InfoNames: An Information-Based Naming Scheme for Multimedia Content

Arun Kumar
Athula Balachandran
Vyas Sekar
Aditya Akella
Srinivasan Seshan

Technical Report #1677

July 2010

InfoNames: An Information-Based Naming Scheme for Multimedia Content

Arun Kumar[1], Athula Balachandran[2], Vyas Sekar[2], Aditya Akella[1], and Srinivasan Seshan[2]

[1]University of Wisconsin-Madison

[2]Carnegie Mellon University

ABSTRACT

Recent proposals have argued for data-centric mechanisms that decouple data delivery from the sources of the data and the transfer protocols. We take this idea to its logical completion and argue for enabling content distribution schemes to name and query directly for the underlying *information*. The motivation for this information-aware design is that we see a proliferation of diverse producers of multimedia content offering varying presentation modes and significant heterogeneity in the operating conditions of Internet-enabled devices that seek access to such multimedia content.

In addition to decoupling content from available sources and transfer protocols, information-aware names, or *InfoNames*, explicitly decouple the information from content presentation. Thus, it enhances *availability* and allows users to maximally leverage sources of multimedia content offering diverse presentation formats. Further, users and providers benefit from having additional *flexibility* to dynamically adapt content delivery depending on application and network constraints.

In this report, we address the first challenge to realizing this idea - How should we *name information* in multimedia content in a way that is invariant across presentation formats and can be consistently generated/verified without relying on a centralized authority? We leverage techniques from the multimedia and computer vision communities, and propose a set of algorithms for naming, and comparing InfoNames. An extensive evaluation of the proposed schemes on a controlled dataset of images and videos is presented. In addition, an 'in-the-wild' study with a set of videos download from Youtube is also described.

1. INTRODUCTION

Multimedia content is now a dominant fraction of Internet usage. Some of the most popular Web sites today are video hosting services such as Hulu and YouTube. Similarly, Asian and American access ISPs are seeing rapid growth in Video-on-Demand (VOD) subscriptions. The goal of this project is to design mechanisms and interfaces to enable flexible retrieval of multimedia content.

Multimedia access today is characterized by three key trends that point to the need for ground-up design of retrieval mechanisms. First, modern media players are fairly sophisticated with cross-platform players (e.g., VLC) can read and render audio and video in almost any format. Second, any particular media item is available from multiple hosting locations and in many formats and resolutions. Third, there is significant heterogeneity among users' device (screen resolution, media capabilities) and network capabilities. Consequently, users are more interested in the media than in a specific source or even format, and furthermore in dynamically adapting these aspects to match their operating constraints. The host-centric model of the Internet cannot support these requirements. Even data-centric architectures [22, 21, 23] that point out that users care more about data than where it comes from, cannot provide such capabilities to flexibly select from alternate presentations (Section 3).

Our position is that flexible multimedia retrieval requires mechanisms that allow users to directly name, and query for, the underlying *information*. Information is a perceptual entity that captures the content's defining or most significant features. Multimedia objects that correspond to the same source but differ in their resolutions, frame rates, and formats carry the same information although they differ at the bit-level. Using information-aware naming and querying schemes, users can retrieve different portions of media from multiple sources in different presentation formats, thereby increasing *availability* and enabling greater *flexibility* in adapting their multimedia transfers to meet current operating conditions.

Two key mechanisms lie at the core of the design and analysis of such an information-aware framework.

1. Approaches for deriving information fingerprints or *InfoNames* for multimedia content.
2. An interface for building information-aware applications that enables clients to leverage this increased flexibility and availability.

In this report, we address the first question, viz., the design and analysis of an information-based naming scheme for multimedia content. Details about the information-aware framework for leveraging InfoNames as well as the advantages and applications of InfoNames can be found in [**?**].

Designing InfoNames is more complex than traditional hash-based naming in data-centric architectures (e.g., [19]). The key challenge lies in ensuring that InfoNames are presentation invariant and bound to the information. Like data-centric names, InfoNames must be *unique* across dissimilar content and compact relative to content size.

Our central contribution is in synthesizing ideas from the image and video processing literature (e.g., [12, 20, 3, 5, 17]) to design InfoNames for image/video content offering these properties (Section 5). Matching InfoNames to identify if the underlying contents are similar is also more involved than simple string matching. There are inherent minor variations in InfoNames across similar content from different sources, resolutions, etc. Thus, we also need suitable matching algorithms with practical parameters. We use controlled datasets to explain how the parameters for the InfoName generation and matching algorithms can be selected in practice (Section 6).

We also present a more in-depth "in-the-wild" study using a collection of videos from `YouTube`. We validate that our algorithms provide zero false positives (i.e., never mark dissimilar content as being similar or identical) and also achieve close-to-optimal performance in identifying similar content.

2. PRELIMINARIES AND PROPERTIES OF INFONAMES

Given that the intention is to identify and name the *'information'* present in the multimedia content, the questions that naturally arise are - how do we define and isolate the *information*, when do we say that two pieces of *information* are identical, how do we convert the *information* into a signature, what are the properties of that signature, and when do we say that two signatures represent the same *information*? In this section, we address some of these questions to set the stage for the discussion on the design of InfoNames.

2.1 Definitions

We introduce some informal definitions for clarity. *Information* refers to an abstract "perceptual entity" that is verifiable and consistent, i.e, invariant across different presentation formats. The *InfoName* is a perceptual signature/fingerprint that captures this *Information* and has certain properties. The *Content* refers to the collective of the *Information* along with the presentation and other 'meta-factors' that are rendered by higher layer applications. The *Data* refers to the plain raw bits as stored on machines, with no semantics attached to them. E.g. suppose we have two 'same videos' Avatar.flv and Avatar.avi. They are the same *information* but they are not the same *content*, since their formats are different. Obviously they aren't the same *data* either since they will differ at the bit level. Note that InfoNames is not meant to solve computer vision problems of identifying the same 'objects' or semantic notions of equivalence, but rather machine generated and verified signatures that capture and exploit perceptual equivalence.

2.2 Properties of InfoNames

The requirements of InfoNames have natural analogs in the data-centric world. The key difference is that InfoNames additionally require "information-binding". For completeness, we state these below:

1. **Information binding:** For two contents C_1 and C_2 that have identical "information", i.e., Information(C_1) = Information(C_2), but may differ in their presentation formats, we want InfoName(C_1) = InfoName(C_2).
2. **Decentralized operation:** Given the proliferation of content sources and producers, we want InfoName generation and verification to be decentralized and not depend on a single point-of-trust.
3. **Compactness:** InfoNames serve as "keys" to retrieve content. Thus, they must be much smaller than the actual content they represent.
4. **Uniqueness:** If Information(C_1) $\neq$ Information(C_2), then, InfoName(C_1) $\neq$ InfoName(C_2) with high probability, since, due to the compactness requirement, the InfoName cannot be comparable in size to the content itself.
5. **Ease of computation and checking:** Given some content C, it should be computationally 'easy' to generate its InfoName. Also, given an InfoName I, it should be 'easy' to check if I = InfoName(C).
6. **Integrity and pollution resistance:** If a user downloads some content and the InfoName matches that given by the information producer, we want to guarantee that the user was not served bogus/malicious content. In other words, InfoNames should be resistant to attacks where adversaries can generate fake content (e.g., "rick-rolling") with a specific InfoName.

Before we proceed to discussing how to identify information and generate InfoNames, there are some other considerations that need to be understood:

- **Who generates InfoNames?:** We think the naming architecture should be agnostic to this. The InfoName generation scheme can be used either by the original content producer (e.g. a movie studio, a home user, etc.), or independent content creators (e.g. distributors, shops, etc.) or even third-party hosting services (e.g. Youtube, Flickr, etc.)

