The AT&T Internet Difference Engine: Tracking and Viewing Changes on the Web*

Fred Douglis†
Thomas Ball‡
Yih-Farn Chen†
Eleftherios Koutsofios†

†AT&T Labs – Research
180 Park Ave, Florham Park, NJ 07932-0971
{*douglis, chen, ek*}*@research.att.com*

‡Lucent Technologies, Bell Laboratories
1000 E. Warrenville Rd., Naperville, IL 60566-7013
tball@research.bell-labs.com

January, 1998

Abstract

The AT&T Internet Difference Engine (AIDE) is a system that finds and displays changes to pages on the World Wide Web. The system consists of several components, including a web-crawler that detects changes, an archive of past versions of pages, a tool called *HtmlDiff* to highlight changes between versions of a page, and a graphical interface to view the relationship between pages over time. This paper describes AIDE, with an emphasis on the evolution of the system and experiences with it. It also raises some sociological and legal issues.

Keywords and Phrases: World Wide Web, notification, differencing, version repository, HTML, AIDE, Ciao, HtmlDiff, push technology.

1 Introduction

As the World Wide Web (WWW) evolves, it is undergoing rapid change in the ways in which individuals obtain information. First, the volume of content available on the WWW has been growing at phenomenal rates, overwhelming the ability of users to keep up with the available data. Second, the mode of data transfer is shifting to a bimodal "push-pull" model in which some data is *pushed* to users rather than being *pulled* by them upon request [34]. The "push" phenomenon is partially an outgrowth of the scale of the Internet, since it permits services to get the attention of users asynchronously rather than waiting for the users to return on their own. It

*This article appeared in *World Wide Web*, 1(1), pp. 27–44, January 1998.

is also well-suited to transient data, such as stock quotes, headlines, and weather. Finally, it offers a simple interface, permitting installation and use by unsophisticated users.

Data that a user "pulls" may change as well, though usually at a slower rate. Once a user has identified a page[1] of interest, he or she may wish to retrieve a fresh copy of it when it has changed in an interesting way. The problem is to know *when* a fresh copy should be retrieved, and to identify *how* the page has changed.

A number of tools are now available to assist users in keeping track of when pages of interest change. Generally these tools fall into two categories: centralized services that register interests in URLs from many users (e.g., URL-Minder [26], the Informant [19]), and tools that run on a user's machine and will poll the URLs listed in local files such as bookmarks (e.g., Katipo [27], Surfbot [33], Smart Bookmarks [13]). With the exception of information provided explicitly by the content provider in a Smart Bookmarks "bulletin," these tools and methods help answer the *when* problem but not the *how* problem.

In addition, an increasing number of pages have support for notifying users of changes via email, or at least icons or "what's new" pages to highlight important changes. Email notification tells users when to revisit the page, and icons or a special "what's new" page tells them how the page has changed once they retrieve it. However, what's new to one user may be old to another, or vice-versa: if the granularity of updates to a shared "what's new" page does not match the frequency with which a user visits, such a summary is of limited use. Furthermore, deletions are unlikely to appear in a "what's new" page or to be highlighted within a page, even though deletions are often as interesting as changes or insertions.

The AT&T Internet Difference Engine (AIDE) provides a *personalized* view of how pages on the WWW change [10]. It uses a centralized notification system, version archive, and differencing tool to allow users to track and view changes to arbitrary pages on the WWW. It also supports recursive tracking and differencing of a page and its descendants, and graphical viewing of the relationships between these pages [11].

This paper describes AIDE and reports experiences with it. The next section provides a taxonomy of notification systems with respect to several important criteria. Section 3 describes the architecture of the system and discusses its components. Section 4 reports on some lessons from our experience with the system, Section 5 discusses ongoing issues and extensions, and Section 6 describes a few additional applications of AIDE. Finally, Section 7 concludes the paper.

2 Notification Systems: A Taxonomy

The many services that provide notification of changes to WWW pages share a few features in common, and vary in several other respects. Here we give a brief taxonomy of notification services, which is summarized in Table 1.

The common thread of the notification services is the need to identify pages of interest to a user, note the "state" of each page, and subsequently check the page to see if its state has changed. Generally, the "state" refers to the modification timestamp of the page if it is available, or a function of the contents of the page (such as an MD5 signature [30]) if not.

Beyond this common underlying framework, the systems diverge.

[1] We use the commonly used term *page* to refer to the entity that is technically called a *resource*. It is the data one retrieves via, e.g., an HTTP GET request.

	URL-Minder	Informant	Katipo	Surfbot	Smart Bookmarks	AIDE
Platform	Linux	AIX	Macintosh	Windows (95, NT)	Windows (3.1, 95, NT)	UNIX
Client/Server	server	server	client	client	client	server
Track Bookmarks?	no	no	yes	yes	yes	partial
Prioritization?	no	no	no	no	bookmark structure	yes
Recursion?	no	searches	no	yes	no	yes
Notification?	email	web page	web page	separate application	highlights bookmarks	web page or email
Show Differences?	no	no	no	no	no	yes

Table 1: Comparison of notification systems.

2.1 Client or Server Tracking

Client-based tools run on a user's machine, typically on the user's list of bookmarks, either periodically or on demand. Periodic background checks for changes have the advantage of ensuring that the user will have relatively up-to-date information about the pages being tracked, but require Internet connectivity at the time specified and for a duration in proportion to the number of pages tracked, and possibly their size. On-demand checks will not be especially useful if the number of pages being tracked is large.

Server-based tools track pages that have previously been specified by users, for instance by submitting a form with their email address and one or more URLs to track. All such services check for updates asynchronously, and make changes known to users either via email or by reporting changed pages over the WWW upon user request. One could envision integrating notification with other push-based technology such as Pointcast [28] or Castanet [22], notifying users via a separate push-based application rather than email or the WWW.

While server-based tools have the benefit of amortizing the overhead of tracking popular pages, accessing each page once rather than one access per user, they also have a few drawbacks. One is that the client-based tools have direct access to the user's bookmarks, while the server-based tools can at best access the bookmarks over the WWW itself; this requires that the bookmarks be publicly available, or protected via some shared secret between the user and the service. Another is the necessity for the user to provide the server with information about which pages to track, possibly with HTTP authentication information (the user/password information that browsers sometimes prompt users with via a dialog box, rather than for example a secret entered via an HTML form). Both of these drawbacks relating to privacy are discussed in Section 5.2. Finally, direct access to the browser's "history" gives client-based tools the ability to have exact knowledge of the version of a page that the user last viewed, which permits them to give more accurate information about which pages are new. On the other hand, server-based tools will have no knowledge of a user going directly to a page, if the user bypasses the tracking service. The server-based tools will have information about which users have visited which versions of a page, however, again a privacy issue.

2.2 Prioritization

If a user wishes to track just a few URLs, the means by which a tracking system informs the user of changes is relatively unimportant. Email, whether a single message listing all recently changed pages or a message per changed page, should not prove a burdensome intrusion. Similarly, a WWW page listing all changed pages will serve well because all the pages listed can be perused quickly. A typical method of listing all changed pages might be a report showing pages in reverse chronological order of modification date, with the most recently modified pages at the top of the report. Another alternative is to highlight changed pages directly in the view of the user's bookmarks, as Smart Bookmarks do. The notification might even include summary information about ways in which a page has changed, similar to the "Website News" functionality described in Section 3.4.

None of these approaches scales well when a user tracks hundreds or thousands of pages of varying importance. Individual email messages would overwhelm the user with asynchronous interrupts, even if a mail filter might be used to prioritize the messages or automatically sort them into folders. They might also accumulate if multiple updates to a page were reported before a user could process the mail.

A single list of all changed pages, whether via email or as an HTML page, will force the user to scan the list to find the most important ones. Highlighting bookmarks works well with a small bookmark file, but after bookmarks are organized into hierarchical folders, highlighting a folder to indicate that one or more of the URLs contained within it has changed will result in extra navigation by the user to determine exactly which pages have changed.

An alternative, which AIDE uses, is to sort the URLs on the basis of user-specified criteria that will enable the user to find the most important changes easily while still being able to browse the list. By default, AIDE shows changes that were detected since the user last requested a report, then changes that were previously presented to the user; within each group, the pages are sorted by a user-specified numeric priority field. Thus, a recent change to a page the user rates as important will be near the top of the list, while an older change to a relatively unimportant page will appear at the end. While these priorities are currently fixed, it would be possible to vary them dynamically based on user access patterns (e.g., degrading the priority of a page that is reported but not viewed by the user).

In addition, AIDE allows a user to specify that email be sent when changes to particular URLs are detected. If multiple changes are detected in a single pass (usually run nightly), they are listed in order of priority.

2.3 Recursion

Given a URL to track, a tool (whether running on the client or a server) can track the page referred to by that URL, or it can track the page and some subset of the pages to which the page has links. For example, a user who is interested in mobile computing might watch both the "virtual library page on mobile computing" [37] and the pages for the different projects, products, and other entities referenced by the virtual library page. As another example, consider querying a search engine and being notified when either the results of the query change (a new document matches the query) or a page that was previously reported in the result has changed.

As the depth of recursion grows, the tool performing the tracking behaves increasingly like a "spider" such as those used by various search engines, with corresponding complexity and performance considerations. For example, it might be reasonable to poll the virtual library page on a daily basis, but one might not wish to poll the pages to which it refers with so high a frequency.

Tracking pages recursively also poses a user-interface problem. If one registers an interest in many pages, some of which are tracked recursively, how should the user be informed of changes? Since recursive tracking can result in many more pages being tracked implicitly (the result of recursion) than explicitly (the result of a user request that specifies a certain page), notification of changes to recursively tracked pages may swamp notifications of pages in which the user has expressly registered an interest. Possible solutions include notifying the user that the "parent" references "children" that have changed, requiring the user to navigate manually to get information about the children, or separating the implicit and explicit URLs in any reports of changed pages. AIDE uses the latter approach, with implicitly tracked pages at the end of the report and signified by a special icon. An implicit page inherits the numeric priority of the page from which it was reached, permitting a simple clustering of related pages by priority.

Some restricted forms of recursion may also be useful. Informant [19] tracks changes to pages that are the result of a query, for instance to a search engine, but does not recurse in general.

2.4 Differences

Nearly all the tools that track changes to pages provide only a notification that something on a page has changed. Smart Bookmarks permit a content provider to include information within a page about how it has changed, which a browser that is enabled with Smart Bookmarks can display to the user along with the notification that the page has changed, but this approach requires that content providers explicitly provide such information and does not guarantee that the user will see it in a timely fashion.

Web Integrity [25] is not a notification tool; instead, it is a collection of tools to maintain WWW servers and content. It may be used by WWW server administrators and others to view a history of pages served and to see a side-by-side representation of the differences between them.

