Tracking and Viewing Changes on the Web*

Fred Douglis *Thomas Ball*
AT&T Bell Laboratories

1996 USENIX Technical Conference.

Abstract

We describe a set of tools that detect when World-Wide-Web pages have been modified and present the modifications visually to the user through marked-up HTML. The tools consist of three components: *w3newer*, which detects changes to pages; *snapshot*, which permits a user to store a copy of an arbitrary Web page and to compare any subsequent version of a page with the saved version; and *HtmlDiff*, which marks up HTML text to indicate how it has changed from a previous version. We refer to the tools collectively as the *AT&T Internet Difference Engine* (AIDE). This paper discusses several aspects of AIDE, with an emphasis on systems issues such as scalability, security, and error conditions.

1 Introduction

Use of the World-Wide-Web (W^3) has increased dramatically over the past couple of years, both in the volume of traffic and the variety of users and content providers. The W^3 has become an information distribution medium for academic environments (its original motivation), commercial ones, and virtual communities of people who share interests in a wide variety of topics. Information that used to be sent out over electronic mail or USENET, both active media that go to users who have subscribed to mailing lists or newsgroups, can now be posted on a W^3 page. Users interested in that data then visit the page to get the new information.

The URLs of pages of interest to a user can be saved in a "hotlist" (known as a bookmark file in Netscape), so they can be visited conveniently. How does a user find out when pages have changed? If users know that pages contain up-to-the-minute data (such as stock quotes), or are frequently changed by their owners, they may visit the pages often. Other pages may be ignored, or browsed by the user only to find they have not changed.

In recent months, several tools have become available to address the problem of determining when a page has changed. The tool with perhaps the widest distribution is "Smart Bookmarks," from First Floor Software, which has been incorporated into Netscape for Windows as "Netscape SmartMarks." Users have items in their bookmark list automatically polled to determine if they have been modified. In addition, content providers can optionally embed *bulletins* in their pages, which allow short messages about a page to be displayed in a page that refers to it. Other tools for learning about modifications are discussed in the next section.

Each of the current tools suffers from a significant deficiency: while they provide the user with the knowledge that the page has changed, they show little or no information about *how* the page has changed. Although a few pages are edited by their maintainers to highlight the most recent changes, often the modifications are not prominent, especially if the pages are large. Even pages with special highlighting of recent changes (including bulletins) are problematic: if a user visits a page frequently, what is "new" to the maintainer may not be "new" to the user. Alternatively, a user who visits a page infrequently may miss changes that the maintainer deems to be old. Changes deemed uninteresting by a document's author and omitted from a change summary or bulletin actually may be of great interest to readers. Finally, the really major change might be the item that was deleted or modified, rather than added. Such items are unlikely to be found on a "What's New?" page.

We have developed a system that efficiently tracks when pages change, compactly stores versions on a per-user basis, and automatically compares and presents the differences between pages. The *AT&T Internet Difference Engine* (AIDE) provides "personalized" views of versions of W^3 pages with three tools. The first, *w3newer*, is a derivative of one of the existing modification tracking tools, *w3new*, discussed in the next section. It periodically accesses the W^3 to

find when pages on a user's hotlist have changed. The second, *snapshot*, allows a user to save versions of a page and later use a third tool, *HtmlDiff*, to see how it has changed. *HtmlDiff* automatically compares two HTML pages and creates a "merged" page to show the differences with special HTML markups.

While AIDE can help arbitrary users track pages of interest, it can be of particular use in a collaborative environment. One example of a collaborative environment on the W^3 is the *WikiWikiWeb* [3], which allows multiple users to edit the content of documents dynamically. There is a RecentChanges page that sorts documents by modification date. Typically content is added to the end of a page and it is not difficult to determine visually what changes occurred since the last visit. However, content can be modified anywhere on the page, and those changes may be too subtle to notice. Within AT&T, a clone of *WikiWikiWeb*, called *WebWeaver*, stores its own version archive and uses *HtmlDiff* to show users the differences from earlier versions of a page. While the differences are not currently customized for each user, that would be a natural and simple extension. Similarly, integrating the RecentChanges page into the AIDE report of what's new would avoid having to query multiple sources to determine what has recently changed.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 elaborates on our extensions to *w3new*, Section 4 describes the *snapshot* versioning tool, and Section 5 describes *HtmlDiff* in detail. Section 6 describes the integration of the three tools into AIDE. Section 7 relates early experiences with the system. Section 8 discusses other extensions and uses of AIDE, and Section 9 concludes.

