CS-TR-3671 UMIACS-TR-96-55

Using Content-Derived Names for Caching and Software Distribution

Jeffrey K. Hollingsworth
hollings@cs.umd.edu

Ethan L. Miller
elm@cs.umbc.edu

University of Maryland Institute for Advanced Computer Studies

Abstract

Maintaining replicated data in wide area information services such as the World Wide Web is a difficult problem. Ensuring that the correct versions of libraries and images are installed for application programs presents similar challenges. In this paper, we present a simple scheme to facilitate both of these tasks using content-derived names (CDNs). Content-based naming uses digital signatures to compute a name for an object based only on its content.

CDNs can be applied to several common problems of modern computer systems. Caching on the World Wide Web is simplified by allowing references to an object by its content rather than just its location. In a similar fashion, applications can request library objects by their content without having to rely on the presence of a file system hierarchy that the application recognizes. Further, applications that require different versions of an object can coexist peacefully on the same machine. While this idea is still in its early stages, we present experimental evidence from a study of World Wide Web objects that indicates that CDNs could reduce network traffic by allowing requests to be satisfied by differently-named duplicates with the same contents.

Keywords: digital signature, global object names, disambiguation, World-Wide Web caching, WWW object duplication

This page intentionally left blank (I've always *wanted* to say that!)

1 Introduction

The explosive growth of the World Wide Web (WWW) has placed ever-increasing demands on existing computer networks and hardware, and the introduction of WWW-centric languages such as Java has exacerbated the problem. Low bandwidth and high latency are only part of the problem, though. With the WWW linking millions of independent computers worldwide, a single object may be duplicated on hundreds of computers, each using a different local name for the object. Detecting these differently-named duplicates can reduce network bandwidth and free up cache space by allowing local servers to cache just one of the duplicates, satisfying requests for “different” objects with a single copy.

A related problem is the distribution of software. Each computer has its own file system hierarchy, often with different naming schemes that require per-site customization to get a package to work. Worse, software packages often rely on other libraries which themselves evolve. Someone using the latest version of one tool might require the latest version of a library, while another tool that hasn't been updated in several years uses an older version of the same library. Keeping track of different versions of software and matching them can be a daunting task made difficult by the need to customize each piece of software to take the local file system structure into account.

This paper addresses many of these concerns with a simple mechanism: providing a name for objects based solely on their content. This name is the same for any two objects with identical content, regardless of their location. However, it is different for two objects that have the same location-based name but, due to changes over time, different contents. Caching becomes much simpler because an object has the same name regardless of where it is stored; a request for an object could be satisfied if that object were already cached from a different location. Moreover, the software distribution problem is simplified. Since an object's name does not depend on the file system hierarchy, location-specific customization is minimized. If a program specifies a needed library with such a name, it is guaranteed to get the correct version of the library because different versions have different contents and thus different names.

We first describe the work on which we based our new idea for naming documents, including previous research on caching for the WWW and some of the theory underlying digital signatures. We then describe several situations where content-derived object names provide benefits, including WWW caching, and software maintenance. Next we present experimental evidence supporting our claim that a significant number of identical WWW objects with different names exist. We conclude with a look at the implications of using content-based names in URLs, and some possible directions for future work.

2 Background

While the idea of using content-derived names (CDNs) for document identification on the WWW is a new one, there has been substantial previous work in the areas of naming and caching on the WWW. This paper builds on that work, as well as previous work in digital signatures.

2.1 Naming in the World-Wide Web

The most wide-spread naming scheme in the WWW is the URL (Uniform Resource Locator) [1]. This scheme bases object names solely on their locations, requiring users to ask a specific server for an object. This method has several shortcomings, including the inability to specify an object that could be retrieved from any one of several servers.

The URN (Uniform Resource Name) [8] was introduced to address this problem. Instead of specifying the document's actual name, a client can use the URN to find a server that would return the actual name and location of the desired document. Using this method allows a single document to be served in several distinct locations, and provides a method for a client to find the document using a single name. However, some problems still remain. First, the central URN server must be involved in every request for the document, though this load could be lightened by caching URN to URL translations. A second, more important, issue is that of keeping track of replicas. It is straightforward to notify a central authority when a document such as a technical report is duplicated, since it occurs relatively rarely. For objects such as images, though, such duplication will occur far more frequently. As an object is used in more and more places, the URN servers for it will become increasingly loaded. Keeping a small number of central naming authorities for a document is not scalable. Even if a scheme is developed to allow URN servers to scale, anyone creating a copy must notify the original URN server of the duplicate.