To our knowledge, only AIDE provides the user with the ability to see how an arbitrary page has changed since it was previously seen. It does this by archiving pages, either automatically or on demand, and then comparing a later version to an archived one using a derivative of Hirschberg's solution to the longest common subsequence (LCS) problem [17] (adapted to HTML). These components of the system are discussed in detail in the following section.

3 System Architecture

The goal of AIDE is to permit users not only to find out about changes, but to display information about those changes. Figure 1 depicts the architecture of AIDE. Users interact with the system via HTML forms, which pass requests to a CGI script. The script in turn can perform archival operations, to save or retrieve a version of a page; comparison operations, to display the differences in textual or graphical form, or to list new pages.

Users may specify a number of operations, including:

- register a URL to track, including the degree of recursion through links to other pages;
- specify that a new version of a URL that's already being tracked should be saved;
- see a list of new pages;
- view textual differences between a pair of versions (usually the current version and the one most recently archived, but this may be overridden); or
- view a graph showing the structure of a page, including the state of pages to which it refers.

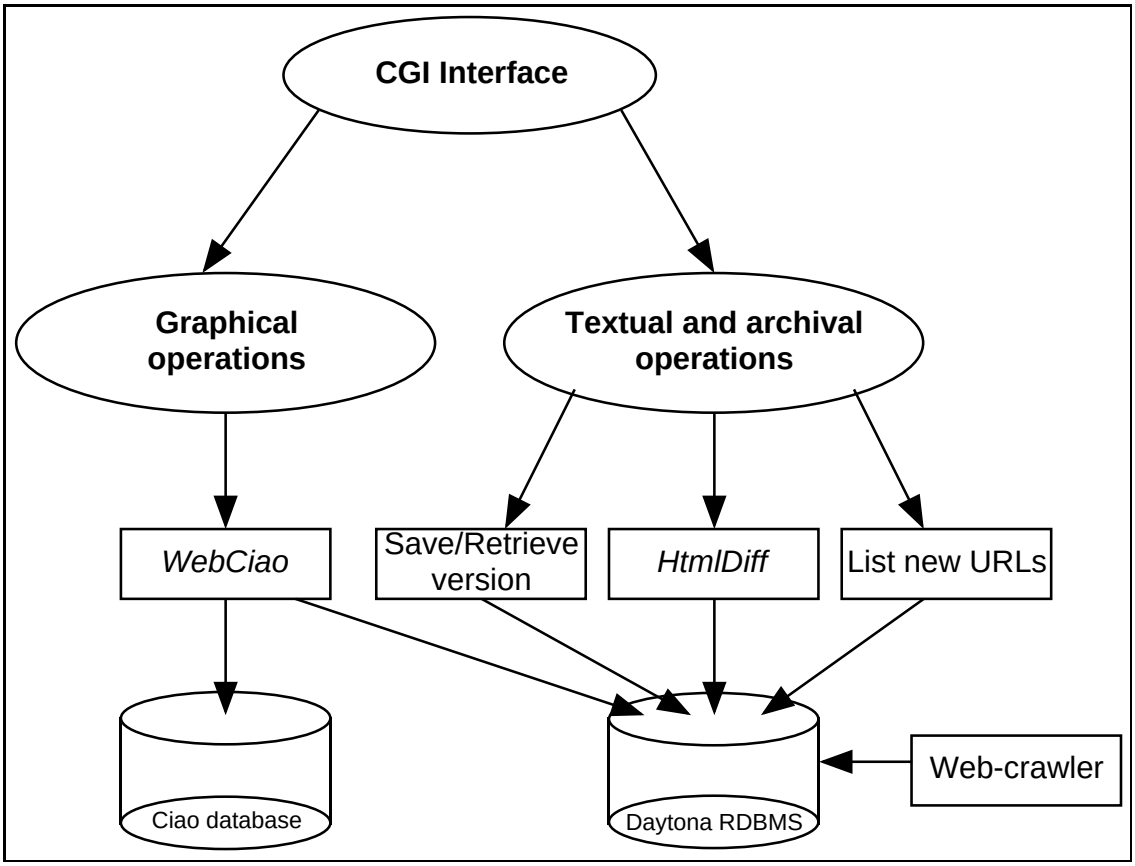


Figure 1: Architecture of AIDE.

In addition, the server periodically runs a web-crawler that checks pages for changes, optionally precomputing differences from past versions. The crawler and CGI script share access to a relational database containing information about all pages being tracked and all users of the system. WebCiao (the graphical component described in Section 3.4) uses the AIDE archive and relational database but stores structural information in its own entity-relationship database as well.

Here we describe several components of the system: notification, difference generation, archival, graphical display, and databases.

3.1 Notification

As noted in Section 2.1, the primary characteristic that distinguishes a notification service is the location where it runs: on a user's machine, or on a central server.[2] AIDE began as the former and eventually moved to the latter, for a few reasons. First, it provided a simpler interface. AIDE's initial development predated tools, such as Java [20] and ActiveX [2], which permit servers

[2] To our knowledge, no one has implemented a hybrid approach using both clients and servers; we defer a discussion of this possibility until later.

to download executable code to clients in a safe and transparent fashion. Instead, users had to manually download and install a Perl script and associated libraries, and few were willing to do so. Running as a server was simpler for individual users: one needed only to create an "account" on the server once, and then provide URLs to track, rather than installing software locally. (Note that the perceived complexity of software installation is largely due to the UNIX environment in which the tool was developed; "self-installing" packages are much more common on PCs and Macintoshes than UNIX, and are now also possible using Java or ActiveX.)

Second, a shared server is more scalable. It's undesirable for many users to poll the same page periodically if one host can poll it, along with others, and notify a given user of all changes at once.

Third, as mentioned above, a server may have Internet connectivity when an individual user does not, or it may have higher-speed access. For instance, an Internet Service Provider (ISP) might poll pages on behalf of its clients rather than expect them to poll the pages themselves over 28.8Kbps modems.

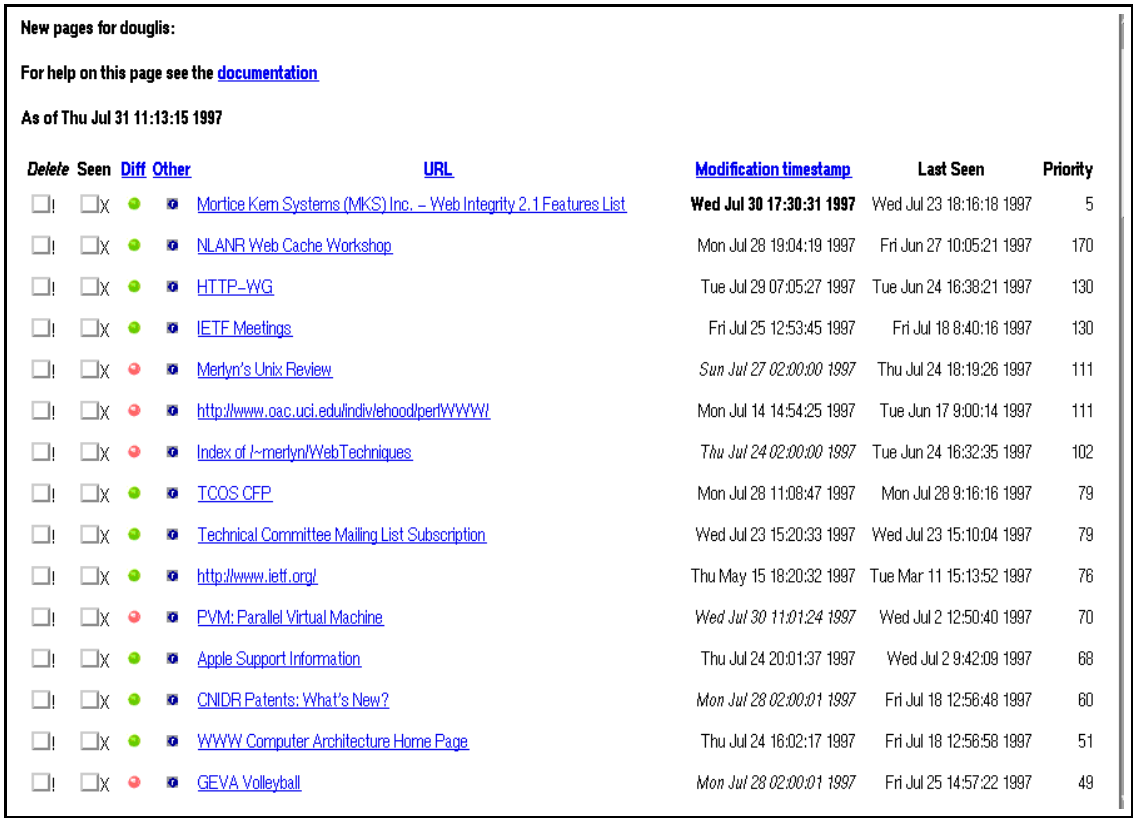
Fourth, integrating the notification service with the difference generation and archival components of the system, on a central server, has interface and performance advantages. For example, the server knows which pages might have differences that have already been computed and cached, and indicates these pages to the user in the list of new pages.

Some of the advantages of a server-side notification tool can be achieved by a hybrid client-server system as well. The server could poll pages in order to take advantage of economies of scale and network connectivity, while the client could use local information such as the browser's history to personalize the content. Also, by using an applet or other client-side tool to display new pages, one could ensure that the view of new pages is kept more current than CGI output would be. Such a hybrid approach has not been implemented in AIDE, nor in any other tools of which we are aware.

AIDE supports recursive tracking of changes. Users specify the number of levels to recurse; currently, they are restricted to tracking just a page or the page and its immediate descendants (called "children"). When recursion is specified, changes to child pages are reported separately by default.

Users interact with the server via a set of forms. Typically, one would request a list of new pages, as shown in Figure 2. The pages at the top of the list, with timestamps in bold text, were "recently" detected. Pages that had previously been reported as new appear below them. Each set of pages is sorted by a numeric priority. The checkboxes allow the user to delete selected pages, or to remove them from the list of new pages. The **Diff** column includes an image of a ball, with the color indicating whether a page is automatically archived and compared against its predecessor (green) or not (red). The icon leads the user to a form to update information about the page, delete it from the set of pages being tracked, see a list of past versions, and other operations. Currently, pages that are deleted from a user's set of pages are kept in the archive indefinitely, but they are not tracked for subsequent changes if no users have an active request for them.

Lastly, the web-crawler is a Perl script that obtains from the RDBMS a list of URLs to track during a given invocation, and retrieves those pages sequentially. It uses the HTTP `HEAD` request to obtain the last-modified timestamp unless it has knowledge from a prior attempt that no timestamp is given, in which case it retrieves the body via `GET` and computes a simple checksum. It also retrieves the body if recursion is specified, since it parses the page to find its anchors. Only pages that the user has not previously tracked are added as a result of recursion; if one of these child pages is uninteresting to the user and deleted from the list, it is ignored when the web-crawler encounters it again later.