- **What is the granularity of InfoNames?:** This is an important consideration. One could generate an 'InfoName' for the whole content or have separate InfoNames for various chunks of the content. This decision affects the identification and retrieval of the content. We provide further details of this when we discuss our InfoName scheme.

2.3 Taxonomy of Information in Multimedia

We now get to the details of what we define as 'identical information' in the specific context of images and videos. This informal taxonomy is motivated by the following important use case scenario - suppose while a user watches a video, the system in the background fetches one chunk from one source, and the next from another source, based on some conditions. The intention is that a user shouldn't perceive any 'sudden difference' in the video viewing. For example, if the next chunk is brighter, a user might be annoyed by the difference. However, the system can make lightweight changes to the video display, like altering display resolution (without altering the video), etc. We thus apply such high level insights into user perception to concretely classify variations to solve our problem.

For images, if there is any 'perceptual' difference based on the actual pixel values displayed, then they are not considered identical. In this context, if the differences are only in format and/or resolution (provided they do not degrade the perceptual 'quality' too much), they qualify as *'identical'* information. Changes to format and resolution (size) preserve the information, while changes that alter say, brightness, contrast, color tinges, grayscaling, etc. are said to alter the information, since they affect the pixel values that a person sees. We consider the latter category of alterations to have made the new image *'similar'* to the original one, but not *'identical'*. Some other kinds of changes like cropping, rotation, shear, windowboxing/letterboxing, insets, etc. are considered to have altered the image significantly to be called *'different'*. Of course, images that are not perceptually same to begin with are also considered *'different'*. Note that this definition of identity is strict enough to satisfy the use case scenarios mooted earlier.

A video can be viewed as basically being a sequence of images. But it has several other 'meta-factors' than just format and resolution. These include frame rate, aspect ratio, bit rate, etc. Also, it has audio, and possibly, sub titles. Moreover, 'format' of a video comprises of the container format for the video file as well as the codec used for data compression. Ignoring the other factors for now, we can extend the definition of image information identity to two videos by applying that definition to every frame of the videos. But that would work only if the two videos have the same number of frames, which might become too restrictive. This is because a person seeing a video will most likely not be able to discern between two frame rates. Hence, we relax the definition by extending the 'perceptual entity' to a consistent matching 'time slice' of the videos. In other words, it is necessary that any consistent time slice of two videos, which will be two images, match as per the definition of image information 'identity' for the two video to be considered *'identical'*. Else, if all such time slices match at least as per the definition of image information 'similarity', the videos can be considered *'similar'*. Later on, we relax this definition even further by changing the granularity at which we operate from a whole video to 'chunks of information' within the video. We discuss details of this change in Section 5.3.

We define *Metadata* of a video to be its format (codec), resolution and frame rate. We ignore bit rate, and assume that aspect ratio is the same as the resolution's width to height ratio. For now, we ignore sub titles and a similar taxonomy for audio, and they are subjects of future research. Of course, it is also necessary that the audios in the videos are identical and we describe what we do later in our proposed scheme.

3. STRAWMAN APPROACHES FOR GENERATING INFONAMES

The next natural question to consider is - how do we convert the information into a signature? In other words, how do we generate these InfoNames? We now discuss a few 'natural solutions' for naming information and highlight their limitations.

3.1 Content creation timestamps

We can ensure uniqueness and compactness by specifying when/where the content was created; e.g., timestamp and MAC address. However, this fails the ease of checking and decentralized requirements–we need a global database mapping "information" to these. Further, it does not provide information-binding because the same information produced by two sources will be different.

3.2 Unique "filenames"

Several naming architectures extend the filename interface through a heirarchy [1]. For example, we can imagine using the IMDB title-id as an InfoName for movies. Unfortunately, this requires a logically centralized entity to guarantee uniqueness and for verifying the name. So, cross-producer availability of content cannot be exploited by this. Also, such an approach would burden the content producer to manually identify and 'name' the file to meet the information binding requirement, which is nearly impossible to achieve.

3.3 Extending Data-Centric Names

One approach to retro-fit information-binding into data-centric names is as follows. We normalize all content to a baseline representation (e.g. images in 100x100 resolution and BMP format), and use the cryptographic hash of this representation. This appears as a promising alternative at first. However, this notion of information-binding is at the same time both restrictive and insufficient. We used a small image dataset and ran different transforms to try this Strawman solution using SHA-1[1] on the baseline (Table 1). We see that any lossy transformation unidentifiably alters the name. Thus, it reduces availability for clients who are flexible to lossy formats and slight variations in content quality. Also, this only ensures that the content is similar at the baseline representation (i.e., weak integrity assurance).

Image Source	Transform			
	Lossless format (e.g., bmp)	Lossy format (e.g., jpg)	Resolution	Aspect Ratio
Lossless (bmp)	√	×	×	×
Lossy (jpg)	√	×	×	×

Table 1: Evaluating the strawman extension of data-centric naming solutions across different transforms.

3.4 Layering Information-Awareness on Data-Centric Names

Another solution is to simply use *keywords*. We use a search infrastructure (e.g., Google) to bind keywords to content, possibly using the very algorithms we describe in the next section. Keyword searches return the matching data-centric names. Users select one of the search results and use data-centric retrieval. This provides limited late-binding in that it delays committing to a specific data source, but not to the presentation format/quality. At the same time, the search results provided will only be as fresh/relevant as the frequency with which the content is being indexed. Further, Table 1 shows that data-centric names are not robust across lossy representations, which is typical of multimedia content. Thus, this search infrastructure needs to explicitly account for all possible combinations of formats, codecs, and encoders (e.g., different MPEG generation tools can result in different data representation for same video quality). Finally, this does not satisfy the decentralized operation requirement. Thus, the scope of applications it can enable is limited; e.g., this would not be useful in a personal computing scenario.

The key property of data-centric names is that the name (the cryptographic hash) is directly tied to the data that it names. In the same vein, we believe that InfoNames should be based on approaches that tie in to the "information" that the content represents. The above discussion shows the limitations of trying to retrofit information-awareness into existing naming solutions. Rather than come up with such incremental and ad hoc solutions, we want to start from first principles and ask a fundamental question: *how can we name what we really want, the information?*

[1]We could alternatively use locality sensitive hashing techniques [13]. Using a domain-specific hashing algorithm only confirms the need for information-aware naming!

4. A MULTIMEDIA/VISION APPROACH TO INFONAMES

There exists a rich body of work in the Multimedia and Computer Vision worlds that solve the problems of detecting duplicates, copyright violations, song identification etc [6]. Most of them assume that the content themselves are available, but some of them extract a 'signature' out of the content and compare based on that signature.

The first category includes database-based approaches for copyright detection, wherein the scheme identifies if a given video content 'matches' any in a set of existing contents in the video database. It is believed that Youtube uses one such scheme [9]. However, this is not applicable to our problem due to two reasons. Firstly, we do not compare information/content but rather InfoNames. Of course, one can argue that the 'information' can be used as such for a 'name' while comparing, but that will be very inefficient and unnecessary if one wants to do the comparison remotely. E.g. if we want to check if two images are identical, we could simply compare their raw RGB pixel values. But raw pixel values cannot qualify as a 'name'. Similarly, for comparing two videos, we could decode them to sequence of images and compare them. Again, such a 'name' will be very clumsy and several folds larger than the data itself. Morevoer, the intermediate data structures that occur in such approaches where contents are directly compared are generally too huge and detailed to qualify as InfoNames. Secondly, these approaches are tailored to adhere to a much looser definition of 'similarity', since their aim is to capture any kind of transformations to a piece of content. E.g, these schemes want to capture alterations like windowboxing, cropping, color changes, rotations etc., and not just resolution, frame rate, etc., since the 'object' in such transformed contents is still the same. But as per our taxonomy, which is stricter, such alterations are considered to have altered the information itself.

