]; this reflects the average size of normal-quality images taken by smartphones today.[3]

Building the training and test data sets: We evaluate the accuracy of our framework by partitioning our image data sets into a training set and test set. We pre-upload images in the training set to the server and evaluate the accuracy with which our framework is able to identify images in the test set (when we try to upload these images) as having corresponding similar images on the server.

We randomly pick 2000 images from the US cities set and 2000 images from the Kentucky set to create a test set of 4000 images. Though images from the Kentucky data set are chosen randomly, we ensure that no more than 2 images are taken from the same group (the aforementioned 4 images of the same object). The remaining images from the Kentucky data set and the US cities data set are used as our training set; this set is used to build the codebook for the BoW representation and stored at the server.

To eliminate biases with a specific test set, we construct 5 different test sets (by randomly choosing images from the Kentucky and US Cities data sets) and the corresponding training sets. By default, the results reported are the average from our experiments with these 5 different sets.

With the above, we essentially ensure that for each image in our test set that is taken from the Kentucky set, there is at least one similar version of the image in the server database; this can be used as ground truth while estimating our true positive rates in identifying similar/redundant content. For each image taken from the US cities set, there is no similar version in the server database; thus, this set is useful in estimating the false positive rates with our framework.

Remarks: We do recognize that the Kentucky and the US cities data sets contain largely dissimilar images. We tried to perform experiments with a large set of images from a disaster scenario (Hurricane Sandy) but had difficulty in establishing the ground truth for the purposes of quantifying true and false positive rates. While we believe that our framework works well in such cases (based on some limited experiments where we tried to upload about a 100 photos and manually checked for similarity), we need a set of volunteer users to categorize whether the images are similar or not; for a large set of images, this was difficult to do. Using the Kentucky and US cities data sets allowed us to evaluate the accuracy of our framework without human involvement in an objective way.

When emulating human feedback based on thumbnails, we again rely on the objectivity possible with the above data sets. If two images are indeed similar (based on the ground truth), we assume that the user will correctly classify it to be the case; if the images aren't, we assume that the user will correctly decide to upload her image.

A training set of images (and a corresponding codebook) for a specific disaster location, can be built by using images of the same location before the disaster *and* images of the same kind of disaster (for example, an earthquake or a wildfire) that had previously

[3]With modern smartphones, the average file size of high quality images can be between 2 and 2.5 MB [35].

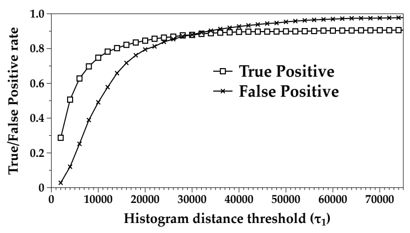


Figure 5: Accuracy with different histogram thresholds

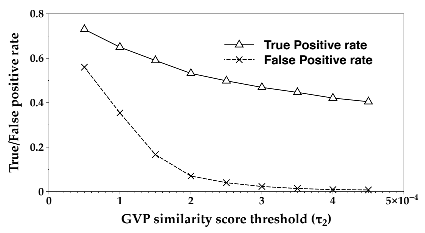


Figure 6: Accuracy with different GVP similarity scores

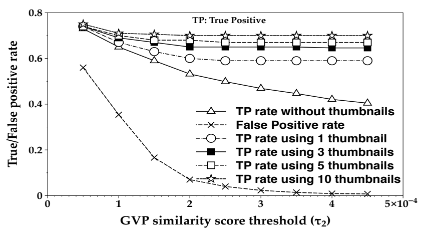


Figure 7: Accuracy with different numbers of image thumbnails

occurred at other locations. A codebook built based on the two image sets is likely to contain key points that are similar to the key points in the images captured at the disaster scene. This determination is based on results from prior research on disaster images; specifically, Yang *et al.* [36] found that images of the same kind of disaster have many similar local features.