2 Related Work

2.1 Tracking Modifications

There has been a great deal of interest lately in finding out when pages on the W^3 have changed. As mentioned above, Smart Bookmarks checks for modifications and also supports an extension to HTML to allow a description of a page, or recent changes to it, to be obtained along with other "header" information such as the last modification date. These bulletins may be useful in some cases but will not help in others. For instance, W^3 Virtual Library pages contain many links to other pages within some subject area and have a number of links added at a time; a bulletin that announces that "10 new links have been added" will not point the user to the specific locations in the page that have been edited. Also, it suffers from the problem of timeliness mentioned in the introduction.

Smart Bookmarks have the advantage of being integrated directly with a user's hotlist, making a visual indication of what has changed available without resorting to a separate page. Several other tools read a user's hotlist and generate a report, in HTML, of recently changed pages. Examples include *webwatch* [16], a product for Windows; *w3new* [4], a public-domain *Perl* script that runs on UNIX; and *Katipo* [13], which runs on the Macintosh.

Another similar tool, *URL-minder* [19], runs as a service on the W^3 itself and sends email when a page changes. Unlike the tools that run on the user's host and use the "hotlist" to determine which URLs to check, *URL-minder* acts on URLs provided explicitly by a user via an HTML form. Centralizing the update checks on a W^3 server has the advantage of polling hosts only once regardless of the number of users interested. However, the need to send URLs explicitly through a form is cumbersome.

There are two basic strategies for deciding when a page has changed. Most tools use the HTTP HEAD command to retrieve the Last-Modified field from a W^3 document, either returning a sorted list of all modification times or just those times that are different from the browser's history (the timestamp of the version the user presumably last saw). *URL-minder* uses a checksum of the content of a page, so it can detect changes in pages that do not provide a Last-Modified date, such as output from Common Gateway Interface (CGI) scripts. *W3new* (and therefore *w3newer*) requests the Last-Modified date if available; otherwise, it retrieves and checksums the whole page.

These tools also vary with respect to frequency of checking and where the checks are performed. Most of the tools automatically run periodically from the user's machine. All URLs are checked each time the tools run, with the possible exception of *URL-minder*, which runs on an Internet server and checks pages with an arbitrary frequency that is guaranteed to be at least as often as some threshold, such as a week. (*URL-minder*'s implementation is hidden behind a CGI interface.)

2.2 Version Repositories

As discussed below, we use the Revision Control System (RCS) [18] to compactly maintain a history of documents, addressed by their URLs. A CGI interface to RCS allows a user to request a URL at a particular date, from anywhere on the W^3. This is similar in spirit to the "time travel" capability of file systems such as 3DFS [5] that transparently allow access to files at arbitrary dates. A slight twist on the versioning is that we wish to track the times at which each user checked in a page, even if the page hasn't changed between

check-ins of that page by different users. This is accomplished outside of RCS by maintaining a per-user control file, allowing quick access to a user's access history.

2.3 HTML Differencing

We know of no existing tools that compare HTML pages and present the comparison as HTML. However, there is much closely related work on heuristics for parallel document alignment and similarity measures between documents [2] that we benefit from. Line-based comparison utilities such as UNIX *diff* [10] clearly are ill-suited to the comparison of structured documents such as HTML. Most modern word processing programs have a "revision" mode that track additions and deletions on-line as an author modifies a document, graphically annotating the differences. *HtmlDiff* graphically annotates differences in a similar manner but operates off-line after document creation, using heuristics to determine the additions and deletions.

3 Tracking Modifications

To our knowledge, the tools described in Section 2.1 poll every URL with the same frequency. We modified *w3new*[1] to make it more scalable, as well as to integrate it with the other components of AIDE. Figure 1 gives an example of the report generated by *w3newer*; the meaning of the links on the right-hand side is discussed in Section 6.

W3newer runs on the user's machine, but it omits checks of pages already known to be modified since the user last saw the page, and pages that have been viewed by the user within some threshold. The time when the user has viewed the page comes from the W^3 browser's history. The "known modification date" comes from a variety of sources: a cached modification date from previous runs of *w3newer*; a modification date stored in a proxy-caching server's cache; or the HEAD information provided by *httpd* (the HTTP server) for the URL. If either of the first two sources of the modification date indicate that the page has not been visited since it was modified, then HTTP is used only if the time the modification information was obtained was long enough ago to be considered "stale" (currently, the threshold is one week).