The second category includes a number of commerical (proprietary) and a few open-source schemes that convert the 'perceptual entity' in the content to a signature [5]. We explored a few such schemes as candidates [17, 16, 5] for our InfoNames but realized that each of them has specific issues which makes them inapplicable for our problem directly as stand alone schemes. We discuss them in the context of images to start with.

4.1 Spatial Moments

The first family of approaches utilizes edge histograms or spatial moments [16] in the image to get a signature. For instance, GIST [17] converts an image into a sequence of 512 floats, which is its signature. The distance between two signatures is their L1-norm. But we found that it cannot discern between a color and a grayscale image, which is bad in case color is used to convey 'information' (e.g. many national flags might appear identical in grayscale). We also learnt that the general aim of such signature schemes is 'object recognition' in terms of defining features, and not 'perceptual naming' which we want [17]. Moreover this approach is not robust to resolution changes which can affect the spatial moments in the image. Hence it fails on both uniqueness and information binding.

4.2 Color Histograms

Another family of approaches utilizes color histograms obtained from the image and uses that to compare. Obviously, the flaw with this is that it will not be unique enough. E.g. consider a white-black strip will be 'named the same as a black-white strip. There do exist applications that use this as one of its components. E.g. it is believed that Google Similar Images uses a variant of one such scheme. However, color histogram alone as an InfoName will violate uniqueness, another key property.

4.3 Frequency Components

The last, and most important family of approaches use the frequency components of an image signal to represent it. Mostly, they use the Discrete Cosine Transform [10] which converts the pixel values to frequency values. One such scheme is [5], which is based on a DCT-based algorithm proposed in [12]. The key idea is that the lower frequencies represent the 'perceptual entity' of the image at a high-level since the high frequencies represent fast variations in the image, which the human eye is not good at capturing. This is also the basis for the JPEG compression standard [2]. The scheme here outputs a 64 bit number as the image's signature. However it doesn't directly qualify as our InfoName. This is because it doesn't satisfy the taxonomy as it considers all 'similar' images as per our definition to be 'identical' to the original image. Also, it is not robust enough to discern low 'quality' images as it discards higher frequencies. Hence this also fails on uniqueness but not as badly as the other approaches since it can tell apart 'different' images from 'similar' and 'identical' ones. Also, it satisfies all the other properties for an InfoName fully. Hence, we decided that this is the most promising start for an InfoName.

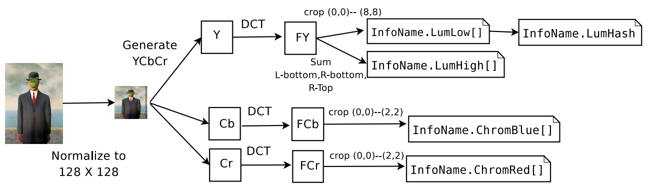


Figure 1: InfoName generation algorithm for images. We scale down the image to a baseline resolution and then convert into its YCbCr representation. Then for each of these components, we generate compact summaries to create the image InfoName.

5. OUR INFONAME SCHEME FOR IMAGES AND VIDEOS

We build on existing techniques from the multimedia community and synthesize suitable algorithms to solve our problem. We then evaluate their feasibility and robustness on real-world datasets. We acknowledge that there might be other candidate InfoName generation algorithms; it is not possible to exhaustively enumerate and evaluate these. Our goal is not to discover an "optimal" InfoName. Rather, we seek to demonstrate a practical and feasible design that satisfies the properties listed out earlier. First, we discuss how we generate InfoNames for images and then describe how we generate InfoNames for videos.

5.1 InfoName for Images

Figure 1 gives an overview of how we generate the InfoName for an image. First, we scale the image to a baseline resolution of 128×128. We chose this resolution because this is lower than most commonly used image/video resolutions, but high enough to allow us to discern detailed structures. (It is square only because it is convenient to apply DCT on square matrices; this is not strictly necessary.) In the second step, we convert this scaled image to the YCbCr representation [8]. The Y value, the *luma* component, corresponds to the image in grayscale. Cb and Cr are *chroma* components that capture the blue and red distributions. (We do not need a green component because that is linearly dependent on Y, Cb, and Cr). The rationale for choosing the YCbCr representation instead of the more conventional RGB representation is as follows. Doing a DCT on the R, G, and B matrices scatters the high energy coefficients in the frequency domain. Thus, we require more coefficients to capture a sufficient amount of signal energy. But with the YCbCr representation, almost all the energy is concentrated in the first few low frequency coefficients in the case of Cb and Cr. Hence we can get a more compact summary for the InfoName.

Starting from the DCT coefficients FY, FCb, and FCr, we create the InfoName with five components: (LumLow, LumHash, LumHigh, ChromBlue, ChromRed). Intuitively, LumLow and LumHash provide a coarse-grained fingerprint, while LumHigh, ChromBlue, and ChromRed capture more fine-grained properties. We describe these below:

LumLow: We extract the lower end 9×9 submatrix of FY to get the LumLow component of the InfoName. The low-frequency components capture most of the signal energy. Instead of a fixed 9×9 matrix, we could pick the smallest submatrix that captures some fraction (say 95%) of the energy. However, the size of such a submatrix varies significantly in practice. This would result in variable length complicating the match process. Later on, we will show an evaluation of the variation of energy with the length across a large image corpus and justify why this length is justified.

LumHash: Additionally, we use a 64-bit summarization of LumLow [12]. We start with LumLow and get the rank-ordered median of its coefficients. Then, we "quantize" each coefficient to 0 or 1 indicating whether it is higher or lower than the median. (We need not explicitly add this field because it can be computed from LumLow. We add it because it provides a simple check to speed up the matching process.)

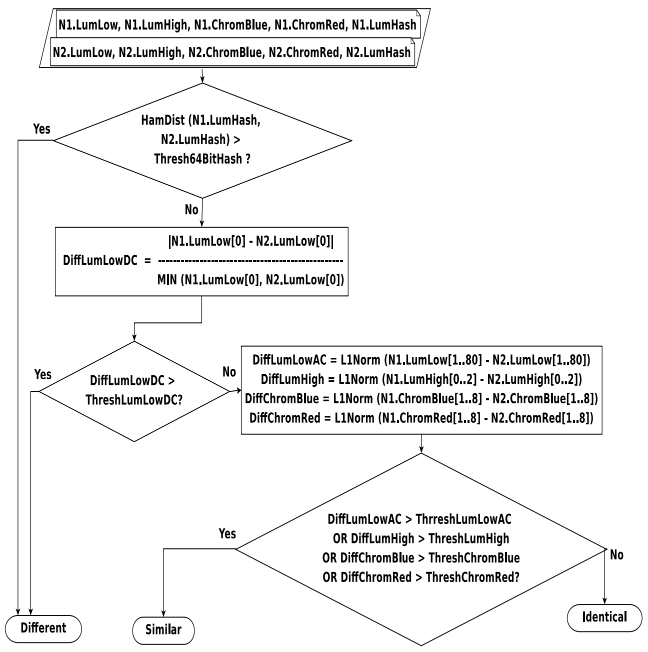
LumHigh: However, discarding the higher frequency components of FY loses the ability to distinguish high and low quality images. Thus, we use LumHigh to summarize the remaining coefficients of FY. We do this by computing the sum of the absolute values of the lower left, lower right, and upper right square submatrices of FY.