New pages for douglis:

For help on this page see the documentation

As of Thu Jul 31 11:13:15 1997

Delete	Seen	Diff	Other	URL	Modification timestamp	Last Seen	Priority
☐!	☐X			Mortice Kern Systems (MKS) Inc. – Web Integrity 2.1 Features List	**Wed Jul 30 17:30:31 1997**	Wed Jul 23 18:16:18 1997	5
☐!	☐X			NLANR Web Cache Workshop	Mon Jul 28 19:04:19 1997	Fri Jun 27 10:05:21 1997	170
☐!	☐X			HTTP-WG	Tue Jul 29 07:05:27 1997	Tue Jun 24 16:38:21 1997	130
☐!	☐X			IETF Meetings	Fri Jul 25 12:53:45 1997	Fri Jul 18 8:40:16 1997	130
☐!	☐X			Merlyn's Unix Review	*Sun Jul 27 02:00:00 1997*	Thu Jul 24 18:19:26 1997	111
☐!	☐X			http://www.oac.uci.edu/indiv/ehood/perlWWW/	Mon Jul 14 14:54:25 1997	Tue Jun 17 9:00:14 1997	111
☐!	☐X			Index of /~merlyn/WebTechniques	*Thu Jul 24 02:00:00 1997*	Tue Jun 24 16:32:35 1997	102
☐!	☐X			TCOS CFP	Mon Jul 28 11:08:47 1997	Mon Jul 28 9:16:16 1997	79
☐!	☐X			Technical Committee Mailing List Subscription	Wed Jul 23 15:20:33 1997	Wed Jul 23 15:10:04 1997	79
☐!	☐X			http://www.ietf.org/	Thu May 15 18:20:32 1997	Tue Mar 11 15:13:52 1997	76
☐!	☐X			PVM: Parallel Virtual Machine	*Wed Jul 30 11:01:24 1997*	Wed Jul 2 12:50:40 1997	70
☐!	☐X			Apple Support Information	Thu Jul 24 20:01:37 1997	Wed Jul 2 9:42:09 1997	68
☐!	☐X			CNIDR Patents: What's New?	*Mon Jul 28 02:00:01 1997*	Fri Jul 18 12:56:48 1997	60
☐!	☐X			WWW Computer Architecture Home Page	Thu Jul 24 16:02:17 1997	Fri Jul 18 12:56:58 1997	51
☐!	☐X			GEVA Volleyball	*Mon Jul 28 02:00:01 1997*	Fri Jul 25 14:57:22 1997	49

Figure 2: Example of AIDE's report of which pages are new. An explanation of the figure appears in the text.

3.2 Difference Generation

Tracking changes to pages is useful in some cases, but when the changes are subtle there is little point in notifying the user of a change—the user will likely not know what has changed. For example, consider a list of persons within an organization; when a new person joins, or one leaves, the appearance or absence of a one-line entry in a long list of persons will hardly be noticed unless one knows where to look. If one could go straight to the specific parts of the page that have changed, even subtle changes would be apparent.

Showing the differences to a WWW page requires two abilities: obtaining both the old and the new version of the page, even when the content provider is only serving the newer version; and comparing the the two versions to display the differences. We discuss archival in the next subsection; here we focus on a new tool we developed, called *HtmlDiff* [10], which performs the latter function. It takes two versions of a page as input and produces a new syntactically correct HTML page that highlights the differences between the two pages. A banner at the top of the new page indicates how many changes have occurred and has a hyperlink to the first change, and each change is similarly linked to the next, with each change marked by a special symbol (e.g., ➧) that also serves as an internal anchor. In addition, *HtmlDiff* flags inserted text with ***bold italics***

and deleted text with ~~struck-through letters~~. Changes to existing text are treated as deletions alongside insertions. An example appears in Figure 3.

Currently, *HtmlDiff* runs as a standalone SML [23] application on a UNIX platform. It is invoked by the daemon that detects changes to pages, in the common case in which newly changed versions are automatically archived (see Sections 2 and 3.3). It is also invoked by the CGI Perl script that handles interactive requests, when the differences have not already been computed or when the page has changed since that computation occurred. An alternative would be to push the computation of the differences out to individual clients, via a Java applet or some other mechanism; this is considered further in Section 3.2.4.

3.2.1 What's in a Diff?

HTML separates content (raw text) from markups. While many markups (such as <P>, <I>, and <HR>) simply change the formatting and presentation of the raw text, certain markups such as images (<IMG src=...>) and hypertext references (<A href=...>) are "content-defining." Whitespace in a document does not provide any content (except perhaps inside a <PRE>), and should not affect comparison.

At one extreme, one can view an HTML document as merely a sequence of words and "content-defining" markups. Markups that are not "content-defining" as well as whitespace are ignored for the purposes of comparison. The fact that the text inside <P>...</P> is logically grouped together as a paragraph is lost. As a result, if one took the text of a paragraph comprised of four sentences and turned it into a list (<UL>) of four sentences (each starting with <LI>), no difference would be flagged because the content matches exactly.

At the other extreme, one can view HTML as a hierarchical document and compare the parse tree or abstract syntax tree representations of the documents, using subtree equality (or some weaker measure) as a basis for comparison. In this case, a subtree representing a paragraph (<P>...</P>) might be incomparable with a subtree representing a list (<UL>...</UL>). The example of replacing a paragraph with a list would be flagged as both a content and format change.

We view an HTML document as a sequence of sentences and "sentence-breaking" markups (such as <P>, <HR>, <LI>, or <H1>) where a "sentence" is a sequence of words and certain (non-sentence-breaking) markups (such as <B> or <A>). A "sentence" contains at most one English sentence, but may be a fragment of an English sentence. All markups are represented and are compared, regardless of whether or not those markups are "content-defining" (however, as described later, certain markups may not be highlighted as having changed). In the paragraph-to-list example, the comparison would show no change to content, but a change to the formatting.

We apply Hirschberg's solution to the longest common subsequence (LCS) problem [17] (with several speed optimizations) to compare HTML documents. This is the well-known comparison algorithm used by the UNIX *diff* utility [18]. The LCS problem is to find a (not necessarily contiguous) common subsequence of two sequences of tokens that has the longest length (or greatest weight). Tokens not in the LCS represent changes. In UNIX *diff*, a token is a textual line and each line has weight equal to 1. In *HtmlDiff*, a token is either a sentence-breaking markup or a sentence, which consists of a sequence of words and non-sentence-breaking markups. Note that the definition of sentence is *not* recursive; sentences cannot contain sentences. A simple lexical analysis of an HTML document creates the token sequence and converts the case of the markup name and associated (variable,value) pairs to upper-case; parsing is not required.

We now describe how the weighted LCS algorithm compares two tokens and computes a non-negative weight reflecting the degree to which they match (a weight of 0 denotes no match).

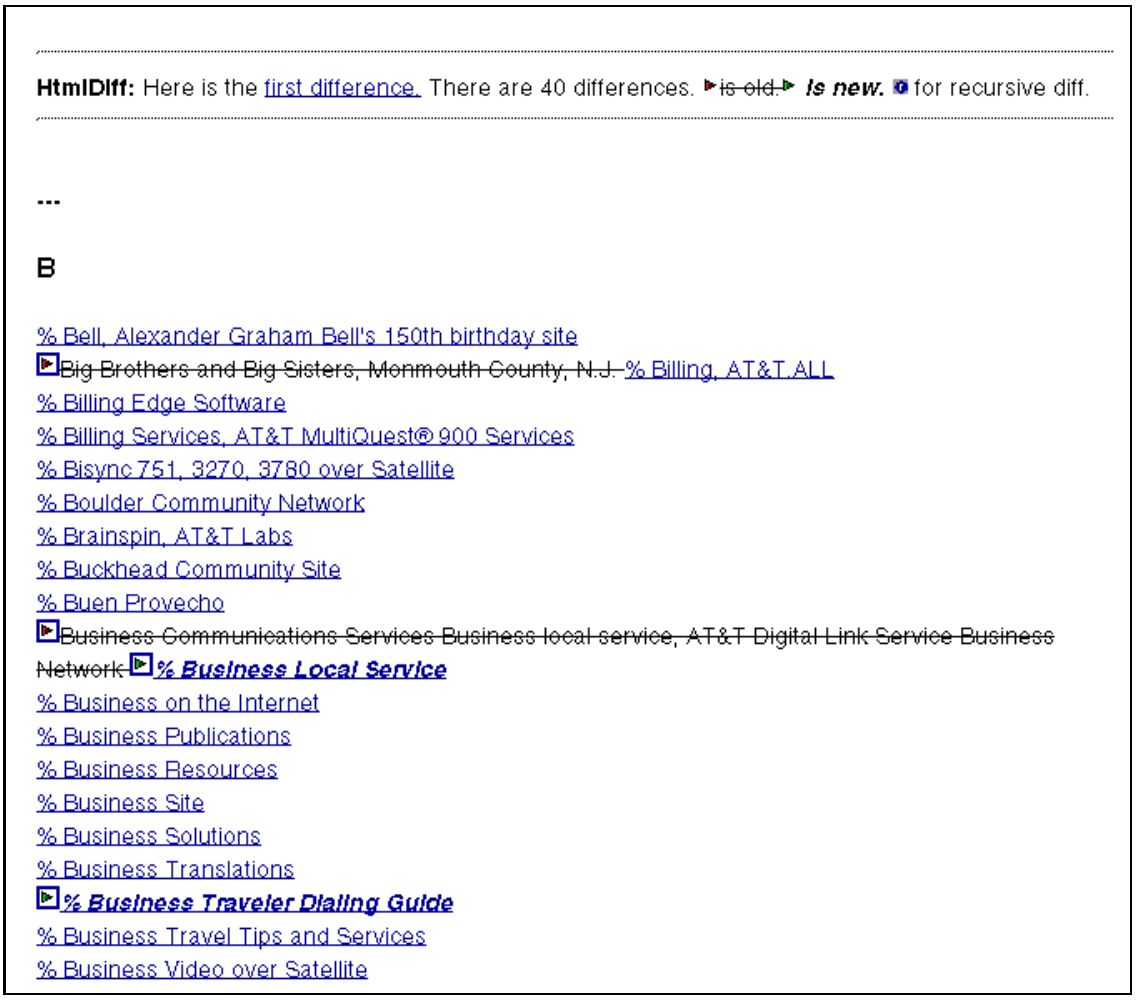


Figure 3: Output of *HtmlDiff* showing the differences between a subset of two versions of the AT&T "A to Z" page, `http://www.att.com/atoz/list.html`, as of 24 March 1997 and 17 April 1997. Arrows (▶) point to changes, with bold italics indicating additions and with deleted text struck out. The borders around the arrows are because they are internal anchors, linked together in a chain of differences for the user's convenience. The percent signs are used for recursive differencing, as described in Section 3.2.3. The banner at the top of the page was inserted by *HtmlDiff*.