Sollins and Masinter [8] mention the use an MD5 digital signature [7] inside a URN to uniquely identify a document. The MD5 signature was used solely to generate a unique object ID, and not to identify objects from different locations with the same signature. In this paper we show that recognizing hidden replication provides the potential for caching to reduce both server and network load.

One of the biggest problems created by the development of the WWW is the bandwidth and server load required to provide the objects request by WWW clients. However, in any caching scheme where objects are mutable, some mechanism must be provided to age or invalidate stale copies of objects.

Gwertzman and Seltzer [4] examined several approaches to insuring that requests do not return stale objects. The Alex and time-to-live (TTL) protocols both perform well, with TTL requiring less bandwidth but imposing a higher server load than Alex. However, both of these protocols are necessary because objects with the same name may not, in fact, be the same object. While including a CDN in an object's name removes the need for consistency protocols for static objects, some objects (10% according to [4]) are dynamically generated and would not benefit from the use of CDNs.

2.2 Digital Signatures

A key feature of CDNs is the use of a digital signature to assign a unique name to an object based on its content. Algorithms such as MD5 [7] are one-way functions that take an arbitrary sequence of bytes and produce a result that is likely to be different from that of any other input sequence. MD5 is well suited as the function to content derived names. The MD5 algorithm produces a 128 bit signature, and Rivest [7] claims that it is NP-hard to find another document with an identical signature. Touch [9] has reported that it is possible to compute MD5 in software at the rate of over 10 MB/second on current RISC workstations; we feel this rate is more than adequate for our proposed use of MD5.

In order for a content-derived name system to work, it must probabilistically guarantee that two different objects will not share the same object identifier. A global information system such as the WWW, however, could have many billions of objects. Fortunately, the probability of such a failure is small. The probability that *m* numbers chosen randomly from a pool of *n* will be unique is $e^{-m(m-1)/2n}$ [5], where $n = 2^{128}$ for MD5. For 10^{15} objects, the probability of success (no two objects with different content have the same name) is $e^{-2^{-29}}$, assuming that object names are randomly distributed. Since $1 - e^{-x} \approx x$ for small *x*, the chance of failure is approximately 2^{-29}, or 10^{-9}. We believe this chance of failure is sufficiently low because it is below the probability that there would be an undetectable failure in a disk or network link somewhere in the world during that time. Thus, undetectable hardware failure is more likely to cause the use of an "incorrect" object than content-derived naming.

While 10^{15} objects would be sufficient for each of one hundred million servers to create ten million unique objects without name collision, 10^{15} unique objects may not be sufficient for many years of object creation by all of the computers connected to the WWW. However, there is no theoretical limit to the length of a digital signature. While MD5 only produces 128 bit signatures, a similar algorithm could be constructed to produce a signature with 256 bits, allowing the creation of 10^{30} unique objects with the chance of collision dropping to below 10^{-17}.This is sufficient to allow each of ten billion servers to create ten million unique objects *per second* for over three hundred years.

3 Applications of Content-Based Naming

Using content to name objects provides a simple way to request the specific instance of an object. It is extremely useful in situations where it is important that a specific version of an object be provided (such as software distribution). Likewise, it is also useful when it does not matter where the copy of an object is found (i.e., replication) as long as its the right object. Content-based names are the ultimate in location independent naming — they include no reference to what machine an object is stored on, nor do they require arbitrary filing hierarchies. In this section, we describe three uses for content-based naming: caching, software distribution, and world wide web search engines.

3.1 World Wide Web Caching

Perhaps the most obvious use for object names based on content is caching. Caching for the WWW has been studied in [2], in which several schemes for managing these caches were proposed. However, all of these caching mechanisms rely on the URL of an object to distinguish it from other objects.