In addition, there is a threshold associated with each page to determine the maximum frequency of direct HEAD requests. If the page was visited within the threshold, or the modification date obtained from the proxy-caching server is current with respect to the threshold, the page is not checked. The threshold can vary depending on the URL, with *perl* pattern matching used to determine what threshold to apply. The first matching pattern is used.

[1] *W3new* was selected because it ran on UNIX and because it was available before the others. If Smart Bookmarks had been available at the time, it would have provided a better starting point.

Comments start with a sharp sign.
perl syntax requires that "." be escaped
Default is equivalent to ending the file with ".*"
Default 2d
file:.* 0
http://www\.yahoo\.com/.* 7d
http://.*\.att\.com/.* 0
http://www\.ncsa\.uiuc\.edu/SDG/Software/-Mosaic/Docs/whats-new\.html 12h
http://snapple\.cs\.washington\.edu:600/-mobile/ 1d
this is in my hotlist but will be different every day
http://www\.unitedmedia\.com/-comics/dilbert/ never

Table 1: An example of the thresholds specified to w3newer. The table is explained in the text.

Table 1 gives an example of a *w3newer* configuration file. Thresholds are specified as combinations of days (d) and hours (h), with 0 indicating that a page should be checked on every run of *w3newer* and never indicating that it should never be checked. In this example, URLs are checked every two days by default, but some URLs are overridden. Local files are checked upon every execution, since a *stat* call is cheap, and anything in the att.com domain is checked upon every execution as well. Things on *Yahoo* are checked only every seven days in order to reduce unnecessary load on that server, since the user doesn't expect to revisit *Yahoo* pages daily even if they change. Dilbert is never checked because it will always be different.

3.1 System Issues

Cache Consistency

Determining when HTTP pages have changed is analogous to caching a file in a distributed file system and determining when the file has been modified. While file systems such as the Andrew File System [9] and Sprite [12] provide guarantees of cache consistency by issuing call-backs to hosts with invalid copies, HTTP access is closer to the traditional NFS approach, in which clients check back with servers periodically for each file they access. Netscape can be configured to check the modification date of a cached page each time it is visited, once each session, or not at all. Caching

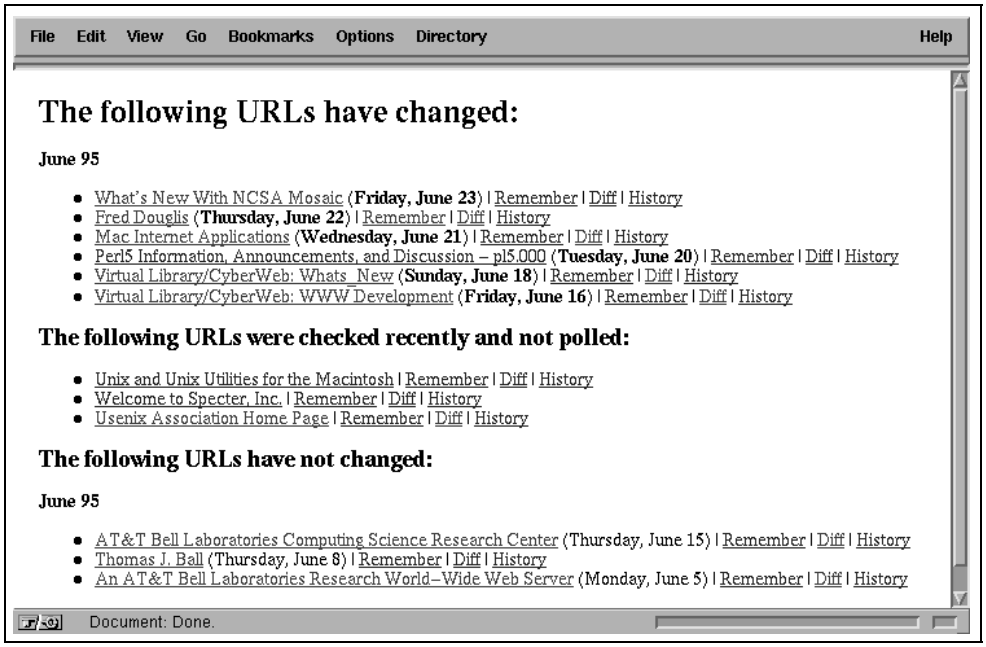

Figure 1: Output of *w3newer* showing a number of anchors (the descriptive text comes from the hotlist). The ones that are marked as "changed" have modification dates after the time the user's browser history indicates the URL was seen. Some URLs were not checked at all, and others were checked and are known to have been seen by the user.

servers check when a client forces a full reload, or after a time-to-live value expires.