5.2 Experimental setup

Our experimental system consists of a server with an associated database; all the images in the training set and their global and local features are stored in the database. We have 20 Android-based smartphones as our client devices; the test images are divided equally among these phones. The smartphones connect to an access point (AP) on a WiFi network; the server is also connected directly to the AP via a 100 Mbps Ethernet cable. To emulate bandwidth constrained settings, we set the network bitrate to 6 Mbps. We experiment with different workloads (upload rates) from our smartphones. Note that we vary the network bandwidth in our simulations in Section 5.8; we also consider uploads using the cellular infrastructure in those studies.

Remark: Unfortunately, we were unable to showcase the performance of our framework via real experiments on cellular networks. Specifically, we do not have a sufficiently large set of phones to create enough load that strained the network bandwidth. Further, we were unable to determine the bit rate on the LTE links and could not accurately characterize the network load; thus, it was difficult to objectively quantify the benefits from our framework. However, in the scenarios of interest (e.g., disasters), we expect that there will be sufficiently large user activity that will strain the capacity of the network [7, 17].

5.3 Accuracy of detecting similar content

Detection accuracy with global features only. First, we examine the accuracy with which Phase 1 of our framework determines if or not the server is in possession of a similar image as compared to one being considered for upload. Recall that image similarity is determined here only by comparing the global OCS histogram associated with two images. Figure 5 shows the true positive rates (correctly detecting similar content) and the false positive rates (wrongly classifying images as containing similar content) with different histogram distance thresholds. If we set a very low threshold (meaning that the Euclidean distance between the histogram of the image to be uploaded and a candidate image in the server database should be very small), we will end up not identifying any similar images; here, the true positive rate will be very low. To increase the true positive rate, we will need to increase the threshold so that we have a bigger likelihood of identifying candidate images in the server database, but this will have the undesired effect of increasing the false positive rate, since some wrong images in the database will also be classified as candidates for similarity checks.

Based on Figure 5, we choose a threshold of τ_1=14000 towards achieving $\approx$ 80% true positive rate; however, this results in a 63% false positive rate, which we seek to drastically decrease with Phase 2.

Improving detection accuracy with local features. In Phase 2 of our approach, the assessment of image similarity is refined by calculating the similarity scores based on GVPs (see Equation 3). Figure 6 depicts the true and false positive rates after this phase, when different similarity score thresholds are used. We observe that with a very low threshold, the false positive rate is very high (images are wrongly classified as similar) but then drops drastically as we increase the threshold. However, increasing the threshold decreases the true positive rate as well, since similar images are discarded for “not being good enough". To avoid missing critical image uploads, a very low false positive rate ($\approx$ 1%) is desirable. If we set a threshold to achieve this, the true positive rate is $\approx$ 47%; this implies that approximately half of the images which have redundant content are detected and are subsequently suppressed at the end of this phase.

Feeding back image thumbnails to further increase the true positive rate. To further improve the detection of similar images, in Phase 3, the server sends back thumbnails to the user for visual inspection (Section 3.4). In our experiments, the size of an image's thumbnail is $\approx$ 11 KB; we believe that this a reasonably small volume of data needed for improving accuracy.[4] Figure 7 shows the increase in accuracy when different numbers of thumbnails are fed back to the user. With 3 to 5 image thumbnails, we find that the true positive rate increases to about 68% (from 47% at the end of Phase 2). Beyond that, we find that we hit a point of diminishing returns; for example, with 10 thumbnails, the true positive rate increases to just over 70%. Note here that the false positive rates do not change after Phase 3 (the human accurately determines if the content is similar or not).

Detection accuracy with parallel uploads. Next, we consider the case where multiple client devices are attempting uploads of similar content in parallel (almost simultaneously). Specifically, we conduct an experiment where three smartphones attempt parallel uploads of an ordered set of 500 images to the server. The database on the server side does not contain any images that are similar to the test images. We manually ensure that the images at the same position in the ordered sets at the three clients are similar; for example, the first image on the first client is similar to the first image on the second and the third client and so on.

First, we conduct the experiment without performing any similarity detection; as a result, all the 1500 images from the clients are uploaded to the server. Next, we implement similarity detection using the process described in Section 3.5. Here, we observe that only 585 images are uploaded to the server. Specifically, 500 images with unique content and 85 images with redundant content are

[4]The overheads due thumbnails are discussed later.