Sentence-breaking markups can only match sentence-breaking markups. They must be identical (modulo whitespace, case, and reordering of (attribute,value) pairs) in order to match. A match has weight equal to 1. Sentences can match only sentences, but sentences need not be identical to match one another.

We use two steps to determine whether or not two sentences match. The first step uses sentence length as a comparison metric. Sentence length is defined to be the number of words and "content-defining" markups such as <IMG> or <A> in a sentence. Markups such as <B> or <I> are not counted. If the lengths of two sentences are not sufficiently close, as defined by a threshold, then they do not match. Otherwise, the second step computes the LCS of the two sentences (where words matching exactly against words are assigned weight 1, and markups match exactly against markups, as before) and assigns a non-negative integer weight W to the two sentences. Let W be the number of words and content-defining markups in the LCS of the two sentences and let L be the sum of the lengths of the two sentences. If the percentage $(2 * W)/L$ is sufficiently large, then the sentences match with weight W. Otherwise, they do not match (equivalently, they match with weight $W = 0$).

For example, the HTML:

```
<p> The quick brown <a href=''fox.html''>fox</a>.
<p> The fat lazy <a href=''cow.html''>cow</a>.
```

contains two sentence breaking markups (<p>), dividing the HTML into two major sentences. If the HTML changes to become:

```
<p> The quick red <a href=''fox.html''>fox</a>.
<p> The languishing <a href=''holstein.html''>heifer</a>.
```

then the first sentence in the original HTML matches the first sentence in the new HTML with weight 4, while the second sentence in the original HTML matches the second sentence in the new HTML with weight 1. In this example, our algorithm will match the first sentence of the original HTML to the first sentence of the new HTML, noting that the word "brown" changed to "red". The algorithm will not match the second sentences of the original and new HTML.

3.2.2 Presentation of the differences

The comparison algorithm outlined above yields a mapping from the tokens of the old document to the tokens of the new document. Tokens that have a mapping are termed "common"; tokens that are in the old (new) document but have no counterpart in the new (old) are "old" ("new"). We refer to the "old" and "new" tokens as "differences".

We investigated three basic ways to present the differences by creating HTML documents:

Side-by-Side A side-by-side presentation of the documents with common text vertically synchronized is a very popular and pleasing way to display the differences between documents. It has been used to display regular text (see, for example, UNIX *sdiff* or SGI's graphical diff tool *gdiff*). Unfortunately, at the time *HtmlDiff* was first implemented, there was no good mechanism that allowed such synchronization. Subsequently, support for *frames* has offered the opportunity to display the changes in this fashion, and at least one tool—Web Integrity, from Mortice Kern Systems, Inc. (MKS)—now provides this sort of comparison [25]. We have not yet implemented side-by-side comparison in *HtmlDiff*.

Only Differences Show only differences (old and new) and eliminate the common part (as done in UNIX *diff*). This optimizes for the "common" case, where there is much in common

between the documents. This is especially useful for very large documents but can be confusing because of the loss of surrounding common context. Another problem with this approach is that an HTML document comprised of an interleaving of old and new fragments might be syntactically incorrect.

Merged-page Create an HTML page that summarizes all of the common, new, and old material. This has the advantage that the common material is displayed just once (unlike the side-by-side presentation). However, incorporating two pages into one again raises the danger of creating syntactically or semantically incorrect HTML (consider converting a list of items into a table, for example).

Our preference was to present the differences in the merged-page format to provide context and use internal hypertext references to link the differences together in a chain so the user can quickly jump from difference to difference. We currently deal with the syntactic/semantic problem of merging by eliminating all old markups from the merged page (note that this doesn't mean all markups in the older document, just the ones classified as "old" by the comparison algorithm). As a result, old hypertext references and images do not appear in the merged page (of course, since they were deleted they may not be accessible anyway). However, by reversing the sense of "old" and "new" one can create a merged page with the old markups intact and the new deleted. A more Draconian option would be to leave out all old material. In this case, there are no syntactic problems given that the most recent page is syntactically correct to begin with; the merged page is simply the most recent page plus some markups to point to the new material.

With *HtmlDiff*, modified "content-defining" markups are highlighted, while changes to other markups (such as <P>) are not. Consider the example of changing the URL in an anchor but not the content surrounded by <A>...</A>. In this case, an arrow () will point to the text of the anchor, but the text itself will be in its original font.

3.2.3 Recursive Differencing

HtmlDiff also supports recursive differencing: when displaying the differences in a page, one can follow a special link inserted by *HtmlDiff* to see the differences in the linked page as of approximately the same points in time as the versions of the current page [11]. *HtmlDiff* queries the database (see below) to determine which linked pages have versions available to compare recursively. A special icon () is used to indicate an anchor that can be compared recursively; in this case, clicking on the icon displays the differences between the two versions of the linked page, while clicking on the anchor goes to that page as it currently is. The differences are computed in real time if they have not previously been computed and cached. A percent sign (%) is used to indicate an anchor that does not have two versions available, in which case clicking on the percent sign leads to the current version of the page, annotated with links for further recursive comparisons [11].[3] Figure 4 gives an example of recursive differencing.

3.2.4 Performance

The version of *HtmlDiff* currently in use is written in SML [23] and must run on a server rather than coresident with a user's browser. Each invocation of *HtmlDiff* has the potential to use significant resources, both computation and memory, thereby limiting the scalability of the system

[3] The use of and % is somewhat historical, and somewhat due to the desire to emphasize recursive differences while making the more common case when no differences are available unobtrusive. Other characters or icons could be used, as long as the likelihood of confusion with the content of a page being displayed is extremely low.

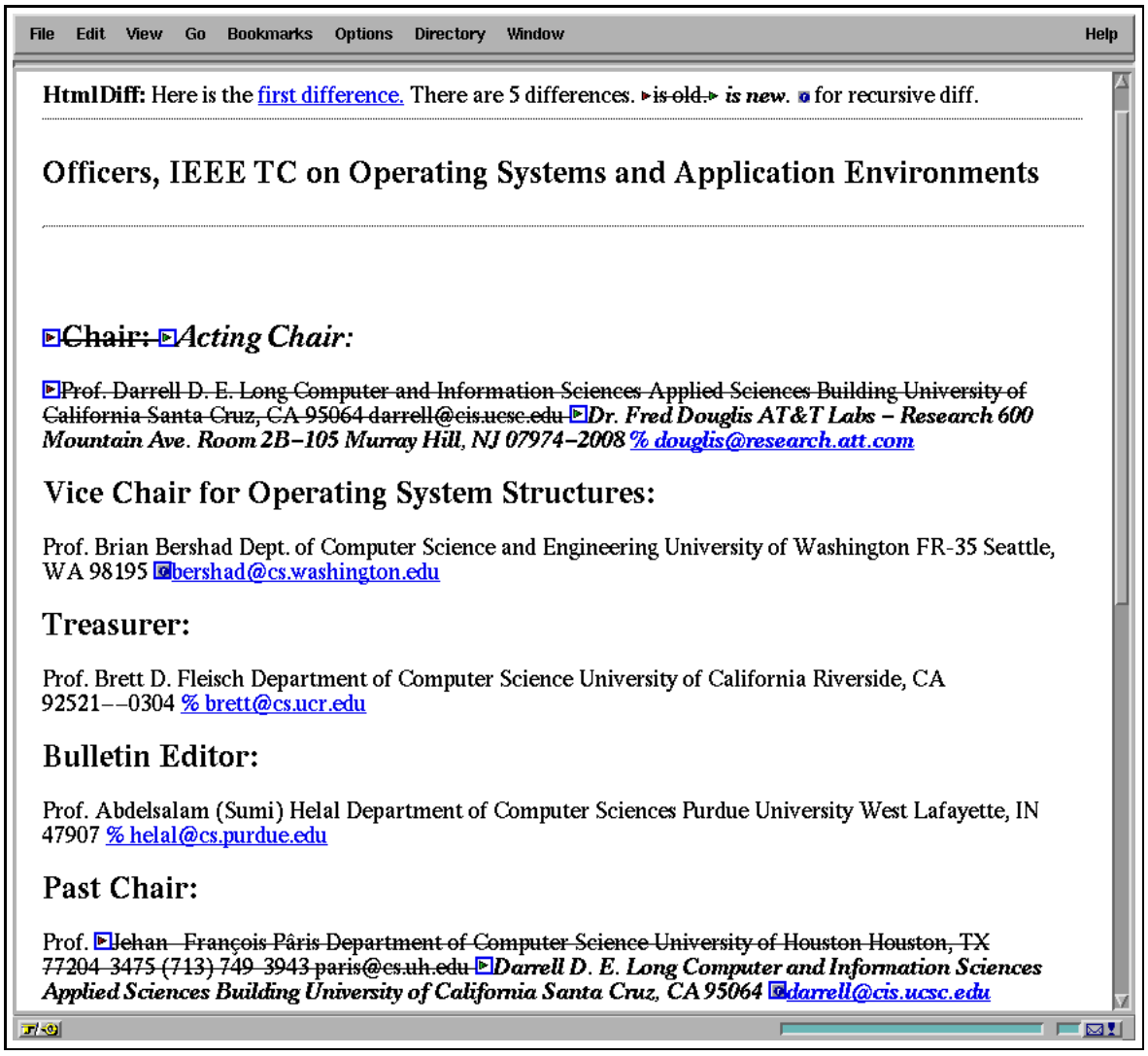


Figure 4: Example of *HtmlDiff* as applied to a page of the IEEE Computer Society Technical Committee on Operating Systems, `http://www.cs.odu.edu/~mukka/tcos/officers.html`, in December 1996. A special icon ( ) indicates the availability of recursive differencing.

and making interactive computation of the differences undesirable. For example, computing the differences between two versions of the AT&T Labs–Research organization chart (with approximately 800 anchors, 4000 words, and 100,000 bytes, and with much similarity from line to line) took over three minutes on a 200MHz Sun Ultra 1, although simple HTML files can be compared in roughly 200ms. In either case, the DRAM "footprint" of the SML process is approximately 20 Mbytes.