This approach has two drawbacks. First, it is impossible to detect the existence of two identical files named by different URLs. In Section 4, we report that such files make up approximately 5% of all objects discovered by a WWW robot*. People who like images often include them in their own WWW pages, and will usually copy the images to their local server when they do so. Copying images reduces load on the original server, provides a safeguard in case the object is deleted from the original location, and reduces the amount of typing necessary to refer to the graphic in local WWW pages. While most of the objects are small, this situation is changing as small static graphics give way to miniature movies; as images grow in size, the bandwidth saved by caching at intermediate sites grows. Current WWW caching schemes do not recognize multiple instances of the same object if their names are different. URNs [8] attempt to solve this problem by providing a central name server to provide the name of the "closest" server, but are limited by the need to keep track of all of the copies of an object. Using an object's digital signature, on the other hand, has no such limitations. If two objects are the same, their digital signature will match. A WWW cache can look for objects bearing the same signature, and return any one that matches even if it was not from the specified server.

The second drawback to traditional caching approaches is that they are prone to caching stale data. Currently, this problem is solved in one of several ways. A WWW server may provide an estimated lifetime for a file; the cache in which the file is placed need not contact the server until this lifetime has expired. However, this approach has difficulty with objects whose lifetimes are highly variable, such as users' home pages. A single page may change several times in a day, or it may remain unchanged for weeks at a time. How can a cache cope with such unpredictability? Another consistency method requires the cache to ask the server when the object was last modified, and base its decision on the returned time and the modification time of the copy in the cache. This approach guarantees that the most up-to-date copy is used, but requires that the server handle a file stat operation for each object that is requested, even if the object is cached near the requester. A third method requires the server to "push" the new version of the object to machines that it believes have an old copy. However, this method introduces scaling issues.

If a digital signature is used, however, the cache can ensure that the object requested is the one it is holding by checking the signatures. If the signatures do not match, it must request a new copy from the object's server. The request goes to the file's server if and only if the file has changed; there are no spurious requests for files and all requests for a file are satisfied with the correct version.

Content-based naming depends on the relative immutability of objects on the WWW. If an object's contents change frequently while its location-based name remains the same, digital signatures will be of little use. Fortunately, the objects for which digital signatures will be of most use are least likely to experience small changes. Images and, eventually, multi-image objects are most likely to be copied around the WWW by users that want to include them in their WWW pages. These images are unlikely to be changed by individual users, and they are also the largest component of WWW traffic [2]. Keeping digital signatures for them will help to reduce the bandwidth requirements for home users who have plenty of disk space but are connected to the WWW via a slow link.

* This fraction would likely increase if the sample size is increased.

3.2 Software Distribution

The current trend in creating software is to re-use components such as classes, dynamic libraries, icons, and sound bites. For technical (size of the objects) or legal (different component vendors) reasons, different objects are stored as separate files. For a software product to work correctly, however, the different components must be compatible with each other. A single package may involve hundreds of individual files, each of which must be the right file in the right place in the directory structure. An incorrect version of a particular library or configuration file, or even the right file in the wrong place, can render the entire package useless. This situation is complicated by the evolution of software and the interdependence of software packages. A single computer often has dozens of software packages, some of which may require different versions of the same software library. Maintaining such systems is difficult at best, and installing new software is often a challenge.

The WWW has further complicated software configuration management by facilitating the distribution of software over the Internet. No longer is the installer a computer expert; instead, complex systems must be "installed" by less-experienced users. Languages such as Java [3] allow users to pull classes from many different locations, yet there is no guarantee that the files obtained in this way will actually work together. Some files may not work with the latest version of a Java class, instead requiring an older version. How can the software publisher specify a particular version of a Java class or similar object?

3.2.1 Ensuring Version Consistency

Currently, names, or names combined with a version string, are used to identify external components (such as dynamically linked library). However, using names as the basis of compatibility is problematic. Software quality assurance requires that product be tested with all compatible components prior to shipping. As a result, many products include copies of the tested components as part of their distribution. When the product is installed, the included components are installed on the target machine with the designated name. This ensures that the last installed product will have the correct components. However, any previously installed software may now break because it may rely on older (or newer) versions of components that have been replaced by more recently installed software.