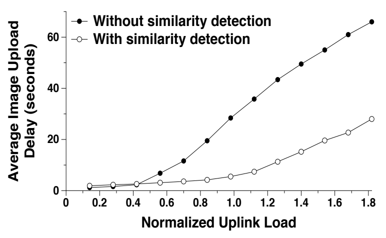


Figure 8: Impact on network load and image transfer delay (Ideal case with 100% detection accuracy)

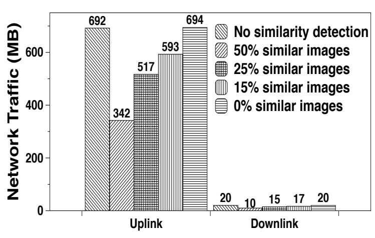


Figure 9: Varying the proportion of similar images at the server (Ideal case with 100% detection accuracy)

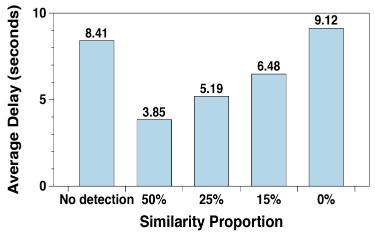


Figure 10: Upload delay with different proportions of similar images (Ideal case with 100% detection accuracy.)

uploaded; this corresponds to a 17% contribution from redundant information.

5.4 Impact of redundant content suppression on network performance

In this section, we seek to understand the impact of redundant content reduction on network performance. Specifically, we seek to quantify the impact on (i) the delay experienced during image uploads (where we can expect a decrease) and (ii) the total sustainable load (where we can expect an increase). We also quantify the overheads due to our approach.

For the experiments in this section, each smartphone sends a test set of 50 images, back to back to the server. The test set consists of 25 images from the Kentucky data set with similar versions on the server, and 25 images from the US cities set. The total size of the test set is $\approx$ 32 Megabytes. Subsequently, we vary the proportion of the redundant content in the test set and quantify the impact on network performance. First, we show the results in an ideal case wherein all redundant content is correctly detected and suppressed; in other words, we assume a 100% detection accuracy unlike what we expect in practice. With our framework, the proportional improvements are reduced by a factor equal to the complement of the true positive rate; we show this later in Section 5.7.

5.4.1 Delays under different network loads

The normalized network load (also referred to as simply network load) is defined as $\frac{n\lambda}{\mu}$, where n is the number of devices in the network, and λ is the load generated per device, and μ is the rate achievable on the transfer link. To vary load, we first fix λ, but vary the number of clients that are attempting image transfers per unit time (n). Each client transfers/suppresses one image completely, before attempting the next transfer. Specifically, we assume that a new image is generated every t seconds. We set $t = 6$ seconds (we have other results but do not report them as they are similar), which implies that a client device (given the 32 MB content volume consisting of 50 images) generates an average load of $\lambda = 0.85$ Mbps. With the wireless bandwidth of 6 Mbps (μ), this corresponds to *each* client generating a *normalized* load of 14% (0.85/6) , on average.

Figure 8 demonstrates the reduction in the delay experienced under different normalized network loads, due to similarity detection/redundancy elimination. The delay experienced by an image is defined as the duration between when the image is generated and when it reaches the server; the delay is computed for only those images that are transferred to the server. Under light network load (e.g., below 40%), sending the images directly without any similarity detection/redundancy suppression is faster! This is because in these regimes, there is no congestion and it is possible to transfer images without much delay; the process of similarity detection adds processing/metadata exchange delays but does not contribute to a reduction in congestion.

However, when the load $> 50\%$, the network transitions into a congested state; this is the regime where redundant content reduction will benefit performance. The figure demonstrates that similarity detection and redundancy suppression allow us to tolerate up to a 100% increase in load. Similarly, at high loads (e.g., at loads $>$ 1.0), more than a 100% reduction in the experienced delay is possible. Note that these results are based on about 50% of the images to be uploaded having similar counterparts at the server.