Obviously, both the memory and processing overheads limit the number of differencing operations the server can perform at a time. DRAM costs are low enough that a dedicated server could be expected to support tens of instantiations at a time, but processing overheads would limit the parallelism more dramatically. Even if the process management overhead could be reduced by moving the differencing calculation into a long-lived server, that might eliminate at most half of the minimum 200ms overhead, permitting at most 10 *HtmlDiff* runs per second on a uniprocessor Ultra 1 that does nothing else. (In practice, pages would be of varying complexity, and sometimes take significantly longer to perform the comparison.) As processor speeds improve, the throughput should increase accordingly.

The cost of computing the differences can be addressed in a few ways. First, AIDE allows users to request that differences be computed and cached at the time a change is detected (typically overnight). Although the total computation is the same as if the differences were computed interactively—and in fact may be greater, since some differences are never requested by users—interactive performance is much better.

Second, we are exploring a version of *HtmlDiff* implemented in Java, which can run as an applet on the user's machine. An initial implementation by Elliot Berk of Princeton University demonstrated the feasibility of an applet but also raised interesting issues, particular the need for an "*HtmlDiff* server" to act as an intermediary between the applet and arbitrary hosts, in order to bypass the restriction that Java applets could only communicate with the host from which they were loaded. The Java implementation has not yet been integrated with AIDE, for a couple of reasons:

- Nearly all difference computation is performed by the server when changes are detected, so there is relatively little opportunity to push computation to the client without an inordinate delay.
- The Java implementation of *HtmlDiff* was demonstrably slower than the SML version in simple test cases, and was unable to function on the complicated example mentioned above (the organization chart). It also needs support for 8-bit character sets. Further debugging and optimization are needed.

Third, *HtmlDiff*—in any language—should be optimized. We have not yet attempted any significant tuning. Note that the overheads associated with HTML-level differencing, as *HtmlDiff* does, are much higher than would be experienced by a delta-encoding system that simply considers changes in the raw data [24].

Since *HtmlDiff* is a separable component of AIDE, it could be used as a standalone service. In fact, it is possible to submit two URLs to AIDE and request that *HtmlDiff* be run interactively to compare the URLs. The performance of the existing implementations suggest that *HtmlDiff* cannot be deployed publicly yet without significant constraints on its use, such as permitting only one client at a time to be active, and keeping a single client from monopolizing the server. (In fact, before AIDE moved to a standalone system, it ran on the personal workstation of one of the authors, and occasional parallel interactive invocations of *HtmlDiff* caused interactive performance to suffer dramatically.)

3.2.5 Experiences

After using *HtmlDiff* at length, several things became apparent. Generally, *HtmlDiff* has been effective at highlighting the subtle changes that previously went unnoticed: for example, a professor's home page when "Assistant Professor" changes to "Associate Professor" upon receiving tenure. *HtmlDiff* has also been of great use in tracking large dynamic output, such as organization charts. However, some aspects of *HtmlDiff* could be improved, including the following:

Structured text *HtmlDiff* treats HTML markup as sentence breaks when appropriate, but it does not consider physical relationships between pieces of text, such as elements of a table or preformatted text with line breaks. Therefore, it is possible to align the new markup "incorrectly," as in Figure 5: while one can make a case for highlighting the insertions across lines, it would appear that the new line was inserted as a unit. (Note that this is a well-known problem with the LCS algorithm [17].) Another issue arises from *HtmlDiff*'s stripping the HTML from text that has been deleted, since structured text such as table entries will be output as a single linear text stream. For instance, Figure 3 shows the text "Business Communications Services Business local service, AT&T Digital Link Service Business Network" as a single phrase, while originally it was spread across two lines.

AAAA	BBBB	CCCC
AAAA	DDDD	EEEE

AAAA	BBBB	CCCC
AAAA	FFFF	GGGG
AAAA	DDDD	EEEE

AAAA	BBBB	CCCC
AAAA	*FFFF*	*GGGG*
AAAA	DDDD	EEEE

Figure 5: Misaligned differences. The left box shows the original text, the center the new text, and the right the highlighted differences. One would prefer that all the text in the second line be highlighted as new.

Mundane differences Changes of the form `This page was last modified on ...` or `There have been 1234 visitors to this page` are generally uninteresting, as are embedded timestamps. *HtmlDiff* needs a mechanism for filtering out such changes in general, and for uninteresting changes that are particular to a single URL or set of URLs.

Additional issues and improvements are discussed in Section 4.2.

3.3 Archival

Displaying differences between versions of a page on the WWW requires that past versions be available. An *archive* of pages can serve several potential purposes, beyond the scope of AIDE. For instance, the Internet Archive is attempting to store a snapshot of the entire WWW that is freely accessible, consisting of terabytes of data [21]. Such an archive is intended for historical purposes, much as libraries archive paper documents for future generations, but is not well-suited to the relatively fine-grained access needed by AIDE: such a large system might revisit a page after weeks or months, not days. However, they ultimately intend to archive multiple versions of pages, and with sufficient resources such an archive might serve our purpose.

In many cases, content providers have access to a history of their documents. For instance, they might use a version control system to record changes to their pages, including comments. Even before AIDE, we developed a CGI interface to permit users to see the revision history of a document, using the Revision Control System (RCS) [35], as long as the CGI script resided on the same machine as the version repository and the RCS file for the document was in the same public

area. From that history one could view a specific version or (once *HtmlDiff* was available) see the differences between a pair of versions. MKS's Web Integrity now provides similar functionality [25].

Using an archive provided by a content provider would not be suited to AIDE, for a number of reasons:

- Most importantly, few content providers support this functionality.
- If they did, the granularity of "check-in" by the owner of a page might not match the frequency of viewing. A page might change several times but be checked in only at the end of the period of instability.
- Unless dynamic data (the output of CGI programs) were archived each time they were generated, which might have prohibitive overhead in disk space, users would not be able to rely on the content provider to return the correct old version.

Note however that server-based archives do have other benefits, particularly when used in conjunction with delta-based encoding schemes to ship updates to previously delivered data [24].

Thus AIDE archives pages itself, using RCS. By centralizing the repository, the system amortizes the storage overhead of storing the same or similar versions of pages for many users simultaneously. Pages may be archived upon request by a user, or automatically when a change is detected. Currently, automatic archival is completely at the discretion of the users: if any user requests automatic archival, via a checkbox in the HTML form indicating that a page should be tracked, it will be auto-archived, and the output of *HtmlDiff* will be computed and cached. The frequency of checks, and therefore of archival, is limited by the invocation of the script that checks for changes (currently it runs twice daily). In practice one would also limit the number of URLs that a particular user could track, or charge for the service, to keep the service from being overwhelmed by a small number of users. Limits would also guard against denial-of-service attacks.

Pages are archived indefinitely, and a page that changes frequently can result in many versions. If the changes are substantial, the storage requirements may be equal to the size of one version multiplied by the number of versions. In practice, only a few pages that have been tracked by AIDE have exhibited this sort of behavior, but this is because people must explicitly request that pages be archived automatically, and it would be unusual to request that a page that changes both frequently and radically be archived on each change.

Assuming that pages are automatically archived when changes are detected, the storage requirements of the archival facility are proportional to the number of pages tracked and the frequency with which they change. Pages that are not automatically archived will generally have less of an impact. As of early August 1997, the facility had tracked about 7200 pages and used 500 Mbytes of disk storage for the versions themselves, with another 5 Mbytes used by the RDBMS managing version histories and other metadata. This averages to 70 Kbytes of overhead per page being tracked, though the variation among pages is significant. In fact, 143 Mbytes (29%) of archive data are accounted for by the 6 largest RCS version histories, all of which are at least 10 Mbytes long and were pages that were randomly selected (using AltaVista [8]) for a study of the rate of change of WWW data, rather than being registered by a human. Thus one may expect the overhead for a real user community to be lower.

We discuss the implications of copyright protection on archival in Section 5.1.

3.4 Graphical Navigation

The Web Graphical User Interface to a Difference Engine, or WebGUIDE [11], uses WebCiao [5], the WWW version of the *Ciao* graphical navigator [4] to browse relationships between pages. WebGUIDE

displays these relationships as a directional graph of nodes, where each node represents a page, and arrows between the nodes represent anchors and are labeled by the content of the anchor ("content" refers to the text surrounded by `<A>...</A>` for textual anchors and the URL of an image for graphical ones).[4] Each node is colored to indicate whether it contains new (red), changed (yellow), deleted (white) or unchanged content (green). Each link is a solid arrow if it existed in both the old and the new versions of the page containing the anchor; it is dashed if it had been added to the previous version, and dotted if it had been deleted from the previous version. For example, Figure 6 shows structural changes in the neighborhood of AT&T Labs' home page, `http://www.att.com/attlabs/`, from 13 February 1997 to 19 Feburary 1997. The picture shows that a link to AT&T Research's home page was just added to the Labs' home page and several pages (including those of AT&T Labs and AT&T) have changed during that one-week period. No pages were deleted in its neighborhood. Users can also visit any page on the graph by simply selecting the "visit" operation on any pop-up node menu.

By tracking and archiving pages recursively, WebGUIDE can start with one "root" page and obtain information from the Daytona RDBMS about the pages pointed to by the root (see the next subsection). It then colors the nodes in the graph accordingly. Thus, recursion goes beyond other tools such as WebCopy [38], which will retrieve a URL and its descendants, since it archives versions of each page separately and indicates when differences are available recursively. The basic mechanism for retrieving pages, parsing them, and retrieving their descendants is the same.

If several users are interested in daily changes of a particular domain (such as telecommunications), we can analyze the first few layers of pages of these companies and provide a daily report on new items appearing on these WWW sites, which we call "Website News." For example, Figure 7 shows a news report on 1 August, 1997 on new text links added in the first two layers of web pages in AT&T, MCI, Ameritech, and other telecommunications companies in the United States.

3.5 Databases

AIDE uses two types of database. Most of the system shares a relational database, while the graphical portion uses a separate entity-relationship database.

3.5.1 Relational Database

The first version of AIDE stored information about version timestamps in a user's home directory, and information about archived versions on the central server. The latter data were kept in per-user flat files, as described in Section 3.3, which became expensive to reparse each time. While we could have moved to a hashing package (e.g., from 4.4BSD [32]), we chose instead to use the Daytona RDBMS [15].

The database stores data on a per-URL and per-user basis. Information from multiple users is aggregated when applied to a URL, so for instance if one user requests that a page be tracked at least once per day and another requests once per week, the more frequent checks are performed on behalf of both users. It also stores the time when a user most recently viewed a page, either directly (using a redirection from a CGI script) or by viewing *HtmlDiff* output, so it can list only those pages that have changed since the user viewed them. Each URL has a set of RCS versions associated with it, and the versions that correspond to the requests from a particular user are also recorded.

[4] This can result in some confusion when taken out of context: for example, the AT&T home page refers to AT&T Labs with the comment *Visit the home of the communication revolution*, so the arrow from `http://www.att.com/` to `http://www.att.com/attlabs` is labeled "home."