Using digital signatures provides a good solution to finding consistent versions of objects. Each library, icon, or sound bite will have an object identifier computed using a digital signature. This signature uniquely identifies an object solely based on its content. We term this digital signature a content-derived name (CDN) since it can be used to fully specify a requested object. When an application (or a library) wants to reference or load an external component, it simply specifies the CDN for the desired object. A library (or perhaps the file system) then locates the requested object and loads it. Since different versions of the same software component will have different CDNs, each application will get the desired version. Consistency is also required for some Web pages. As web pages become more dependent on specific visual layouts, it is necessary to ensure that all components included on the page are the exact ones the author intended.

At software installation, each component included with the distribution is loaded only if its CDN is not already installed. On the other hand, a new version of a software package may leave unmodified many of the files that it uses. These objects will retain the same object identifiers as in the older version of the software, allowing the user to load only the files that have changed since the last version was released. Objects can refer to other objects by their CDN forming a graph of object dependencies. The graph makes it possible to ensure that all required components for an application are installed.

Traditionally, shared objects have been distributed in relatively large units (libraries) because maintaining consistent versions was so difficult. However, the use of content-based naming allows the sharing of objects at a much finer granularity since the verification of consistent versions can be automated.

Sometimes it is possible for an application to be able to use more than one version. For example, an application might be compatible and have been tested with either of two similar object libraries. This case can easily be accommodated by lists of equivalent CDNs. As long as one of the objects specified in the equivalence list is present, the installation process does not need to load an object.

It is also possible that an application could be customized during installation or by the user at a latter time. For example, users might add custom macros to their word processing system. Customizations could be applied to either the application or to individual objects. However, customizations could potentially change the content of an object (and thus its CDN). To accommodate this situation, each object should contain a customization region that contains fields that can be changed. This part of the object would not be used in computing its CDN.

The overall structure of an object in this scheme is shown in Figure 1. An object consists of the object body, external object references, customization region, and its CDN. The object body contains the majority of the object, including its executable code. References to other objects or customization data are represented as pointers to the appropriate section of the object. Each object reference can be a list of CDNs for equivalent objects. Although this information is immutable, it needs to be in a designated section of an object so that the object manipulation routines can identify an object's external object references. The customization region has two sections. One to store customized references to other objects and the second to store free format data. The only requirement is that pointers from the object body can't be modified due to customization since this would change the object's CDN.

3.3 File Hierarchy Independence

A second benefit from digital signatures is file system location independence. Many packages require extensive per-site customization to tell the software where to find files it needs. However, these files usually have the same contents (except for information on the location of other necessary files!) regardless of where in the directory structure they are located. A package that refers to an object using its digital signature need only look it up in a database of signatures and objects. Objects can be assigned human-usable names; this will likely be necessary for other maintenance reasons.

A file system to store objects based on their CDN could be built. Such a file system would support efficient storage and linkage of CDN-based objects. In addition to objects referring to other

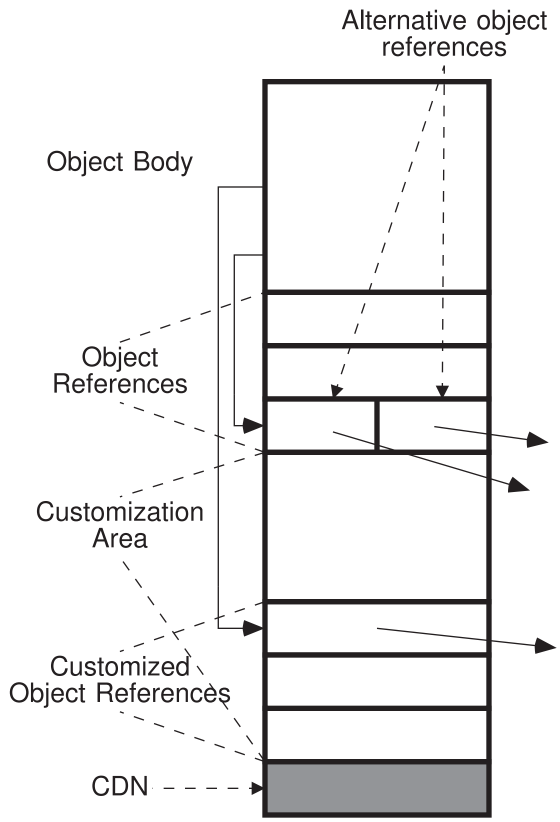

Figure 1. The structure of an object using content-derived names

objects, a user- visible namespace could be provided similar to the way a UNIX directory structure provides a user access to inodes. Objects would not be explicitly deleted from such a file system, but would be implicitly deallocated. Each stored object would have a reference count. When a reference count was decreased to zero, the object would be deleted. Due to the possibility of mutually referential objects creating unreachable cycles, a periodic garbage collection of the object will also be required. Replication in such a file system is easy since a request for a CDN can be serviced by any server that has the desired object. A request for a CDN is a request for specific data, and not a request to translate through a namespace. Traditional consistency concerns only apply for the user-visible name space, not the objects themselves.