5.4.2 Varying the proportion of similar images available at the server

Next, we vary the fraction of uploaded images that have similar counterparts at the server. In these experiments, we use `tcpdump` to capture network traffic transferred over the wireless network. We show the results when the normalized load is 0.6; the behavioral results are similar at other loads. The results are shown in Figure 9.

First, the figure depicts a case where the server does not contain any image that is similar to any image being considered for upload. In this case, there is an overhead associated with each image due to the metadata, but this overhead serves no purpose (images are ultimately uploaded). In this extreme case, we find that the performance with our framework is only slightly worse than in the case without it; the upload data volume increases from 692 MB to 694 MB. Second, as one might expect, as the likelihood of the server finding a similar image increases (the proportion of similar images present is increased), the performance with our framework improves in terms of a drastic reduction in network load. The figure shows that when the redundancy in content is about 50%, the decrease in network traffic (because of redundancy elimination) is in fact slightly higher than 50%. The main reason for this artifact is that the reduction in network load also reduces the overheads due to retransmissions of corrupted packets that are typically incurred, if an image is in fact uploaded. These experiments also inherently account for the uplink overheads with our framework; the results suggest that these overheads are extremely low (because the gains are as expected in an ideal setting with no overhead). Finally, the metadata overhead consumed in the reverse direction (from the server to the smartphones) is also depicted; this corresponds to a very small fraction of the upload content volume ($\approx$ 3%).

We also examine the delays incurred in transferring images, while varying the proportion of similar images available to the server. The results are shown in Figure 10. Again, if no similar images are present at the server for any of the images being considered for upload, there is a very slight increase in delay (from 8.41 seconds to 9.12 seconds) due to the metadata exchange and processing. This demonstrates the extremely low overheads with our approach. As the proportion of similar images increase, drastic reductions (54%

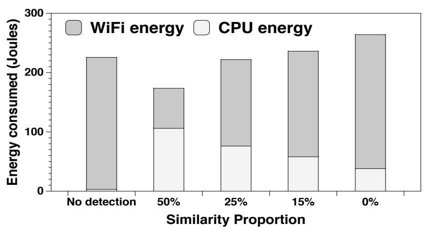


Figure 11: Impact on Energy

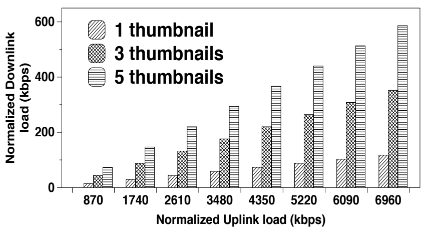


Figure 12: Overheads from thumbnails

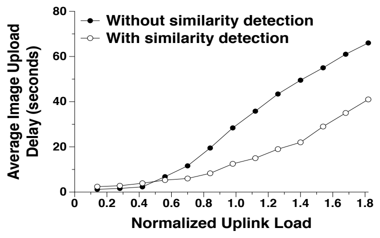


Figure 13: Upload delay with our framework (True Positive rate ≈ 70%)

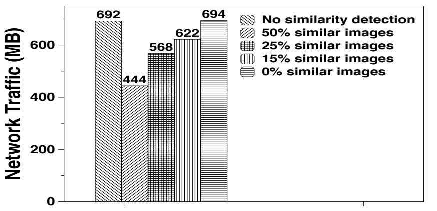


Figure 14: Uplink Network traffic with our framework (True Positive rate ≈ 70%)

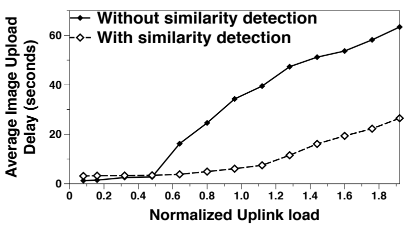


Figure 15: Delay vs load with WiFi (ns3)

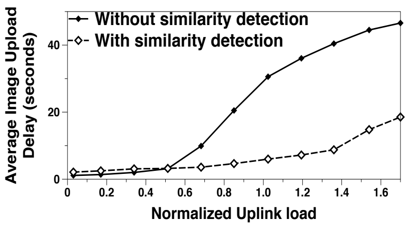


Figure 16: Delay vs load with LTE (ns3)

when this proportion is 50%) in image transfer delays are realized with our framework as depicted in the figure.