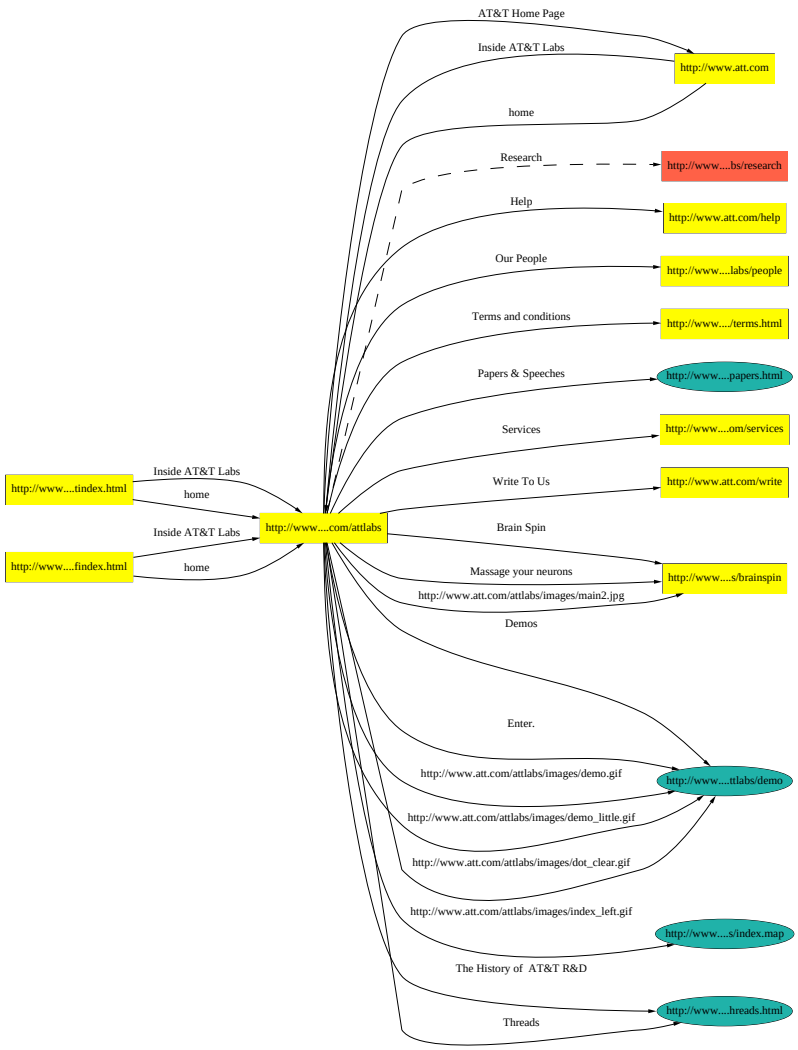


Figure 6: Structural changes in the neighborhood of `http://www.att.com/attlabs/`, from 13 February 1997 to 19 Feburary 1997. It shows three pages that link to this page, and numerous pages linked to by this page. (One page, `http://www.att.com`, links into this page via two different anchors and is linked to by this page as well.) The dashed line to `.../research` indicates a new anchor. The light-gray (yellow when in color) rectangles correspond to changed pages or images, the dark-gray (red) rectangle is newly linked to the `attlabs` page, the dark-gray (green) ovals are unchanged, and there were no deletions. Note that the ovals are included here to distinguish them in grayscale; normally they would be rectangles.

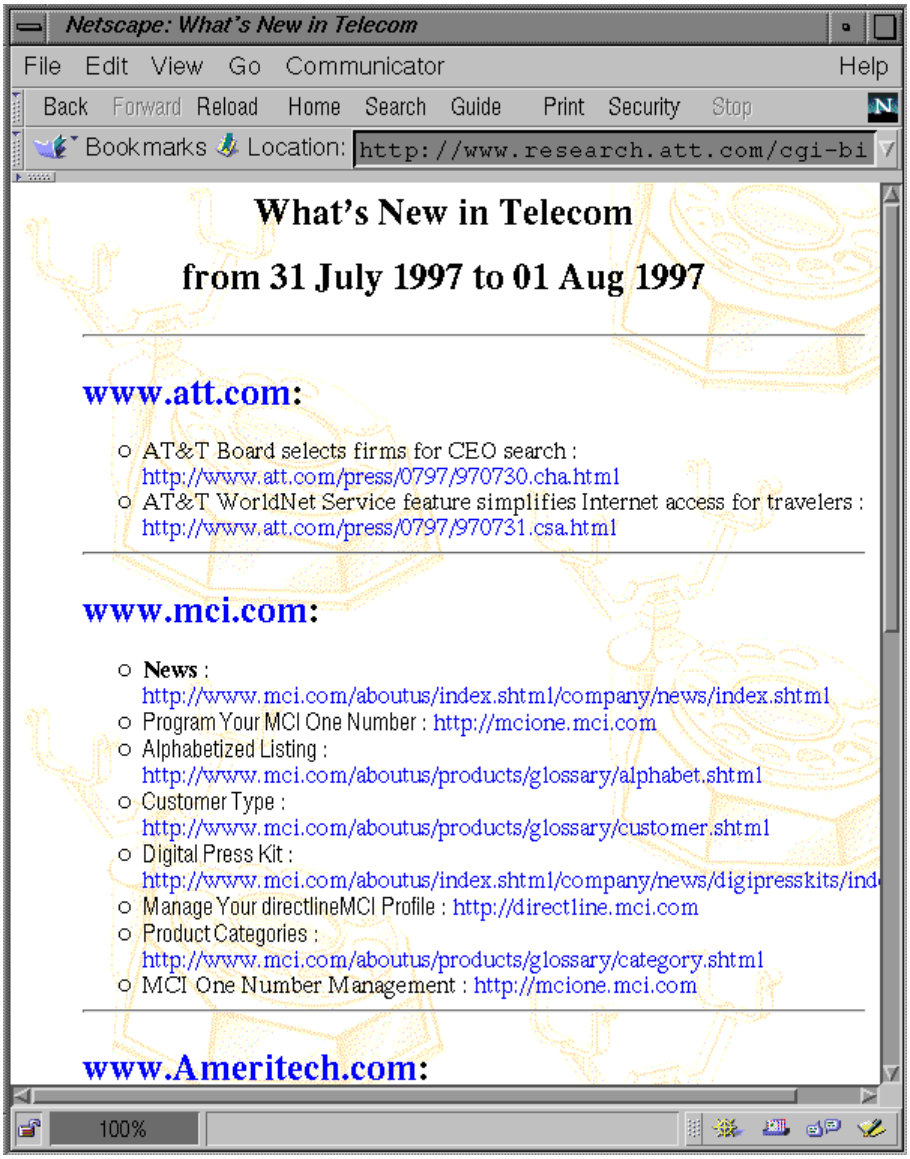


Figure 7: New website links detected on 1 August 1997 on selected telecommunications companies.

The database supports queries to list all URLs that need to be checked, all URLs that are out of date with respect to a particular user, and many others.

3.5.2 Entity-Relationship Database

Ciao uses an "abstractor" to convert different forms of data into an internal representation that can be graphed. Each abstractor is specific to a domain, such as C program source or HTML. The internal representation is an Entity-Relationship [3] database that can be processed by CQL [14]. In the case of HTML, each entity is a URL, which can point to a web page, image file, email address (`mailto:`), CGI script, and so on. The relationships reflect which anchors are included in each page, and which page an anchor links to. The text of a text anchor (such as "Inside AT&T Labs" in Figure 6) or the complete URL of an image anchor (such as `http://www.att.com/attlabs/images/index_left.gif`) is also recorded in the database.

By abstracting two versions of an HTML document, WebCiao can create a difference database to determine any anchors or images that have been inserted or deleted. In addition, WebCiao can query the AIDE relational database, as discussed in Section 3.4, to determine which pages are new to the user. It then colors nodes accordingly.

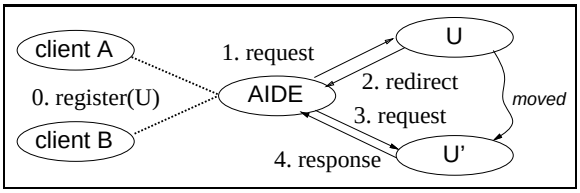
4 Lessons Learned

4.1 Dynamic Nature of the www

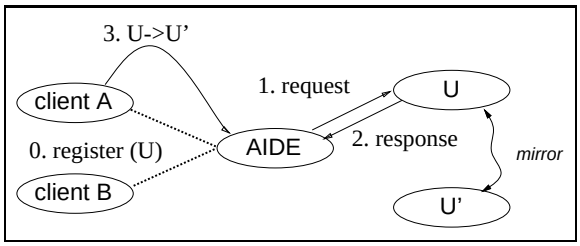
The motivation behind AIDE has been, of course, the fact that data within the WWW changes at different rates and in different ways. However, updates to content within pages are only one type of change. Pages themselves get deleted or moved with surprising frequency; even when they are available, the semantics of accessing them may change. For example, a past run of the script to check the WWW for new pages attempted to retrieve 323 pages, of which 289 (89%) were available, 20 were either temporarily unavailable or on servers that no longer existed and did not respond to HTTP requests, and 14 pages had other errors: 11 of these no longer existed on the server, 2 were rejected (code 406) because the request (sent by the AIDE script) did not specify appropriate MIME data types in its `Accept` header, and 1 required authorization to connect that was not required when the URL was added to AIDE. Of the 289 available pages, 13 (4.5%) were redirected from the URL specified by the user.

One would expect a user who is interested in changes to a page to also be interested in its unavailability, and possibly interested in knowing that it has relocated. AIDE reports the first occurrence of an error on a page, via email, to each user who has registered an interest in it. In the report of changed pages that the user accesses on the WWW, AIDE reports the number of pages a user tracks that have encountered an error in the most recent attempt to access them, with a hyperlink to go to a full report on HTTP errors.

When an error indicates that a page is not found, the user may locate a new copy of the page and wish to inherit the history from its former location. Currently, there is no support for actively renaming a URL, keeping its version history while starting to retrieve new versions from a different location. The paradigm to support such functionality is unclear, especially if a URL is tracked by multiple users. As shown in Figure 8(a), if users A and B track page U and the AIDE system gets a redirection from U to U', then it is trivial to associate U and U'. However, it would be better to use the same aliasing mechanism to support mirrors or other cases where different URLs resolve to the same document (Figure 8(b)), without giving individuals the opportunity to affect the views of other users. Thus, if instead of the server for U notifying AIDE that U has moved to



(a) Server-specified URL relocation. AIDE can trust the redirection and apply it to all users.



(b) Client-specified URL relocation. If one client tells AIDE that a page has moved, perhaps to access a different mirror, other users should not be affected.