Here the problem is complicated by the target environment: one wishes to know not only when a currently viewed page has changes, but also when a page that has not been seen in a while has changed. Fortunately, unlike with file systems, HTTP data can usually tolerate some inconsistency. In the case of pages that are of interest to a user but have not been seen recently, finding out within some reasonable period of time, such as a day or a week, will usually suffice. Even if servers had a mechanism to notify all interested parties when a page has changed, immediate notification might not be worth the overhead.

Instead, one could envision using something like the Harvest replication and caching services [1] to notify interested parties in a lazy fashion. A user who expresses an interest in a page, or a browser that is currently caching a page, could register an interest in the page with its local caching service. The caching service would in turn register an interest with an Internet-wide, distributed service that would make a best effort to notify the caching service of changes in a timely fashion. (This service could potentially archive versions of HTTP pages as well.) Pages would already be replicated, with server load distributed, and the mechanism for discovering when a page changes could be left to a negotiation between the distributed repository and the content provider: either the content provider notifies the repository of changes, or the repository polls it periodically. Either way, there would not be a large number of clients polling each interesting HTTP server. Moving intelligence about HTTP caching to the server has been proposed by Gwertzman and Seltzer [7] and others.

One could also envision integrating the functionality of AIDE into file systems. Tools that can take actions when arbitrary files change are not widely available, though they do exist [15]. Users might like to have a unified report of new files and W^3 pages, and *w3newer* supports the "file:" specification and can find out if a local file has changed. However, *snapshot* has no way to access a file on the user's (remote) file system. Moving functionality into the browser would allow individual users to take snapshots of files that are not already under the control of a versioning system such as RCS; this might be an appropriate use of a browser with client-side execution, such as *HotJava* [17] or recent versions of Netscape.

Error Conditions

When a periodic task checks the status of a large number of URLs, a number of things can go wrong. Local problems such as network connectivity or the status of a proxy-caching server can cause *all* HTTP requests to fail. Proxy-caching servers are sometimes overloaded to the point of timing out large numbers of requests, and a background task that retrieves many URLs in a short time can aggravate their condition. *W3newer* should therefore be able to detect cases when it should abort and try again later (preferably in time for the user to see an updated report).

At the same time, a number of errors can arise with individual URLs. A URL can move, with or without leaving a forwarding pointer. The server for a URL can be deactivated or renamed. Its site may disallow retrieval of this URL by "robots," meaning that the administrator for its site has created a special file, robots.txt, and requested that automated programs such as "web crawlers" not retrieve the URL. Currently, programs only voluntarily follow the "robot exclusion protocol" [14], the convention that defines the use of robots.txt. Although *w3newer* currently obeys this protocol, it is not clear that it should, at least for URLs the user would other wise access directly periodically.

Finally, automatic detection of modifications based on information such as modification date and checksum can lead to the generation of "junk mail" as "noisy" modifications trigger change notifications. For instance, pages that report the number of times they have been accessed, or embed the current time, will look different every time they are retrieved.

W3newer attempts to address these issues by the following steps:

- If a URL is inaccessible to robots, that fact is cached so the page is not accessed again unless a special flag is set when the script is invoked.
- Another flag can tell *w3newer* to treat error conditions as a successful check as far as the URL's timestamp goes, so that a URL with some problem will be checked with the same frequency as an accessible one. In general, however, it seems that errors are likely to be transient, and checking the next time *w3newer* is run is reasonable.
- When a URL is inaccessible, an error message appears in the status report, so the user can take action to remove a URL that no longer exists or repeatedly hits errors.