5.5 Impact on energy consumption

Our next set of experiments capture the impact of our framework on the energy consumption on the client devices. Specifically, we pay particular attention to the (i) energy consumed due to processing, towards extracting local and global features, and (ii) the energy consumed by the network interfaces due to content/metadata transfers. We compare the energy consumed on smartphones with and without our framework.

We use the PowerTutor tool [37] to capture the energy usage on our smartphones. For clarity, we only show the results with a Sony Ericsson Xperia Arc phone in our test bed. However, we observe similar results on different phones from different vendors as well. Figure 11 shows total energy consumed and the energy breakdown when a set of 50 images is uploaded from the phone. It is observed that our approach only induces a very small energy overhead on client devices when the similarity detection fails in all cases (server does not have any similar images to the ones considered for upload). As the volume of the transferred data is reduced, the energy consumed by the WiFi connection is also reduced. However, the energy consumed due to processing increases (due to the computation of global/local features of the images). The highest difference in energy consumption is between when no similarity detection is deployed and when there are no images with similar versions on the server side; this is about 40 Joules. On today's modern smartphones, the battery capacity is around 1800mAh with a voltage of 3.7 Volts; the above energy overhead only only corresponds to about 0.2% of the battery capacity. Given the large number of image transfers considered from each phone here, the overhead is likely to be even lower in practice and thus, will not adversely affect user experience.

5.6 Overhead due to thumbnails

As our final experiment on our Android testbed, we seek to quantify the overheads incurred in sending different numbers of thumbnails from the server to the smartphone clients. In this experiment, 50% of the images that are considered for uploads have similar versions on the server side. Figure 12 shows the normalized (upload) load and the corresponding overhead in terms of download load if 1, 3 and 5 thumbnails are generated for those images for which the server comes up with a negative result at the end of Phase 2, in our system. It is observed that when the upload rate is increases, the overhead of generating thumbnails also increases. However, even when the generated upload load is higher than the capacity of our link (6Mbps), the overhead of generating 5 thumbnails is still less than 10% of the link capacity. This demonstrates that Phase 3 of our framework is lightweight and is a viable option in practice.

5.7 Improvement in network performance with our framework in practice

Thus far, we have shown the improvements on network performance in an idealized settings where we assume that similarity detection can be performed with 100% accuracy. Next we show the impact on network performance with our framework, using our test set of images. Here, the true positive rate is approximately 70% (as described in Section 5.1). Figure 13 shows the reduction in upload delay if the proportion of redundant content is 50%. Unlike in an ideal case, our framework is able to eliminate ≈70% of the redundant content (given the true positive rate achieved). The figure shows that when the load is high, the upload delay without similarity detection is about ≈44% higher (even though only ≈35% of the data considered for upload gets suppressed). The reason for this higher than expected delay reduction is the same as that in the ideal case. The retransmission overheads due to corrupted packets decrease (to significant extents in cases where the link quality is poor) as compared to a case without similarity detection (when the images actually get uploaded); this in turn further reduces the aggregate network load and thus decreases delay. The elimination in redundant content also allows the network to sustain a higher load. For a target expected delay of 30 seconds, the sustainable load increases by about 60% as seen in the figure. Figure 14 shows the uplink network traffic when different amounts of redundant content are present. For the same reason as above (fewer retransmissions), the reduction in the total (uplink) traffic is typically higher than the proportion of redundant content that is suppressed except in the case where there are no similar images at all (0% of similar images). In this extreme case, all images are uploaded, and there are slight overhead penalties due to our framework.

5.8 Evaluations via simulations

Finally, we examine the impact of our framework on network performance using ns-3 based simulations, which allow us to experiment with different network set-ups and scales.