ChromBlue and ChromRed: These are the lower end 3×3 submatrices of the FCb and FCr. We choose a smaller slice for these (compared to LumLow) because we saw that most of the signal energy in these lies in the first 2-3 frequencies.

5.2 Matching Image InfoNames

Figure 2 shows how we match two image InfoNames. As discussed earlier, LumHash lets us immediately distinguish two different images. If two InfoNames' LumHashes do not match, the images are likely very different. We compute the Hamming distance between the two 64-bit values and check if it exceeds a threshold.

If the LumHash fields match, we match the other components. Each of these can be treated as a vector. Thus, we use the L1-norm of the difference between two vectors as the distance metric. We use component-wise thresholds for each of these L1-norm differences to check if the two InfoNames match. However, we check the DC coefficient (zero-th frequency) of LumLow separately. (The DC coefficients are typically much larger than the other coefficients and hence can skew the L1-norm computation.) We currently omit the DC component of ChromBlue and ChromRed while matching, because this does not add much value in terms of distinguishing images. However, from a completeness perspective (and for better integrity), one can check these two for a match too.



N1.LumLow, N1.LumHigh, N1.ChromBlue, N1.ChromRed, N1.LumHash
N2.LumLow, N2.LumHigh, N2.ChromBlue, N2.ChromRed, N2.LumHash
HamDist (N1.LumHash, N2.LumHash) > Thresh64BitHash ?
Yes
No
DiffLumLowDC = |N1.LumLow[0] - N2.LumLow[0]| / MIN (N1.LumLow[0], N2.LumLow[0])
DiffLumLowDC > ThreshLumLowDC?
Yes
No
DiffLumLowAC = L1Norm (N1.LumLow[1..80] - N2.LumLow[1..80])
DiffLumHigh = L1Norm (N1.LumHigh[0..2] - N2.LumHigh[0..2])
DiffChromBlue = L1Norm (N1.ChromBlue[1..8] - N2.ChromBlue[1..8])
DiffChromRed = L1Norm (N1.ChromRed[1..8] - N2.ChromRed[1..8])
DiffLumLowAC > ThrreshLumLowAC OR DiffLumHigh > ThreshLumHigh OR DiffChromBlue > ThreshChromBlue OR DiffChromRed > ThreshChromRed?
Yes
No
Different
Similar
Identical

Figure 2: Matching two image InfoNames

If these differences are small (based on some preset thresholds), we declare that the images are *identical*; otherwise, we classify them as being *similar*. We explain our thresholds later during the evaluation.

5.3 InfoName for Video

Next, we discuss how we can extend the image InfoName to video content. A video can essentially be viewed as a sequence of images or *frames*. The naive solution would be to simply concatenate InfoName for each frame and use it as the video InfoName. This naive solution has two drawbacks. Firstly, the InfoName becomes very large, possibly comparable to the size of the video itself, thus, violating the compactness requirement. Secondly, image level processing and generation of the InfoName for each frame is computationally expensive, thus violating the ease of computation requirement.

Also, it is well known that videos have significant redundancy across frames, i.e., successive frames do not carry much "new" information. This is exploited in video encoding schemes like MPEG [4]. Thus, a natural next step is to exploit this temporal redundancy for the video InfoName. One could use a sampling-based approach, e.g., the 3-D DCT scheme proposed by Sankur et al [12] deterministically samples 1-in-k frames and applies a low-pass filter on them. However, these approaches are inherently sensitive to minor variations in timing and frame rates.

The alternative is to choose 'distinct' frames, logically 'chunk' the video using these as boundary markers, and process only this subset. Similar to robust data-aware chunking [14, 18], we believe that chunking should be information-aware. That is, rather than imposing artificial chunking boundaries (e.g., bytes or time durations), we want the boundaries to be derived from the information itself. Such information-aware chunking will be robust to time-shifts (e.g., credits missing), minor edits, and other effects (e.g., embedded content, mashups).

There is a rich literature on scene/shot detection [11] that can serve this purpose. Basically, these identify *keyframes*, where the image changes significantly (measured in terms of a suitable feature). One of the most common features is the color histogram [15]. However, we found that this has two drawbacks. Firstly, it generates inconsistent chunk boundaries across many tranforms (e.g., format, resolution and quality changes). Secondly, it requires expensive per-frame processing, which extremely slow for high-resolution videos.

In order to obtain *consistent* chunk boundaries, we need to choose image features that are invariant across common transforms. Thus, we consider two 'representative' features of the image InfoName itself - LumLow and LumHash as candidates. (Recall that LumHigh, ChromBlue, and ChromRed only capture fine-grained variations.) We analyzed how these features varied across frames within a video for a controlled dataset. We realized that LumHash was too sensitive to tranforms and also created keyframes even when scenes did not change. That is, we could not choose an appropriate threshold on LumHash for generating consistent chunk boundaries. In contrast, we found that choosing chunk boundaries by calculating the variation in the DC (zero frequency) component of LumLow yielded more consistent boundaries across transforms. We compute the *relative distance* between two successive frames i and $i + 1$ thus:

$$DistSeq(i) = \frac{|LumLow[0]_{i+1} - LumLow[0]_i|}{\min(LumLow[0]_{i+1}, LumLow[0]_i)}$$

and check if this value crosses a threshold $ChunkThresh$. (We take the relative distance since it is more meaningful than the absolute value.) Note that we can get $LumLow[0]$ directly from the YCbCr representation–this is simply the amplitude of the signal; we do not need to compute the entire per-frame DCTs. Hence it is a very 'lightweight' feature. As an illustration, Figure **??** plots the DistSeq of a video and a transform of its, where the format is altered.

Additionally, in the chunking process, we impose a minimum chunk size of 0.5 seconds. This ensures that the chunk

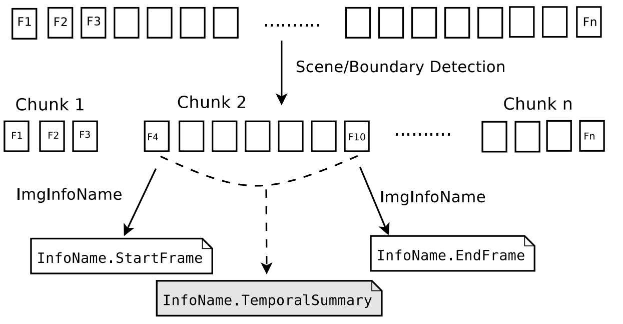


Figure 3: InfoName generation for videos. This builds on the image InfoName algorithm. We start with the YCbCr representation for each frame and chunk the video. Then for each chunk, we compute the image InfoNames of the start and end frames. We keep a summary of the variation within this chunk for integrity checks.

sizes are practical, i.e., the query/lookup overhead are low compared to the time to transfer the chunk. In doing so, we tradeoff a slight decrease in availability, e.g., neighboring boundaries might become ambiguous. In the next section, we explain our choice of $ChunkThresh$ and also show the distribution of chunk sizes in a large real-world set of videos.

Figure 3 summarizes how we generate InfoNames for a video. We start with the YCbCr representation for each frame. Then, we detect chunk boundaries using the DistSeq values. Then for each chunk, we compute the image InfoName of its first and last frames. Additionally, we include a compact summary of the other frames within this chunk.[2] Thus, the InfoName for each chunk has three components: I_{start}, I_{end}, and $ChunkSummary$. The last field serves as an additional integrity check in adversarial settings; e.g., inserting bogus frames within the chunk or randomly dropping some frames (see Section 6.5). For some application scenarios in Section **??**, we can ignore this component; e.g., synchronizing trusted personal devices. We can leverage any lightweight image InfoName feature to serve this purpose. Again, the candidates are the DC of LumLow and the LumHash. We evaluated both features on a controlled video data set to determine which would be better. Our current algorithm uses the LumHashes of each frame in the chunk. Later on, while discussing the evaluation of these algorithms, we explain why we picked this. We also include two auxiliary fields: T, the duration AH, the InfoName for the audio track (see Section 5.5). These serve as additional integrity checks and also help speed up the matching process described next.