In addition, *w3newer* could be modified to keep a running counter of the number of times an error is encountered for a particular URL, or to skip subsequent URLs for a host if a host or network error (such as "timeout" or "network unreachable") has already occurred. Addressing the problem of "noisy" modifications will require heuristics to examine the differences at a semantic level.

4 Snapshots

In addition to providing a mechanism for determining when W^3 pages have been modified, there must be a way to access multiple versions of a page for the purposes of comparison. This section describes methods for maintaining version histories and several issues that arise from our solution.

4.1 Alternative Approaches

There are three possible approaches for providing versioning of W^3 pages: making each content provider keep a history of all versions, making each user keep this history, or storing the version histories on an external server.

Server-side support Each server could store a history of its pages and provide a mechanism to use that history to produce marked-up pages that highlight changes. This method requires arbitrary content providers to provide versioning and differencing, so it is not practical, although it is desirable to support this feature when the content provider is willing. (See Section 8.1.)

Client-side support Each user could run a program that would store items in the hotlist locally, and run *HtmlDiff* against a locally saved copy. This method requires that every page of interest be saved by every user, which is unattractive as the number of pages in the average user's hotlist increases, and it also requires the ability to run *HtmlDiff* on every platform that runs a W^3 browser. Storing the pages referenced by the hotlist may not be too unreasonable, since programs like Netscape may cache pages locally anyway. There are other external tools such as *warmlist* [20] that provide this functionality.

External service Our approach is to run a service that is separate from both the content provider and the client, and uses RCS to store versions. Pages can be registered with the service via an HTML form, and differences can be retrieved in the same fashion. Once a page is stored with the service, subsequent requests to remember the state of the page result in an RCS "check-in" operation that saves only the differences between the page and

its previously checked-in version. Thus, except for pages that change in many respects at once, the storage overhead is minimal beyond the need to save a copy of the page in the first place.

Drawbacks to the "external service" approach are that the service must remember the state of every page that anyone who uses the service has indicated an interest in and must know which user has seen which version of each page. The first issue is primarily one of resource allocation, and is not expected to be a significant issue unless the service is used by a great many clients on a number of large pages. The second issue is addressed in our initial prototype by using RCS's support for datestamps and requesting a page as it existed at a particular time. In the next version of the system, a set of version numbers is retained for each $<\mathrm{user},\mathrm{URL}>$ combination. This removes any confusion that could arise if the timestamps provided for a page do not increase monotonically and also makes it easier to present the user with a set of versions seen by that person regardless of what other versions are also stored.

Relative links become a problem when a page is moved away from the machine that originally provided it. If the source were passed along unmodified, then the W^3 browser would consider links to be relative to the CGI directory containing the *snapshot* script. HTML supports a BASE directive that makes relative links relative to a different URL, which mostly addresses this problem; however, Netscape 1.1N treats internal links within such a document to be relative to the new BASE as well, which can cause the browser to jump between the *HtmlDiff* output and the original document unexpectedly.

4.2 System Issues

The *snapshot* facility must address four important issues: use of CGI, synchronization, resource utilization, and security/privacy.

CGI is a problem because there is no way for *snapshot* to interact with the user and the user's browser, other than by sending HTML output. (The system does not currently assume the ability of a browser to support Java [11], although moving to Java in the future is possible and might help address some of these issues.) When a CGI script is invoked, *httpd* sets up a default timeout, and if the script does not generate output for a full timeout interval, *httpd* will return an error to the browser. This was a problem for *snapshot* because the script might have to retrieve a page over the Internet and then do a time-consuming comparison against an archived version. The server does not tell *snapshot* what a reasonable timeout interval might be for any subsequent retrievals; instead this is hard-coded into the script. In order to keep the HTTP connection alive, *snapshot* forks a child process that generates one space character (ignored by the W^3 browser) every several seconds while the parent is retrieving a page or executing *HtmlDiff*.

Synchronization between simultaneous users of the facility is complicated by the use of multiple files for bookkeeping. The system must synchronize access to the RCS repository, the locally cached copy of the HTML document, and the control files that record the versions of each page a user has checked in. Currently this is done by using UNIX file locking on both a per-URL lock file and the per-user control file. Ideally the locks could be queued such that if multiple users request the same page simultaneously, the second *snapshot* process would just wait for the page and then return, rather than repeating the work. This is not so important for making snapshots, in which case a proxy-caching server can respond to the second request quickly and RCS can easily determine that nothing has changed, but there is no reason to run *HtmlDiff* twice on the same data.