In some ways, a CDN-based file system would be similar to the PILOT file system used on the Altos [6]. In the Pilot file system, all stored objects had a globally unique object identifier. However, this object identifier was created by concatenating the host identifier of the server where the object was created and a sever relative identifier. The Pilot file system also supported linking files together based on their object identifiers. However, a program using the file system needed to explicitly mark an object and immutable before its object identifier was frozen. With a CDN, no explicit designation is required. However, a server must explicitly make an object available, an operation similar in meaning to marking it immutable.

The ability to locate objects by an identifier based solely on their contents will simplify software distribution via the WWW. Already, programs such as Netscape use a single directory to cache retrieved objects. Currently, the identifiers for these objects in the cache bear no relation to the objects' contents. Using digital signatures instead would allow the “cache” to grow to the size of the entire disk. At that point, a Unix-like (or Mac-like, or Windows-like) directory structure could be used as an overlay, providing a user-friendly interface to a cache of objects fetched from the WWW. Any piece of software could be distributed in this way; the user could assign names to those pieces of software that she wished to access directly, such as the main executable for a word processing program. Other objects would be fetched as necessary. To allow disconnected operation, a package might arrive with a self-installer that contains all files necessary to run the program. Rather than install all of the files, though, such an installer would merely install the files not already present locally.

3.4 Web Search Engines

Content-derived names can also improve the quality and efficiency of web search engines. Many popular documents are replicated (mirrored) at several different sites. Each of these replicas will have different URLs, but the same content. Not only will the server name be different, but sites often arrange their file systems slightly differently so the path names of replicas will also be different. To the user of a search engine, though, each replica is identical. Unfortunately, most search engines currently generate different “hits” for each of these copies. However, if the search engine maintains a CDN for each object, duplicate hits due to replication can be presented as a single hit with several choices. Merging replicas can save time for users of search engines and reduce load on servers since users will only visit a single site. Use of CDNs by search engines can easily be added since it only requires changes to the internal information of search engines, not any web protocol.

CDNs can also be used to assist in the maintenance of search engines indices. To see if a document has changed, search engines must either fetch and re-index the document or query a server to discover the modification time of an object. While using modification time provides an indication if an object has changed, using a CDN such as MD5 would provides a better indication if a document has really changed.

In addition, each server could export a list of publicly visible objects stored on that server and their CDNs. This list could be periodically updated by the server. Search engine crawlers would then be able to compare the URL to CDN mappings provided by the server to previous values. Out of date index entries for document would be identified by differences in their CDNs.

In addition, mapping information can also be used by a crawler to identify if a URL is a replica of some other object that has already been indexed. If the object (as indicated by its CDN) is a copy of an existing object, the crawler would not need to request the document from the server. Instead, it would simply add an entry for the new copy. This method works equally well for identifying when a document has not changed, but has simply be moved to a new part of the server's file system. Again, the URL will be different, but the CDN will remain the same.

4 Experiments

To evaluate the amount of object replication in the web, we wrote a simple web robot to scan objects and record their URL and MD5 hash values. The idea was to try to find pages on the web that have different URLs, but are in fact the same object. Since we were interested general object duplication, we computed MD5 values for all objects found including HTML documents, images, and Postscript files. We term the same object with two (or more) different URLs as a name aliased object.

[NOTE TO REVIEWERS: This results in this section are very preliminary. A more detailed study will be presented in the final paper. It will include a larger number of pages, starting from several randomly selected URLs, and a more complete analysis of the results.]

The scanning program was written in Perl and configured to scan recursively to a depth limit of 13. The robot was started at a single web page, and left to run. The robot scanned and computed the MD5 hash value for 28,974 web pages*. After running this robot, we then looked for different URLs that resulted in the same MD5 hash value. We were interested in finding out not only how many URLs mapped to the same object, but also finding out why the same object was being replicated.