[2]This makes the InfoName proportional to the video length. Note, however, that the InfoNames are the same size for videos with the same duration and frame rate.

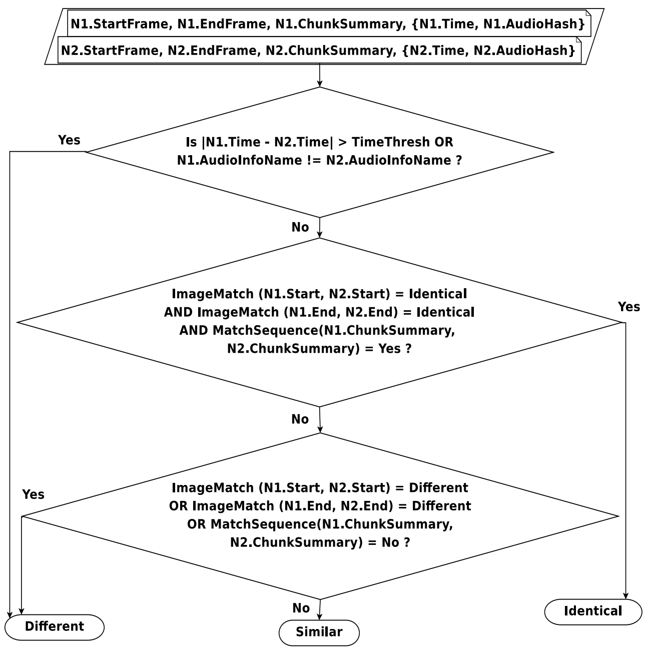

N1.StartFrame, N1.EndFrame, N1.ChunkSummary, {N1.Time, N1.AudioHash}
N2.StartFrame, N2.EndFrame, N2.ChunkSummary, {N2.Time, N2.AudioHash}
Is |N1.Time - N2.Time| > TimeThresh OR N1.AudioInfoName != N2.AudioInfoName ?
Yes
No
ImageMatch (N1.Start, N2.Start) = Identical AND ImageMatch (N1.End, N2.End) = Identical AND MatchSequence(N1.ChunkSummary, N2.ChunkSummary) = Yes ?
Yes
No
ImageMatch (N1.Start, N2.Start) = Different OR ImageMatch (N1.End, N2.End) = Different OR MatchSequence(N1.ChunkSummary, N2.ChunkSummary) = No ?
Yes
No
Different
Similar
Identical

Figure 4: Matching two video InfoNames

5.4 Matching video InfoNames:

Figure 4 describes how we match two video InfoNames. First, we mark two chunks as different if either the time difference is too high or the audio InfoNames do not match. In practice, we want the time difference threshold to be much smaller than the minimum chunk size; we currently set it to 0.25 seconds. Then, we run the image InfoName match algorithm on the start and end frames. If these match (either similar/identical), we proceed to the next step. We next describe how we match the sequence of LumHashes values in the InfoName.

Now, we need to match the sequences of LumHashes. We use the following heuristic. The high-level idea is to map points from the longer sequence into the shorter sequence. Consider two sequences, $|S_1| = l_1$, and $|S_2| = l_2$, with $l_1 > l_2$. We map the i^{th} index in S_1 to a very small window around the $j = i * \frac{l_2}{l_1}$-th index in S_2. The window basically provides some tolerance while matching videos of different frame rates; we currently set it at 3. We take the pairwise Hamming distance between that LumHash in S_1 to each LumHash of S_2 in the window, and take the minimum among these. Repeating this for every LumHash in S_1, we get a sequence D of Hamming distances. If the sequences are the same length, we do a one-to-one mapping, rather than use a tolerance window to get D. The idea is that if the videos do indeed have identical information, these distances should all be low. Hence, we check if the maximum value across these, the MaxLumHash, is greater than a threshold.

5.5 InfoName for Audio

To create InfoNames for audio content, we currently leverage an off-the-shelf fingerprinting algorithm [3]. This scheme first normalizes the audio data into a common format (e.g., mono, 8000Hz sampled) and generates the frequency domain representation for each frame (roughly 1 sec) of audio. The frequency spectrum for each frame is split into Bark bands (related to human hearing) and a linear regression fit is computed for the power spectra of each band. The coefficients in the linear regressions for the various bands for different audio frames (per second) are packed into a 424-byte fingerprint for each audio chunk. Our experiments with a personal music collection showed that this fingerprint was robust across different types of transforms (e.g., format, quality) (not shown).

The only issue here is ensuring that the audio chunk boundaries coincide with the video chunk boundaries, when we compute the AH component of the video InfoName. Note that this is necessary only in a streaming context. For bulk-transfer applications, we can download the video and audio tracks separately, and merge them during playback.

6. EVALUATING INFONAME ALGORITHMS

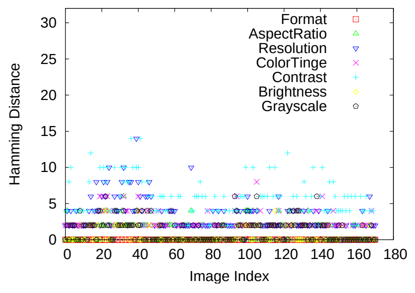
Here, we present a set of experiments designed to configure and evaluate the InfoNames. Also, a measurement study is done on videos collected from the wild.

6.1 Configuring Image InfoNames

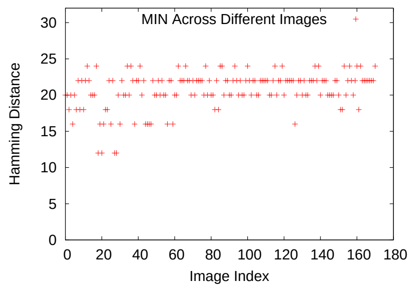
First, we address the following question:
Can we configure suitable thresholds to distinguish identical, similar, and different images?

Ideally, we want a scheme with zero false positives (i.e., never identifies two distinct contents as being similar) and zero false negatives (i.e., may mark two images that are similar as different). In practice though, this might not always be possible. Thus it might be okay to suffer a smal loss in availability (increase false negatives) in favor of guaranteeing correctness (no false positives).

For this study, we used the Univ. of Washington image dataset [7]. This contains roughly 150 images with diverse real-world scenes (e.g., nature, people, events, plants etc.) in JPEG format. For each image, we applied the following transforms: (1) changing format (to BMP,PNG), (2) changing resolution (scaled to one-third), (3) changing the aspect ratio (converting to 1:1), (4) increasing brightness, (5) altering the color by increasing the red-component, (6) increasing contrast, and (6) converting the image to grayscale. We consider format, resolution and aspect ratio changes as identity - preserving transforms, and the remaining as similarity - preserving transforms.