The latter point relates to the general issue of resource utilization. *Snapshot* has the potential to use large amounts of both processing and disk space. The need to execute *HtmlDiff* on the server can result in high processor loads if the facility is heavily used. These loads can be alleviated by caching the output of *HtmlDiff* for a while, so many users who have seen versions N and $N+1$ of a page could retrieve *HtmlDiff*$(page_N, page_{N+1})$ with a single invocation of *HtmlDiff*. The facility could also impose a limit on the number of simultaneous users, or replicate itself among multiple computers, as many W^3 services do.

Lastly, security and privacy are important. Because the CGI scripts run with minimal privileges, from an account to which many people have access, the data in the repository is vulnerable to any CGI script and any user with access to the CGI area. Data in this repository can be browsed, altered, or deleted. In order to use the facility one must give an identifier (currently one's email address, which anyone can specify) that is used subsequently to compare version numbers. Browsing the repository can therefore indicate which user has an interest in which page, how often the user has saved a new checkpoint, and so on.

By moving to an authenticated system on a secure machine, one could break some of these connections and obscure individuals' activities while providing better security. The repository would associate impersonal account identifiers with a set of URLs and version numbers, and passwords would be needed to access one of these accounts. Whoever administers this fa-

cility, however, will still have information about which user accesses which pages, unless the account creation can be done anonymously.

5 Comparison of HTML pages

In our experience, only a small fraction of pages on the W^3 contain information that allows users to ascertain how the pages have changed—examples include icons that highlight recent additions, a link to a "changelog", or a special "What's New" page. As was mentioned in the introduction, these approaches suffer from deficiencies. They are intended to be viewed by all users, but users will visit the pages at different intervals and have different ideas of "what's new". In addition, the maintainer must explicitly generate the list of recent changes, usually by manually marking up the HTML.

Automatic comparison of HTML pages and generation of marked-up pages frees the HTML provider from having to determine what's new and creating new or modified HTML pages to point to the differences. There are many ways to compare documents and many ways to present the results. This section describes various models for the comparison of HTML documents, our comparison algorithm, and issues involved in presenting the results of the comparison.

5.1 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 Hirshberg's solution to the longest common subsequence (LCS) problem [8] (with several speed optimizations) to compare HTML documents. This is the well-known comparison algorithm used by the UNIX *diff* utility [10]. 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 uppercase; 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). Sentence-breaking markups can only match sentence-breaking markups. They must be identical (modulo whitespace, case, and reordering of (variable,value) pairs) in order to match (see section 5.3 for a discussion of the ramifications of this). 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," 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). 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.

5.2 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 (see, for example, UNIX *sdiff* or SGI's graphical diff tool *gdiff*). Unfortunately, there is no good mechanism in place with current HTML and browser technology that allows such synchronization.

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 is 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. We are exploring other ways to create a merged page.

An example of *HtmlDiff*'s merged-page output appears in Figure 2. Markups are used to highlight old and new material as follows. Two small arrow images are used to point to areas in the document that have changed. A red arrow points to old content and a green arrow points to new content. The arrows are also internal hypertext references to one another, linked in a chain to allow quick traversal of the differences. A banner at the front of the document contains a link to the first difference. Old text is displayed in "struck-out" font using <STRIKE>, which in our experience is rarely used in HTML found on the W^3. Unfortunately, there is no ideal font for showing "new" text. We currently use <STRONG><I>. Ideally, we would like to be able to color code the text to highlight, but this capability is not provided by all browsers.

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.

5.3 Issues and Extensions

Since *HtmlDiff* can parse an HTML document and rectify certain syntactic problems, such as mismatched or missing markups, the only real problem it is likely to encounter is a set of changes that are so pervasive as to make the resulting merged HTML unreadable. For instance, if every other line were changed, then the mixture of unrelated struck-out and emphasized text would be muddled. We are experimenting with methods for varying the degree to which old and new text can be interspersed, as well as thresholds to specify when the