Simulation set-up: We evaluate the network performance with both WiFi and LTE: (i) In the WiFi set-up, all the mobile devices connect to the same wireless access point (AP) through a 802.11b link, which in turn connects to a server node through a dedicated link of 100 Mbps. All the the clients are initially distributed evenly in a square area of 50x50 meters; the AP is positioned at the center of the area. We use a random walk mobility model for the clients; each client moves at a random speed in a random direction inside the area. The WiFi channel is characterized by a distance based loss propagation model. The channel bit rate is kept constant (but varied in different experiments) to allow us to compute the normalized load. (ii) In our LTE set-up, all the smartphones connect to the same base station (regarded to as an enb node in ns-3) through a LTE network; the base station is connected to a LTE gateway, which in turns connects to the server node via a dedicated link of 100 Mbps. In this set-up, all the LTE clients are initially distributed evenly in a square area of 100x100 meters; the base station is also positioned at the center of this area. We again use the random walk mobility model. The LTE channel is characterized by the default Friis path loss model. Again, the channel rate is kept at a constant value (but varied across experiments) in order to be able to characterize the normalized load.

Impact on image delay time: We examine the impact of our framework with different loads and with different numbers of mobile devices when 50% of the content to be uploaded is redundant, and is correctly eliminated. In all our experiments, under the same load, we observe consistent and very similar results even if the number of devices and data rates are changed. For simplicity, we only show one sample result from our WiFi experiments and one result from our LTE experiments. Figure 15 shows the results with a 54 Mbps WiFi network and 200 smartphones. Figure 16 shows the results with a 15 Mbps LTE network and 50 smartphones. In both experiments, a new image is generated every 10 seconds. These results match those obtained from the real experiments with our 20-phone Android testbed (shown in Figure 8); more than a 100 % improvement in the average network delay is achieved at high loads (> 0.8).

6 Conclusions

In this paper, we propose a framework for detecting similar content in a distributed manner, and suppressing the transfer of such content in bandwidth constrained wireless networks. Such constraints are likely to be imposed on networks during events such as disasters. With our framework, we seek to enable the timely delivery of every unique piece (possibly critical in some cases) of information. In building our framework we tackle several challenges, primary among which is the lightweight decentralized detection of redundancies in image content. We leverage, but intelligently combine, a plurality of state-of-the-art vision algorithms in tackling this challenge. We perform both experiments on a 20-node Android smartphone testbed and ns-3 simulations to demonstrate the effectiveness of our approach in decreasing network congestion, and thereby ensuring the timely delivery of unique content.

7 Acknowledgments

This work was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

We would like to thank our shepherd, Gianluca Iannaccone, for his constructive comments and guidance through the camera-ready submission process.

8 References

[1] "More photos shared daily on Snapchat than Facebook, Instagram combined." `http://bit.ly/1sfog1M`, 2013.

[2] "Apple support communities blog." `http://bit.ly/1iy8gOQ`, 2014.

[3] T. Anthony and M. Moss, "Preparing cities for crisis communication," in *CCPR*, 2005.

[4] C. James, , A. Popescu, and T. Underwood, "Impact of Hurricane Katrina on internet infrastructure," in *Renesys*, 2005.

[5] A. Hughes, L. Palen, J. sutton, S. Liu, and S. Veweg, "Site-seeing in disaster: An examination of on-line social convergence," in *ISCRAM*, 2008.

[6] T. Preis, H. Moat, S. Bishop, P. Treleavan, and E. Stanley, "Quantifying the digital traces of Hurricane Sandy on Flickr," in *SREP*, 2013.

[7] U. Weinsberg, Q. Li, N. Taft, A. Balachandran, V. Sekar, G. Iannaccone, and S. Seshan, "CARE: Context aware redundancy elimination in challenged networks," in *ACM HotNets*, 2012.

[8] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *ICCV*, 2011.

[9] O. Chum, J. Philbin, M. Isard, and A. Zisserman, "Scalable near identical image and shot detection," in *CIVR*, 2007.