(a) Across transforms

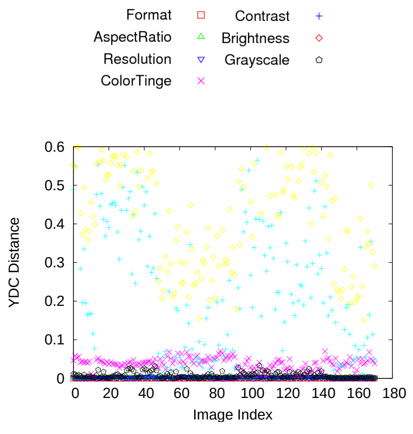


(b) Across distinct images

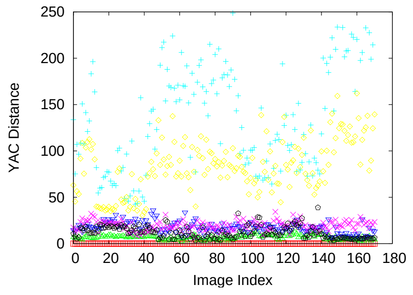
Figure 5: Comparing the Hamming distances of the LumHash field across transformed versions of images and distinct images

Our first goal is to determine the threshold for LumHash to distinguish completely different images. Figure 5(a) plots the Hamming distances of the LumHash across transformed versions of the images. Figure 5(b) does the same for every pair of different images, taking the *minimum* across all other images. We can visually verify a clear boundary separating the transformed versions from the different images. We pick $ThreshLumHash = 11$. This minimizes the false negative rate (i.e., maximizes availability) while giving a zero false positive rate. This ensures that we can check if two images are different or similar/identical. Next, we proceed to more fine-grained classification of similar vs. identical images.

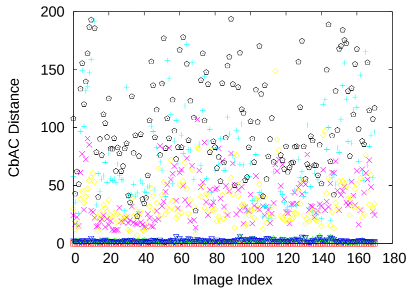
Next, we want to pick a set of thresholds across the remaining InfoName components to differentiate similar vs. identical images. The LumLow0, LumLow, ChromBlue and ChromRed components are the features useful for this and they constitute the feature vector for the threshold estimation process. Figure 6 shows the plots of the features for the various transforms mentioned. The LumHigh component is used for integrity checks, as explained later. Note that the goal for the threshold estimation algorithm is to minimize false negatives (i.e., classifying a identity-preserving transform as similar), subject to zero false positives (i.e., classifying a



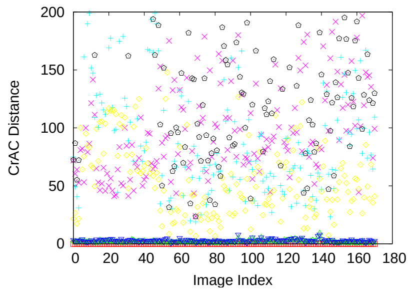
(b) LumLow 0



(c) LumLow



(d) ChromBlue



(e) ChromRed

Figure 6: Finer-grained components of the image InfoName that help distinguish between similar and identical images. For our discussion, we treat format, resolution, and aspect ratio changes as identity-preserving and the remaining as only similarity-preserving.

similarity transform as identical). We employed an iterative exhaustive search on the four dimensional search space of feature vector with appropriate lower bounds (zero) and upper bounds (the maximum value on each individual feature). At each point, the false positive and false negative rates were computed. Surprisingly, this process yielded a significantly large 'perfect' solution region, i.e., a space where the false positive and false negative rates were both zero. Hence, the usual technique of subset division and cross-validation across the subsets doesn't make sense here. We thus, picked a point in the approximate center of this perfect solution region, and the specific threshold are - $\mathit{ThreshLumLow0} = 0.06$, $\mathit{ThreshLumLow} = 45$, and $\mathit{ThreshChromBlue} = \mathit{ThreshChromRed} = 15$.

Validation: We also validated our choice of thresholds on a completely different set of 250 images from the same corpus. We apply the same transforms as before and used the above thresholds in the matching algorithm. The false-positive rate was zero (i.e., dissimilar images were clearly marked as such), and the false-negative rate was 1.3%.

6.2 Variation of Energy Captured

Next, we ascertain if the length chosen for LumLow is justified. For this, we analyze the variation of the captured signal energy. The signal energy is defined as the squared magnitude the frequency vector (the full DCT of the image). The idea is that, the more signal energy we capture, the more integrity we have in the name. A smaller LumLow submatrix captures lower energy. The larger the LumLow, the greater will be the strictness of the match, improving the accuracy. But this also makes the InfoName less compact. Moreover, LumHigh summarizes the higher frequencies, which means that too many terms in LumLow is perhaps not that useful. To capture this tradeoff, we plot the fraction of signal energy captured against the length of the LumLow matrix, based on the same image data set as before. We also plot the fractional energy captured per coefficient to reflect the 'usefulness' of each coefficient. Note that the size of LumLow increases quadratically with length. Figure 7 shows these two graphs.

The plot shows that roughly 80% of the signal energy resides in the DC coefficient alone. Subsequently, it increases slowly with the length. The energy per coefficient drops drastically and then flattens with a long tail. For our chosen length of 9, the energy per coefficient is slightly above 1% and the fraction of signal energy captured is about 89%. Thus, our chosen length seems to be a reasonable operating point, since, beyond this, more coefficients do not provide much utility - the curve reaches 90% only at length 14, with energy per coefficient at 0.4%. However, it might also be okay to choose a lower length, say, 4 or 5, which happen to be at the 'knee of the curve'. This would make the InfoName more compact, but it might also reduce the accuracy of the

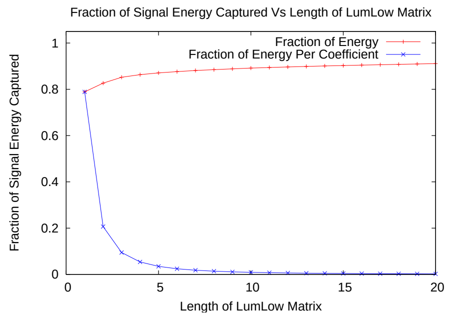


Figure 7: Variation of captured signal energy with length of LumLow matrix

match, though it is tricky to quantify this. We leave further analysis of this for future work.

6.3 Video InfoNames: Controlled DataSet

We use a controlled dataset of 50 videos (trailers from YouTube) and apply two transforms (changing format and bitrate). Using this setup, we answer the following questions:

- What is a suitable threshold for the chunk boundary detection?
- Is our chunking mechanism consistent across different transforms?

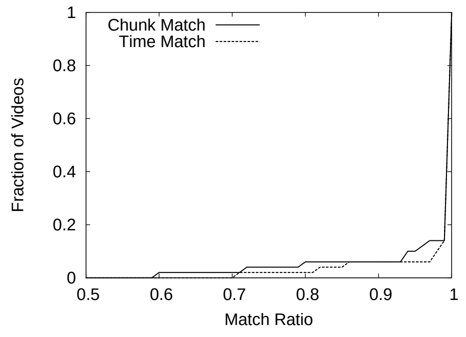
Threshold	Mean chunk size	25%le match rate
0.2	3.1	0.90
0.5	5.9	0.82
1.0	10	0.75

Table 2: Tradeoffs in chunking threshold

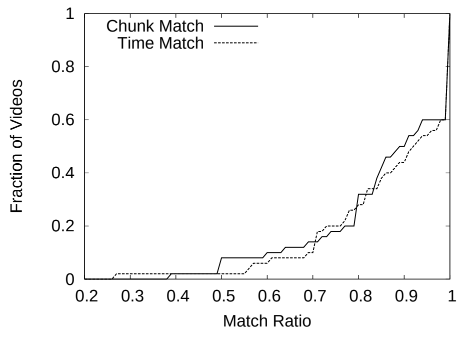
Selecting the Chunking Threshold: There are three key factors in choosing a suitable threshold for determining chunk boundaries: chunk size, InfoName generation time, and the effective match rate. Smaller chunks provide higher match rates in detecting the same content across transforms, but increase generation time as we need to process more frames to generate their image InfoNames. Larger chunks have low generation time, but are more likely to result in misses if there are slight variations within the chunk. At the same time, we do not want the chunk size to be too small, as the query/lookup overhead per-chunk will become impractical for applications. Table 2 shows three candidate thresholds for the $\mathit{ChunkThresh}$ from Section 5. We pick $\mathit{ChunkThresh} = 0.5$ to balance this tradeoff. In the next section, we confirm that this choice yields practical chunk sizes on a larger real-world dataset.