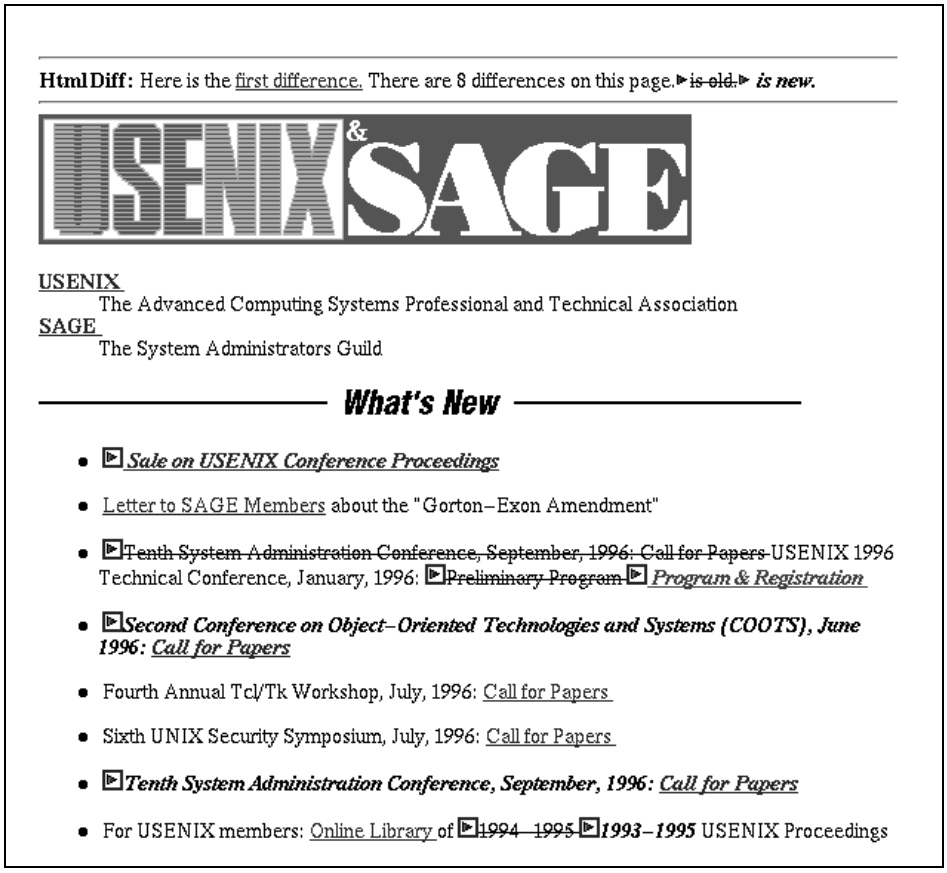

Figure 2: Output of *HtmlDiff* showing the differences between a subset of two versions of the USENIX Association home page (as of 9/29/95 and 11/3/95). Small arrows point to changes, with bold italics indicating additions and with deleted text struck out. The banner at the top of the page was inserted by *HtmlDiff*.

changes are too numerous to display meaningfully.

Currently, *HtmlDiff* is neither "version-aware" nor "web-aware". That is, *HtmlDiff* only compares the text of two HTML pages. It does not compare versions of the entities that the pages refer to, access them, or invoke itself recursively on other referenced pages. This has a number of consequences. The good news is that *HtmlDiff* does not incur the overhead of pulling versions from a repository or sending requests over the W^3 for information. This cost is consumed by *w3newer* and *snapshot*. The bad news is that some differences may be ignored. For example, if the contents of an image file are changed but the URL of the file does not, then the URL in the page will not be flagged as changed. To support such comparison would require some sort of versioning of referenced entities and would also require *HtmlDiff* to have access to the version repositories. Full versioning of all entities would allow interesting comparisons to be done, but would dramatically increase storage requirements. A cheaper alternative would be to store a checksum of each entity and use the checksums to determine if something has changed. We are exploring how to efficiently perform such "smarter" comparisons.

6 Integrating the tools

There are two entry points to AIDE, one through *w3newer* and one through *snapshot*.

Currently, *w3newer* is invoked directly by the user, probably by a *crontab* entry, and generates an HTML document indicating which pages have changed. As shown in Figure 1, *w3newer* associates three links with each document in the hotlist:

Remember Send the URL to the *snapshot* facility, to save a copy of the page. Though the page is retrieved, the RCS *ci* command ensures that it is not

saved if it is unchanged from the previous time it was stored away.

Diff Have the *snapshot* facility invoke *HtmlDiff* to display the changes in a page since it was last saved away by the user.