[10] Y. Zhang, Z. Jia, and T. Chen, "Image retrieval with geometry-preserving visual phrases," in *CVPR*, 2011.

[11] D. Nistér and H. Stewénius, "Scalable recognition with a vocabulary tree," in *CVPR*, 2006.

[12] F. Liu, B. Li, L. Zhong, B. Li, H. Jin, and X. Liao, "Flash crowd in P2P live streaming systems: Fundamental characteristics and design implications," in *IEEE Trans. Parallel Distrib. Syst.*, 2012.

[13] C.-H. Chi, S. Xu, F. Li, and K.-Y. Lam, "Selection policy of rescue servers based on workload characterization of flash crowd," in *SKG*, 2010.

[14] A. Koehl and H. Wang, "Surviving a search engine overload," in *WWW*, 2012.

[15] Q. Tran, K. Nguyen, K. E, and Y. S, "Tree-based disaster recovery multihop access network," in *APCC*, 2013.

[16] S. M. George, W. Zhou, H. Chenji, M. Won, Y. O. Lee, A. Pazarloglou, R. Stoleru, and T. A, "Topics in situation management distressnet: A wireless ad hoc and sensor network architecture for situation management in disaster response," in *IEEE Communications Magazine*, 2010.

[17] K. Fall, G. Iannaccone, J. Kannan, F. Silveira, and N. Taft, "A disruption-tolerant architecture for secure and efficient disaster response communications," in *ISCRAM*, 2010.

[18] Y. Zhang, Y. Wu, and G. Yang, “Droplet: A distributed solution of data deduplication.,” in *GRID*, 2012.

[19] P. Riteau, C. Morin, and T. Priol, “Shrinker: Improving live migration of virtual clusters over WANs with distributed data deduplication and content-based addressing,” in *Euro-Par*, 2011.

[20] X. Zhang, Z. Huo, J. Ma, and D. Meng, “Exploiting data deduplication to accelerate live virtual machine migration,” in *CLUSTER*, 2010.

[21] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid, “Evaluation of GIST descriptors for web-scale image search,” in *CIVR*, 2009.

[22] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. Comput. Vision*, 2004.

[23] J. Sivic and A. Zisserman, “Video Google: A text retrieval approach to object matching in videos,” in *ICCV*, 2003.

[24] O. Chum, J. Philbin, and A. Zisserman, “Near duplicate image detection: min-hash and tf-idf weighting,” in *BMVC*, 2008.

[25] Z. Wengang, L. Yijuan, L. Houqiang, S. Yibing, and T. Qi, “Spatial coding for large scale partial-duplicate web image search,” in *MM*, 2010.

[26] A. Sarkar, P. Ghosh, E. Moxley, and B. S. Manjunath, “Video fingerprinting: features for duplicate and similar video detection and query-based video retrieval,” in *SPIE*, 2008.

[27] O. Miksik and K. Mikolajczyk, “Evaluation of local detectors and descriptors for fast feature matching,” in *ICPR*, 2012.

[28] J. Heinly, E. Dunn, and J. Frahm, “Comparative evaluation of binary features,” in *ECCV*, 2012.

[29] C. Grana, D. Borghesani, M. Manfredi, and R. Cucchiara, “A fast approach for integrating ORB descriptors in the Bag of Words model,” in *IS&T/SPIE*, 2013.

[30] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[31] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, “Finding a needle in Haystack: Facebook’s photo storage,” in *OSDI*, 2010.

[32] “OpenCV library.” `http://www.opencv.org`.

[33] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in *VISSAPP*, 2009.

[34] “Image Magick.” `http://www.imagemagick.org`.

[35] “Apple iPhone 5s reviews.” `http://bit.ly/TXPtqr`, 2013.

[36] Y. Yang, H.-Y. Ha, F. Fleites, S.-C. Chen, and S. Luis, “Hierarchical disaster image classification for situation report enhancement,” in *IRI*, 2011.

[37] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, “Accurate online power estimation and automatic battery behavior based power model generation for smartphones,” in *CODES/ISSS*, 2010.