Figure 8: Examples of URL redirection. In both figures, clients A and B register an interest in URL U, which AIDE tracks. In the left figure, the server for U redirects AIDE to U' while in the right figure, A tells AIDE to start tracking U' while inheriting the history of U.

U', user A wants to start tracking U' while inheriting the history of U, the system should support the alias without affecting user B.

Of course, if a page is truly unavailable, the user is able to delete it from the list of pages to track. Deletions require manual intervention because the user must be able to determine when an error is permanent and whether other changes, such as registering a new URL in place of an unavailable page, may be appropriate.

The WWW is dynamic in other ways as well. Many of the features of HTML that are taken for granted, such as frames, were introduced after AIDE (and particularly *HtmlDiff*) were first developed. There will always be newer features that the latest browsers can take advantage of, but which must be added to other tools such as AIDE once they gain acceptance.

4.2 Interesting Changes

Detecting interesting changes, or even detecting changes at all, is harder than simply looking at timestamps. Problems include:

Modification timestamps. Many pages do not provide a modification timestamp. These pages are generally dynamic documents, the results of CGI scripts or server-side includes, as opposed to static documents with a modification time stored in the server's file system. In these

cases AIDE must take a checksum of the entire document, which has more overhead and is typically less reliable than a timestamp: trivial changes can cause a checksum to change. We alleviate this problem by invoking *HtmlDiff* on those pages that are automatically archived, and using its output to determine whether the changes affected the content before flagging a page as modified.

A small number of pages that *do* provide modification timestamps result in apparent changes without actual modifications (i.e., the timestamp changes while the body does not). Using just a `HEAD` request, AIDE might decide that a change has occurred, but again the automatic difference generation serves as a double-check that the page changed. Once a `HEAD` request indicates a change, a `GET` request retrieves the entire page. If it has not changed, the page is flagged so that future checks always retrieve the body and compare its latest checksum against the previous version.

Daily changes, images, and "Heisenbergs." Some pages change every day, or appear to. Some of these change because the act of retrieving them updates them (those that count the number of "hits"): the act of observing them changes their behavior, similar to the impossibility in quantum physics of observing an event without affecting it whatsoever. Others, such as the AT&T home page (`http://www.att.com/`), have content changes such as embedded dates. Server-side imagemaps pose another problem, since changes to the behavior of the imagemap are undetectable unless AIDE submits every possible set of coordinates to find the corresponding URL.

Netmind, which runs the URL-minder service [26], at one point announced a `URL-minder-ignore` tag. Text contained within the tag will not be compared by URL-minder, so content providers can voluntarily exclude such things as timestamps and hit counts from consideration. While AIDE can support this tag, a mechanism for end users to filter uninteresting changes, rather than content providers, would extend this functionality to all pages on the WWW.

Forms, Java and Javascript Just as AIDE does not recursively follow images to determine if a page has changed, it does not handle forms using POST, nor applets or other embedded objects. (In fact, the Java applet security model [6] suggests that if AIDE archives or marks up a page, serving it from a different site from the original, its applets may be inaccessible.) Executable content that is embedded within a page (i.e., Javascript) is treated like any other content. Over time, the increase in dynamic and executable documents will likely require extensions to AIDE to track their changes. In particular, applets (currently ignored) should be archived or checksummed to determine when they have changed, even if AIDE has no mechanism to indicate *how* they have changed.

Quality and extent of changes Given that a page actually has changed, a user might want information about how it has changed before actually visiting it. Currently, the only information in the textual "what's new" report is the modification timestamp and the user's own priority for the page (discussed above and in the following subsection). Additional information might include:

- The number of changes reported by *HtmlDiff*.
- The number of links added or deleted. This might be especially useful when tracking pages that themselves enumerate other pages, such as in the WWW Virtual Library [36] or Yahoo [40]. The "Website News" pages described above do list information about changes to links.

- The fraction of text that has been modified.

The user can also use WebCiao to get a graphical depiction of the changes, obtaining for instance an overview of which pages have had anchors added since they were last seen.

4.3 Managing Many URLs

The prioritization method described in Section 2.2 was an improvement over strict timestamp ordering when dealing with a large number of URLs. In addition, there is a CGI interface to permit a user to make changes to many URLs at once, such as changing a URL's priority, editing the string associated with an anchor, or deleting URLs that are no longer of interest. The form can also be used to select a number of URLs, using checkboxes, for which values will be set as a group. For instance, one could set the priorities of several related URLs to the same value, perhaps also setting the frequency to check them, without having to enter the values for each URL individually.

However, some improvements and enhancements could be made:

- Users might prefer to group URLs by topic, rather than by priority across all URLs. Changes to URLs could be highlighted in one's bookmark list, as with Smart Bookmarks [13], or in some other order specified by the user. This grouping would also have the advantage of tying together URLs that are tracked recursively but were not explicitly registered by the user.

- Static priorities are helpful for bringing pages of particular importance to the attention of a user and for grouping related pages. However, once the user sets the priority for a URL, it is time-consuming to revise the priority later. We are considering a *dynamic* prioritization scheme, in which URLs are sorted by a function of the static priority specified by the user and the number of times the user has seen the change reported without viewing it. The system could learn over time that a particular page is ignored despite its priority, and list it later in the report. It might also track changes to it less frequently than requested by the user.

- Some pages change more dynamically than others. If the user's specification for the minimum frequency to check for updates were treated as a hint rather than a requirement, the system could learn about URLs over time. If a page hasn't changed in a month, perhaps checking it daily is overkill. The time-based protocols used for cache consistency [16] are especially applicable for tracking modifications for individual users.

- The form to edit all URLs at once is an HTML table, so that columns are aligned. However, Netscape (both versions 2.0 and 3.0) does not handle extremely large tables well, causing there to be long delays when rendering the table or scrolling it. The typical method that WWW servers use for handling large amounts of data is to display some of it at a time, with links to see other parts, but that would partially defeat the purpose of being able to set values for many URLs at once. A better user interface might permit the user to drag and drop entries, select a region rather than many individual checkboxes, use database queries to select and update entries, and so on.

5 Ongoing Issues and Extensions

In addition to the many extensions discussed above, three areas are of particular importance and concern. Section 5.1 discusses copyright issues, Section 5.2 considers security and privacy, and

Section 5.3 describes how AIDE might be integrated with other tools and protocols.

5.1 Copyright

The implications of copyright law on a system such as AIDE are somewhat unclear. Copyright comes up in two respects: archiving pages, and marking up changes. If *archival* means that copies of pages are kept permanently on a central server, then the right of the copyright holders may be infringed unless they grant explicit permission for the archival. On the other hand, saving an old version of a page is not significantly different from a proxy-caching server keeping a copy of a page until it determines that the page is out of date or the page's expiration timestamp is reached. Thus, AIDE could potentially cache the most recent version of a page unless and until it expires. In addition, an individual may be able to archive pages for personal use under the "fair use" doctrine.

A second issue is the output of *HtmlDiff*. Clearly, taking the current version of a page and marking it up to include text from a previous version, with struck-out text, is a derivative work that may require permission of the copyright holder. It is not so clear that simply marking up new or changed text in a different font would be a derivative work, since it affects appearance without affecting content and browsers are free to handle markup for fonts any way they choose. Again, there may be a distinction between a service that highlights the changes and individuals who do so for personal use. In any event, modifying *HtmlDiff* to use frames to display the old and new versions side-by-side (as Web Integrity does [25]) rather than marking up the new version should avoid the "derivative work" issue completely.

Finally, one can examine other systems, such as the Internet Archive [21] and DejaNews [7], which perform similar functions. The Internet Archive relies on users to deny access explicitly, via a `robots.txt` file [31], or a `Pragma: no-cache` directive, if they do not want their content archived. AIDE could use the same conventions, but the legality of that assumption is uncertain, and it applies to archival but not differencing (a content provider might permit caching, and implicitly archival, while not wanting its differences to be highlighted).

DejaNews archives Usenet postings indefinitely and requests that users put a special header in their postings if they do *not* want them archived. AIDE could archive pages as long as they do not explicitly request that they are not archived, but this policy apparently does not conform to copyright law. A more conservative approach would be to request that content providers explicitly grant permission to archive via a comment in an HTML document. The PICS protocol [29] may ultimately be used in this fashion as well. Explicit permission, however, returns to the same problem of requiring actions on the part of the content provider and therefore probably reducing the set of URLs available to the AIDE system by orders of magnitude.

Of course, in the context of an intranet or other organization where the service performing the archival and differencing also owns the content, copyright is a non-issue. Therefore, AIDE and similar tools may be widely used within intranets long before the intellectual property issues for the Internet have been settled.

5.2 Security and Privacy

When a central service such as AIDE or URL-Minder gets a request from a user to track a URL, the user sacrifices a degree of privacy to the provider of the service. The service provider (SP) can determine which users are interested in which URLs, how often they use the service, and many other potentially sensitive pieces of information. The problem is exacerbated when a page itself is sensitive, since the user may not want to give the SP permission to access the page directly, or

to see its contents. Currently, AIDE allows users to associate a login name and password with a URL, and then provides that information when asked for simple HTTP authentication. Note that having a service such as AIDE retrieve protected pages on behalf of a client may violate licensing agreements between the user and the content provider.

The simplest and safest approach to provide AIDE's functionality while ensuring privacy is to let individual users run AIDE directly—in effect, moving back to the original decentralized system, including a decentralized archive. A user could potentially use the centralized AIDE to track nonsensitive pages and a private copy to track sensitive ones, then merge the results—perhaps via a Java applet. In practice, though, experience with a system that requires individual installation and maintenance would suggest that most people will simply not track those sensitive pages in the first place.

5.3 Integration with Other Tools

Since AIDE was first deployed, changes to the rest of the WWW have suggested ways in which AIDE might be extended and integrated. First, AIDE arose from a desire to provide certain functionality without relying on support from content providers. However, support for versions may become more commonplace. In particular, the proposed standard for HTTP 1.1 includes support for providing a `Content-Version` along with a page, and for clients to upload changes to the page with a `PATCH` directive and a `Derived-Version` header specifying the version against which the patch should be applied [12]. While this support applies to collaborative environments in which people update a common base of pages on a server, it could be extended to allow servers to provide multiple versions and clients such as AIDE to request specific versions rather than archiving them themselves. Of course, as discussed above in Section 3.3, versions provided by a content provider may not have an appropriate granularity for AIDE.

Second, the differences between versions of a page are useful not only to highlight changes for the user, but also to improve performance [39, 24]. If either a content provider or an intermediary such as a proxy-caching server makes the differences between versions of a page available, a system like AIDE should be able to take advantage of that functionality; for example, it might register a call-back to be invoked when a new version is available. Conversely, a "local" version archive could enable an intermediary to obtain past versions of pages that it does not currently cache, against which it could provide deltas to update a client to a more recent version. (The latter scenario might apply when access to the AIDE version repository is much faster than simply transferring the entire page to the client; this seems unlikely in general except for extremely slow links between the client and the proxy-caching server.)