Because the scan was done using a robot, rather than measuring traffic requests, the data presented is based on a static count of WWW pages rather than on dynamic access counts, such as those available by monitoring traffic through a proxy cache). Dynamic data would be more useful for investigating the potential opportunity for caches to identify replicated objects since it captures temporal re-use of objects. Static data has the advantage that it is easier to gather information from disparate parts of the web since proxy based monitoring would be biased by idiosyncrasies in the proxy's usage pattern.

Type	Count	Percent
fragment identifiers (# characters)	3,130	10.8
host alias	298	1.0
substring match	427	1.5
name aliased object, same server	471	1.6
host alias and name alias	58	0.0
Other matches	430	1.5
Total duplicates	4,814	16.6

Table 1. Duplicated objects by type of replication.

A summary of duplicate objects is shown in Table 1. Of the 28,974 URLs scanned, there were 24,156 different MD5 hash values. The majority, 3,130 (65%) of the duplications were due different fragment identifiers that were part of the URL. Fragment identifiers (preceded by an octothorpe) are used to identify views, often relative offsets, into documents. These are generally uninteresting duplications because it is trivial to eliminate them. Another source of duplication

* Due to time constraints, the search was terminated before a complete scan at this depth was finished.

was host name aliases. This type of duplication occurs when a single host has several DNS names that map to the same IP address. This type of duplication was responsible for 298 (6%) of the duplicates. Again, this type of replication could be detected by existing name based schemes if the names are defined based on IP address rather than host name.

The remaining four categories of duplication, totaling 1,386 objects (29%), would be impossible to detect using name based object equivalence. Three of these four categories are due to intra-server aliasing of object names. The category "substring match" is the case where one URL name is a substring of another URL name. Substring aliases were responsible for 427 duplications or 9% of the duplicates. Substring aliasing results from servers mapping requests for a directory names (e.g., /foo/bar) to a specific pages (e.g., /foo/bar/index.html). It is easy to visually identify these aliases. However, since the mapping from directory name to default page can be configured on a per-server basis, it is impossible for caches or robots to identify these aliases based on name. Same server name aliasing of objects also occurs where one name is not a substrings of the other. This results from either having duplicate copies of the same object stored on a server, or from local file system aliasing (e.g. symbolic links). This type of aliasing was responsible for 471 duplications (10%). The final type of intra-server aliasing was a combination of host name (DNS) alias and local file system alias. This was relatively rare and accounted for only 58 of the duplications seen.

The final category of name aliasing is duplication of the same object on different servers. We found 430 duplicates (9%) were due to inter-server name aliasing. This type of duplication results from replication or cloning of objects. We would like to distinguish replication, an explicit copy for availability or load balancing, from cloning, a copy of a useful object (such as a red ball). The distinction between replication and cloning is important because several proposals have been made to accommodate replication in the web and other distributed information services. However, all of these schemes require explicit management of replication and do not accommodate cloning. Unfortunately, it is difficult the accurately identify if copies are replicas or clones. However, for content-based naming, the fact it is the same object with two different names is all that is important.

Type	**All Objects**		**Duplicated Objects**	
	Count	**Percent**	**Count**	**Percent**
text/html	23,654	81.6	1,099	79.3
image/gif	3,622	12.5	253	18.3
text/plain	959	3.3	16	1.2
image/jpeg	422	1.5	11	0.8
Other	316	1.1	7	0.5
Total	28,973		1,386	

Table 2. Object types.

We also investigated the question of whether different object types were more likely to be replicated than other types. A comparison of the types of all objects with those that were replicated is shown in Table 2. In both cases, the most common object type is a HTML document (approximately 80% of all documents scanned were HTML). The fraction of replicated objects that are gif images was somewhat higher than the rate of occurrence of gif images in the overall sample space. This matches the intuition that people "borrow" GIF images from other web pages to use them on their pages.