Are chunks consistent? For each video, we measure the *match ratio* in terms of the number of chunks and total duration of match between the source and tranformed version.

For brevity, we only show the results with the intra-chunk MaxLumHash check for integrity disabled because it had a negligible effect on the match rates in the controlled setting. Here, we count only *identical* matches (i.e., the last row in Table 3). Figure 8(a) shows the CDF of the match ratio when we change the presentation format (from `flv` to `avi`). We see that format changes have minimal impact; the match rate is $\geq 95\%$ for more than 95% of the videos, both in terms of time and number of chunks. Similarly, in Figure 8(b), the match rates exceed 80% (time and # chunks), more than 80% of the videos for the quality change case.



(a) Format change



(b) Bitrate change

Figure 8: Distribution of match ratio (time/# chunks) w.r.t transformed versions of the video

For completeness, we give a breakdown across different scenarios that can occur in the video matching process in Table 3. At a high-level, there are three categories where two chunks can be different, similar, or identical. They can can be different if either the time length or image matches for the start/end frames fail. We mark two chunks as similar if the image matches for one of start/end pair marks them as similar (refer Figure 4). If we relax the notion of match from identical to identical or similar, we get a marginal (7%) improvement in the match rate for the bitrate change case. Interestingly, the dominant sources of misses is when the start and end frames are marked as different. Intrigued by this, we analyzed the specific frames where this occurred. We noticed that many of the LumLow) values are quite close to 0. (This typically occurs for monochromatic images; e.g., blank screens.) In this case, the LumHash becomes less stable and the Hamming distances between similar frames becomes high. In fact, the original proposal for LumHash [12] also considers this as a corner case of this quantization procedure. We believe that it is easy to treat this effect as a special case, but leave it for future work.

Category	Result	Bitrate	Format
Time difference	Different	0.012	0.004
Start/End frames differ	Different	0.052	0.013
Start/End similar	Similar	0.008	0.004
Start/End identical/similar	Similar	0.076	0.001
Start/End identical	Identical	0.848	0.981

Table 3: Breakdown of the different categories (in terms of chunk match ratio) in the video InfoName matching algorithm using the controlled dataset.

To further understand how chunking affects the match rate, we disable chunking and do a per-frame match. We found that the chunk-level and frame-level match ratios are very close (less than 0.5% difference in total match rate). This augurs well for our measurement study in the next section. That dataset is much larger and running the frame-level analysis is expensive, especially for high-resolution videos. Because the chunk-level match rates closely follow the frame-level match rates, we are not biasing our content similarity estimates because of chunking.

6.4 Video InfoNames: Measurement Study

So far, we evaluated our algorithms on controlled datasets. Next, we analyze how our video InfoName algorithm performs "in-the-wild". We start with a collection of 80 distinct URLs of movie trailers from `YouTube`. For each URL, we download the video it serves and the top-20 *related* videos listed on this page. Our goal is to see if there is reasonable overlap in similar/identical content among related videos. This gives a rough bound on the availability of this content– we are ignoring other versions of the video and its segments within Youtube (that did not appear in the related list) and on other video hosting services. As in our previous setup, we only report the numbers with the MaxLumHash check disabled for brevity since it did not affect the results significantly.

Are the observed chunk sizes practical and useful? Figure 9 shows the distribution of the chunk sizes, measured in terms of seconds across the different videos in the YouTube dataset using $\mathit{ChunkThresh} = 0.5$. We see that the median chunk duration is 2 seconds and the 90%ile value is around 10 seconds. Assuming a streaming rate of 400Kbps (typical low quality stream in YouTube), these translate to 100KB and 500KB respectively. As a point of comparison, the typical chunk sizes recommended for data-centric applications is 128-512KB [], to amortize per-chunk overheads. This result confirms that chunk sizes in an information-centric setting are practical for common application contexts.

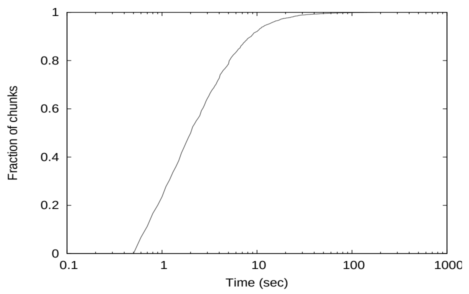


Figure 9: Observed distribution of chunk sizes. We measure chunk size in terms of time to normalize across different resolutions and bitrates.

Detecting identical/similar content: Next, we look at how much repeated content we find in our dataset. For this, we compute pair-wise chunk matches within the videos downloaded from the same URL. For each chunk, we count the number of other chunks that match it (either identical or similar). Figure 10 shows the distribution of number of matches per chunk, for identical, similar, and identical+similar matches. We notice that relaxing the definition to include similarity matches boosts the match ratio. (Unfortunately, we do not have ground truth to verify if these were different versions of the same content.) We also see significant diversity in the distribution; many of the chunks have zero matching chunks, while a few have more than 10 matches. We noticed that some URLS had high-levels of content similarity, while others had close to zero similar content in the related videos. One possible avenue for future work is to correlate the match rates with content popularity; we hypothesize that the match rates will be better if we target popular content.

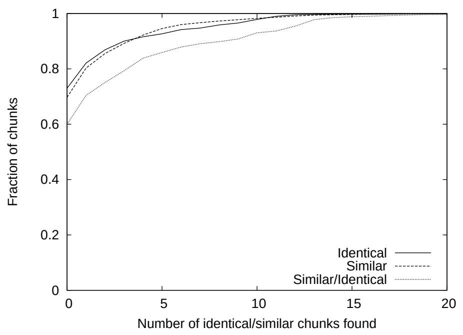


Figure 10: Distribution of number of identical/similar chunks observed per chunk

Comparison to data-centric naming: A natural question is if the overlap identified using InfoNames would have been detected by data-centric schemes. We compare the pair-wise chunk match ratio from the above analysis to a data-centric naming solution []. We show this result for a subset of 20 URLs from the larger dataset in Figure 11. Each point on the x-axis represents one such URL. For each URL, we summarize the distribution of the number of identical chunks detected using InfoNames and data-centric names. In the InfoNames case, we restrict to identical matches for a fair comparison. We use a box-and-whiskers plot to summarize this match distribution: each box shows the median, 25%ile, and 75%ile of the number of identical chunks found, and the whiskers show the complete distribution (excluding outliers). The plot also marks the outliers in each case. As discussed earlier, many URLs have a median match rate of zero both in the InfoNames and data-centric case.

We make two main observations. First, InfoNames is *strictly better* than the data-centric approach in each URL. (We confirmed this for the remaining 60 URLs as well but do not show this for brevity.) Second, in many cases, the content overlap can *only* be exploited using InfoNames. That is, the median value for the data-centric is zero, but much higher for the InfoNames case.