Third, notification about updates should be integrated with "push" technology like Pointcast [28] or Castanet [22]. Push technology should provide a good intermediate step between email and user-initiated reports. Also, Castanet supports multiple versions of content, which might be integrated directly with a tool like AIDE.

6 Applications

AIDE has always been viewed as a system to permit individuals to track changes to pages of personal interest and to view the ways in which those pages change over time. However, there are a number of other uses for AIDE, both already in practice or as potential future uses. Examples include:

Shared communities of interest. If multiple users share interests, a single report can be generated to track pages of interest to all those users. Updates can be sent via email to a mailing list consisting of those users as well. For instance, we have a "what's new in our lab" collection of pages that members of the lab can subscribe to via email or access over the WWW. The "Website News" report on changes to pages of interest to the telecommunications industry, mentioned above in Section 3.4, is another example.

Collaborative editing. A colleague in Lucent Technologies implemented a shared WWW-based "blackboard" that implemented its own report of the most recently changed pages, but used *HtmlDiff* to highlight recent changes to a page on request [1].

Disconnected or low-speed access. When a user is connected over a slow connection, or intermittently connected, the system can provide support for refreshing cached copies of pages that have changed. In addition to techniques for propagating the updates to cached pages in order to save bandwidth [39, 24], one could use a system like AIDE as a clearinghouse to be notified of which pages have changed. For instance, one might see a report listing 100 recently changed pages, click on checkboxes associated with 20 of them, and submit the form to institute an efficient (delta-based) retrieval of the changes to each page, including *HtmlDiff* output. One could then review the changes at one's leisure, perhaps disconnected while traveling.

7 Discussion

AIDE has been in regular use within AT&T for over two years, though it has yet to establish a significant user community. While we initially blamed its lack of use on the complexity of running a personal tool to track changes, moving to a central server resulted in only a few additional users. Now, we attribute the lack of use to people's priorities: even if there are a few pages that a user might like to track, there is no compelling reason to invest the time to learn how to use AIDE when there are so many other demands on one's time. There is also the issue of the increased complexity of managing a list of URLs to track in addition to a bookmark file. We have addressed this in part by providing a mechanism to upload the URLs in an arbitrary WWW page as if they were entered by hand, so one could get a "jump start" by using a bookmark file or other source of URLs; however, once uploaded, this page is not itself tracked to ensure a one-to-one correspondence (e.g., deleting URLs from AIDE once they are deleted from that file).

Many people outside AT&T have requested AIDE, or at least *HtmlDiff*. In addition, the "Website News" report is available outside AT&T and retrieved by a significant number of users,[5] though it just reports structural changes such as new links without using *HtmlDiff*. Thus we have little doubt that AIDE could establish a following if it were released in its full functionality, but the legal issues discussed in Section 5.1 currently preclude this action. AT&T has released *HtmlDiff* for noncommercial use, and we are considering a wider release for use in intranets or other environments where copyright issues are not a concern. We are also exploring standards in the HTTP community that might allow content providers to specify the extent to which a page can be archived and/or marked up via a tool such as *HtmlDiff*.

In addition, the easier AIDE is to use, and the better it becomes at managing the kind of information overload that WWW users are susceptible to, the better a chance it will have to establish a user community once the legal issues are resolved.

[5] There were more than 1500 visits during the period of 1 January 1997 to 15 April 1997. See `http://www.research.att.com/~chen/web-demo/`

Availability

Although AIDE is not available outside AT&T, the *HtmlDiff* component has been released for non-commercial use. See `http://www.research.att.com/cgi-bin/access.cgi/as/vt/ext-software/www-ne-license.cgi?form.aide.source` for information.

Acknowledgments

First, we wish to thank the anonymous referee from *World Wide Web*, who provided many suggestions for improvement. Over the course of the development of AIDE, many others have provided assistance with tools or comments on earlier papers. Rick Greer provided considerable assistance with the use of Daytona, and Emden Gansner did likewise with SML, used by *HtmlDiff*. Gene Nelson and Stuart Mayer, in the AT&T Intellectual Property Division, have been helpful in understanding the legal issues involving copyright. Brooks Cutter wrote the Perl script (*w3new*) from which the AIDE modification tracking evolved. Elliot Berk wrote a Java version of *HtmlDiff* as a student at Princeton University, which may be incorporated into AIDE at a future date. Gaurav Banga, Tony DeSimone, Doug Monroe, Sandeep Sibal, David Thorn, and the anonymous reviewers for USENIX'96 and WWW5 provided comments on earlier texts from which this paper is derived. Thanks also to the many people within AT&T and Lucent Technologies who used the system and provided feedback.

Some of the material in this paper was previously reported in earlier conferences [1, 9, 10, 11].

References

[1] Thomas Ball and Fred Douglis. An internet difference engine and its applications. In *Digest of Papers, COMPCON '96*, pages 71–76, February 1996.

[2] David Chappell. *Understanding ActiveX and OLE*. Microsoft Press, September 1996.

[3] P. P. Chen. The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[4] Yih-Farn Chen, Glenn S. Fowler, Eleftherios Koutsofios, and Ryan S. Wallach. Ciao: A Graphical Navigator for Software and Document Repositories. In *International Conference on Software Maintenance*, pages 66–75, 1995. See also `http://www.research.att.com/~ciao`.

[5] Yih-Farn Chen and Eleftherios Koutsofios. WebCiao: A Website Visualization and Tracking System. In *Proceedings of WebNet97*, Toronto, Ontario, Canada, November 1997. An extended version appears as AT&T Labs – Research TR 97.17.1 and is available via `http://www.research.att.com/~chen/webciao`.

[6] Drew Dean, Ed Felten, and Dan Wallach. Java security: From HotJava to Netscape and beyond. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 190–200, Los Alamitos, CA, 1996. IEEE Press.

[7] DejaNews. `http://www.dejanews.com/`, December 1996.

[8] Digital Equipment Corporation. AltaVista. `http://www.altavista.digital.com`. Random URL selection at `http://www.altavista.digital.com/cgi-bin/query?pg=s&target=0`, January 1997.

[9] Fred Douglis. Experiences with the at&t internet difference engine. In *Proceedings of the 22nd International Conference for the Resource Management & Performance Evaluation of Enterprise Computing System (CMG96)*, December 1996. available on CD-ROM.

[10] Fred Douglis and Thomas Ball. Tracking and viewing changes on the Web. In *Proceedings of 1996 USENIX Technical Conference*, pages 165–176, San Diego, CA, January 1996.

[11] Fred Douglis, Thomas Ball, Yih-Farn Chen, and Eleftherios Koutsofios. WebGUIDE: Querying and navigating changes in Web repositories. In *Proceedings of the Fifth International World Wide Web Conference*, pages 1335–1344, Paris, France, May 1996.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, et al. RFC 2068: Hypertext transfer protocol — HTTP/1.1, January 1997.

[13] First Floor Software. `http://www.firstfloor.com/`, August 1997.

[14] Glenn Fowler. cql – A Flat File Database Query Language. In *Proceedings of the USENIX Winter 1994 Conference*, pages 11–21, January 1994.

[15] Rick Greer. All about Daytona. AT&T Bell Laboratories internal document, December 1994.

[16] James Gwertzman and Margo Seltzer. World-Wide Web cache consistency. In *Proceedings of 1996 USENIX Technical Conference*, pages 141–151, San Diego, CA, January 1996.

[17] D. S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(4):664–675, October 1977.

[18] J. W. Hunt and M. D. McIlroy. An algorithm for differential file comparison. Technical Report Computing Science TR #41, Bell Laboratories, Murray Hill, N.J., 1975.

[19] Informant. `http://informant.dartmouth.edu/`, December 1996.

[20] Javasoft. Java. `http://www.javasoft.com/`, December 1996.

[21] Brewster Kahle. Preserving the internet. *Scientific American*, March 1997. Also available as `http://www.sciam.com/0397issue/0397kahle.html`.

[22] Marimba, Inc. Castanet. `http://www.marimba.com/datasheets/castanet-ds.html`, August 1997.

[23] R. Milner, editor. *The Definition of Standard ML: Revised*. MIT Press, June 1997.

[24] Jeffrey Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *Proceedings of ACM SIGCOMM'97 Conference*, pages 181–194, September 1997. An extended version appears as Digital Equipment Corporation Western Research Lab TR 97/4, July, 1997, available as `http://www.research.digital.com/wrl/techreports/abstracts/97.4.html`.

[25] Mortice Kern Systems (MKS), Inc. Web Integrity. `http://www.mks.com/solution/ie/`, 1997.

[26] Url-minder. `http://www.netmind.com/URL-minder/URL-minder.html`, December 1996.

[27] M. Newbery. Katipo. `http://www.vuw.ac.nz/~newbery/Katipo.html`, December 1996.

[28] Pointcast, Inc. `http://www.pointcast.com/`, 1997.

[29] Paul Resnick and James Miller. PICS: Internet access controls without censorship. *Communications of the ACM*, 39(10):87–93, October 1996.

[30] Ronald L. Rivest. The MD5 message-digest algorithm. Internet Request for Comments, April 1992. RFC 1321.

[31] A standard for robot exclusion. `http://web.nexor.co.uk/mak/doc/robots/norobots.html`, December 1995.

[32] Margo Seltzer and Ozan Yigit. A new hashing package for UNIX. In *USENIX Conference Proceedings*, pages 173–184, Dallas, TX, January 21-25 1991. USENIX.

[33] Surfbot. `http://www.surflogic.com/products.html/`, March 1997. formerly known as WebWatch.

[34] Lisa Sweet. Pushing it to the limit. *ZD Internet Magazine*, March 17 1997. Also available as `http://www8.zdnet.com/zdimag/content/anchors/199703/17/1.html`.

[35] W. Tichy. RCS: a system for version control. *Software—Practice & Experience*, 15(7):637–654, July 1985.

[36] The World Wide Web virtual library. `http://www.w3.org/hypertext/DataSources/bySubject/Overview2.html`, December 1996.

[37] The World Wide Web virtual library on mobile computing. `http://snapple.cs.washington.edu/mobile/`, December 1996.

[38] Webcopy. `http://www.inf.utfsm.cl/~vparada/webcopy.html`, March 1996.

[39] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for World-Wide Web documents. In *Proceedings of ACM SIGCOMM'96 Conference*, pages 293–305, August 1996. Also available as `http://ei.cs.vt.edu/~succeed/96WAASF1/`.

[40] Yahoo. `http://yahoo.com/`, December 1996.