5 Future Directions

The work presented in this paper is still in early form, and we see several different ways in which this idea could be used to build better systems. One of the difficulties with the use of CDNs as part of an object's WWW name is that changes of this scope require wide agreement to be effective. However, CDNs could still be used on a small scale to test their effectiveness. The HTTP protocol allows user-defined fields to be transmitted along with an actual document; one of these fields could contain the MD5 digital signature of the document. The client or proxy cache could cancel the transfer if it already had a document with that digital signature, possibly saving bandwidth by removing the need to transfer the entire object. This would be particularly useful for larger images since smaller documents, less than a few kilobytes, would be transferred in their entirety before the message to stop would arrive at the source. Nonetheless, we believe this strategy would prove particularly useful in caching image files and binary files — many sites mirror software distributions, yet detecting that the files are the same even though they come from different sites is difficult.

Another immediate use for CDNs is in WWW search engines. Currently, search engines are unable to detect duplicate documents stored in different places. Our experience with several searches using Digital Equipment's Alta Vista search engine (http:/altavista.digital.com) turned up many instances of the same document being found on different sites. This problems occurs both for WWW pages and Usenet articles that are archived at several sites around the Internet. Simply keeping MD5 digital signatures for each page would allow the search engines to return shorter lists, and allow a user a choice of site from which to fetch a document that she wants.

Mobile computing can also benefit from CDNs. One of the problems in mobile computing is the ability to operate a computer away from its "base," since requests for objects must eventually be sent through the network to the mobile computer's home file server. CDNs present an attractive alternative: a mobile computer can merely ask the local file servers for an object using its CDN. The mobile computer need not follow the same file pathname conventions as the local server, since the object is identified solely by its content. Moreover, the mobile computer can check that it received the object it requested by computing the digital signature on the object, so it need not even trust the local server.

Eventually, if network-based software distribution replaces physical distribution, most of the disk space on client computers could be turned into a cache for network objects. When a commercial software package is purchased, a request for an object would fetch it from the WWW if it was not already cached. There would be no need for garbage collection; an object would simply be removed from the cache to make room for new objects. If that discarded object were needed again, it could be refetched. Much research on caching strategies and other issues is necessary before this goal can become a reality, but its implementation would greatly simplify the operation of computers in an environment where access to the WWW is constant and omnipresent.

6 Conclusions

In this paper we have presented a new approach to naming objects for distributed systems and software configuration. Rather than using user-assigned names to identify objects, we proposed to derive object names automatically based on the content of an object. Our scheme makes it possible to identify references to the same object even if the objects have completely different names. We described how content-based naming of objects can be used to manage cache and replication in a distributed information service, to ensure consistent software configuration, and to improve the performance and quality of WWW search engines.

Content-based naming represents an attractive alternative to creating location-independent names for objects. By splitting the namespace from the content-space, it is possible to decouple namespace issues — hierarchical vs. relational naming, name aliasing from content-space issues such as caching, replication, and storage management.

Finally, we presented the results of a robot that scanned for duplicated WWW objects with different names. We discovered that approximately 5% of the objects found were duplicates of other objects, supporting our claim that using CDNs will provide immediate benefit to proxy caches and search engines in addition to the longer-term benefits to software distribution and WWW caching.

References

1. T. Berners-Lee, L. Masinter, and M. McCahill, Uniform Resource Locators (URL), RFC 1738, Network Working Group, 1994.
2. A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, “A Hierarchical Internet Object Cache”, USENIX — Winter 1996 Conference. January 1996, San Diego, CA, pp. 153-163.
3. J. Gosling, B. Joy, and G. Steele, The Java Language Specification. 1996: Addison-Wesley.
4. J. Gwertzman and M. Seltzer, “World-Wide Web Cache Consistency”, USENIX — Winter 1996 Conference. January 1996, pp. 141-151.
5. R. Motwani and P. Raghavan, Randomized Algorithms. 1995: Cambridge University Press.
6. D. D. Redell, Y. K. Dalal, T. R. Horsley, H. C. Lauer, W. C. Lynch, P. R. McJones, H. G. Murray, and S. C. Purcell, “Pilot: An Operating System for a Personal Computer”, Communications of the ACM, Feb 1980. **23**(2), pp. 81-92.
7. R. L. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, Network Working Group, 1992.
8. K. Sollins and L. Masinter, Functional Requirements for Uniform Resource Names, RFC 1737, Network Working Group, 1994.
9. J. D. Touch, “Performance Analysis of MD5”, SIGCOMM. Aug 1995, Cambridge, MA, pp. 77-86.