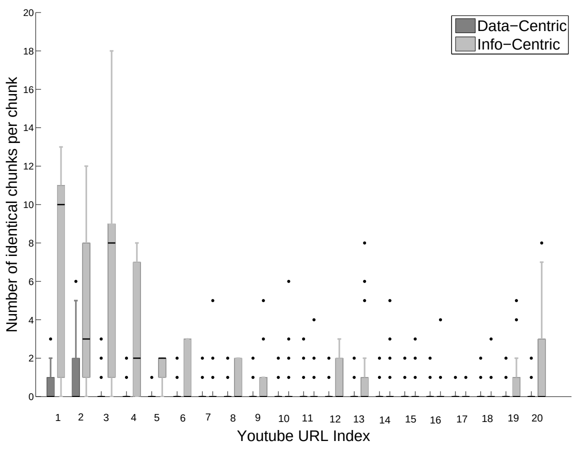


Figure 11: Comparing chunk match ratios of data-centric and information-centric naming mechanisms.

6.5 InfoName Integrity

Next, we study the robustness of our image and video InfoNames against a spectrum of integrity attacks and explain if/how the specific components of the InfoName can guard against these. A rigorous understanding of information integrity is outside the scope of the paper; we use these preliminary results to represent the kinds of integrity attacks we expect in practice.

Images: Table 4 summarizes the different types of integrity checks we evaluate. Note that these are quality attacks; the altered versions of the images are still similar to the source image. (If it is substantially different, then LumHash component will detect the change.) Our current set of InfoName features protect against inset, quantization, and resize attacks. Our features/thresholds do not detect more sub-

Attack	Description	Protection?
Inset	embedding bogus content into image	LumLow changes
Quantization	making quality really poor; e.g., large pixels	ChromBlue, ChromRed change
Resize	rescale image and blow it up	LumHigh changes
Sharpness	making pictures hazy	none
Subtitles	adding random subtitles at base	none

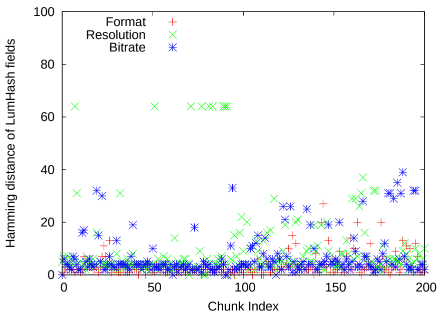
Table 4: Different types of integrity attacks against the image InfoName

tle changes: e.g, subtitles or transforms that make the image hazy. We believe that the types of attacks can be corrected/compensated using existing image restoration techniques.

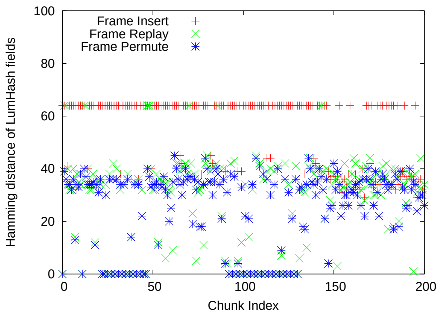
Videos: Since our video InfoName uses the image InfoName for the start/end frames, it inherits the robustness and integrity properties described above. The only concern then is with respect to integrity of frames within the chunk. We consider three types of integrity attacks: (1) frame insertion: a embedding bogus frames (e.g., advertisements), (2) frame replay: denying service by repeating the same start/end frame, and (3) frame permutation: disturbing user experience randomly swapping frames.

Figure 12 shows the effectiveness of the MaxLumHash in this context. We compare the MaxLumHash observed across transformed versions (Figure 12(a)) with that observed in the above integrity attacks (Figure 12(b)). As the plot shows, it is feasible to use a threshold to distinguish between the two. E.g, a threshold of 8 gives an effective false negative rate of 17% and 'false positive rate' of 17% too. However, most of the 'false positive' cases (where the attacks' MaxLumHash is near zero) in Figure 12(b) arise from the replay/permutation attacks. We discovered that in these instances the video itself had multiple instances of the same frame repeated over the chunk. For example, these chunks had multiple blank screen or MPAA rating notice frames. Thus, replay or permutations of the frames effectively results in the same content, and are not really attacks. These can be handled as special cases but we leave it for future research.

In contrast, we found that the other candidate for the feature *ChunkSummary*, viz., LumLow0 is not suited for this purpose. The match algorithm for HashSeq can be extended to this sequence of LumLow0 values. After the mapping stage, we can think of taking the Cosine Similarity Index (CSI) across the two equal length 'vectors' of frequencies. Ideally, for transforms, the CSI should be close to 1, while for integrity attacks, it should be close to 0. Figure 13 shows the actual plot for a set of video chunks, across their attacked versions. It shows that the points are all very close to 1. This shows that this feature is not viable as ChunkSummary for



(a) Transforms



(b) Integrity attacks

Figure 12: Using the LumHash sequence as an integrity check for videos

integrity purposes. Further analysis can be done with different metrics and features for ChunkSummary. But we leave a more comprehensive analysis of the integrity aspects of InfoNames as a subject of future research.

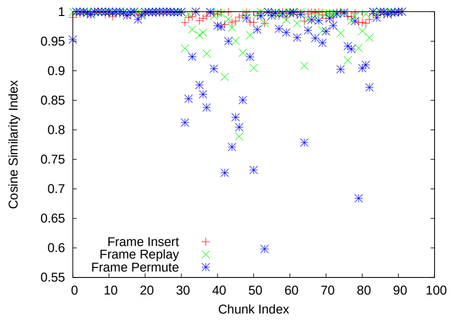


Figure 13: Plot of Cosine Similarity Index across attacked versions of chunks shows the inefficacy of LumLow0 sequence as ChunkSummary. Ideally, the points should be nearer to 0.

7. REFERENCES

[1] A Conversation with Van Jacobson: Making the case for content-centric networking. ACM Queue, Feb 2009.
[2] The jpeg standards. `http://www.jpeg.org/`.
[3] libFooID - free fingerprinting library. `http://www.foosic.org/`.
[4] The mpeg standards. `http://www.chiariglione.org/mpeg/`.
[5] The open source perceptual hash library. `http://phash.org`.
[6] Shazam. `http://www.shazam.com`.

[7] The uwash image database.
http://www.cs.washington.edu/research/imagedatabase/.

[8] The ycbcr color space. `http://en.wikipedia.org/wiki/YCbCr`.

[9] YouTube Video Identification Beta.
`http://www.youtube.com/t/video_id_about`.

[10] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transforms. *IEEE Trans. Computers*, 27(1):291–301, 1974.

[11] J. Boreczky, , J. S. Boreczky, and L. A. Rowe. Comparison of video shot boundary detection techniques. pages 170–179, 1996.

[12] B. Coskun and B. Sankur. Robust video hash extraction. In *In Proceedings (CD-ROM) of the European Signal Processing Conference, EUSIPCO 04*, 2004.

[13] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[14] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987.

[15] R. Kasturi, S. H. Strayer, U. Gargi, and S. Antani. An evaluation of color histogram based methods in video indexing. Technical report, Penn State University, 1996.

[16] C. L. Novak and S. A. Shafer. Anatomy of a color histogram. In *In Proc. of Computer Vision and Pattern Recognition (CVPR). IEEE*, pages 599–605, 1992.

[17] A. Oliva and A. Torralba. Building the gist of a scene: the role of global image features in recognition. *Progress in brain research*.

[18] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. of NSDI*, 2007.

[19] M. Rabin. Fingerprinting by random polynomials. Technical report, Harvard University, 1981. Technical Report, TR-15-81.

[20] S.-C. Cheung and A. Zakhor. Estimation of web video multiplicity. In *Proc. SPIE Internet Imaging*, 2000.

[21] T. Koponen et al. A Data-Oriented (and Beyond) Network Architecture. In *Proc. SIGCOMM*, 2007.

[22] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil. An architecture for Internet data transfer. In *Proc. NSDI*, 2006.

[23] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking Named Content. In *Proc. CoNEXT*, 2009.