History Have *snapshot* display a full log of versions of this page, with the ability to run *HtmlDiff* on any pair of versions or to view a particular version directly.

Thus, each page that is reported as "new" can immediately be passed to *HtmlDiff*, and any page in the list can be "remembered" for future use. A user may also interface with *snapshot* directly, via a form, to check-in pages, view differences, or view the version history.

One disadvantage of the current approach is that there is no direct interaction between *w3newer*, *snapshot*, and the W^3 browser. Viewing a page with *HtmlDiff* does not cause the browser to record that the page has just been seen; instead, the browser records the URL that was used to invoke *HtmlDiff* in the first place. Subsequently, *w3newer* uses the obsolete datestamp from the browser and continues to report that the page has been modified more recently than the browser has seen it. As a result, the user must view a page directly as well as via *HtmlDiff* in order to both remove it from the list of modified pages and see the actual differences.

7 Experiences

In the approximately half-year since AIDE was built, we have been using the system regularly ourselves and have attempted to build a user community within AT&T. Personal use has been successful: one of us has recorded over 250 URLs and the other nearly 100. Adoption by others has been harder, and the reason we hear back from prospective users is nearly always the same: it is too time-consuming to install *w3newer* on one's own machine. This reluctance is the primary motivation for moving the functionality of *w3newer* into the AIDE server.

In using AIDE ourselves, we realized another difficulty with the present arrangement: information overload. Merely sorting URLs by most recent modification dates is not satisfactory when the number of URLs grows into the hundreds. Instead, we are moving toward a user-specified prioritization of URLs along the lines of the Tapestry system, which prioritizes email and NetNews automatically [6].

So far, disk usage has not been a problem. There are over 500 URLs archived (many of these are for fixed collections of pages as described below in Section 8.2), and the archive uses under 8 Mbytes of disk storage (an average of 14.3 Kbytes/URL). Three files account for 2.7 Mbytes of that total, and each file is a URL that changes every 1–3 days and is being automatically archived upon each change.

8 Extensions

This section describes some possible extensions to the work already presented. Sections 8.1 and 8.2 discuss some interfaces that are already implemented, while Sections 8.3 and 8.4 presents unimplemented extensions to integrate tracking modifications into the server and to invoke scripts via the HTTP POST protocol.

8.1 Server-side Version Control

The tools described above do not require any changes to arbitrary servers or clients on the W^3. Existing GET and POST protocols are used to communicate with specific servers that save versions of documents and provide marked-up versions showing how they have changed. However, if a server runs *HtmlDiff* and some *perl* scripts, it can provide a direct version-control interface and avoid the need to store copies of its HTML documents elsewhere.

The *perl* scripts we have written provide an interface to RCS [18]. A CGI script (/cgi-bin/rlog) converts the output of *rlog* into HTML, showing the user a history of the document with links to view any specific version or to see the differences between two versions. Another script (/cgi-bin/co) displays a version of a document under RCS control, while still another (/cgi-bin/rcsdiff) displays the differences. If the file's name ends in .html then *HtmlDiff* is used to display the differences, rather than the *rcsdiff* program.

As an example, one might set up a Last-Modified field at the bottom of an HTML document to be a link to the rlog script, with the document name specified as a parameter. After clicking on this unobtrusive field, the user would be able to see the history of the document.

8.2 Fixed Pages

In addition to permitting individuals to archive URLs of interest to them and find out about modifications to those URLs, AIDE can provide a community of users with specialized "What's New" pages that report when any of a fixed set of URLs has been changed. Rather than having users specify when to archive a new version, each page is automatically archived as soon as a change is detected. Then users can easily see the most recent changes to a page using *HtmlDiff*, and they can also use the History feature to see earlier versions they may have missed.

Automatic archival of new versions is useful in this context, but it has the disadvantage of increasing disk space dramatically when the entire contents of the page changes (such as the "What's New in Mosaic" page). When the entire contents are replaced, there is no use for *HtmlDiff*. Automatic archival would still be useful in cases when one wants a way to go back to arbitrary old versions, but in many cases (including this example), the content provider has its own archive.

8.3 Server-side URL Tracking

Currently, *w3newer* runs on the user's machine, so multiple instantiations of the script may perform the same work. Although it runs a related daemon on the same machine as an AT&T-wide proxy-caching server, which returns information about pages that are currently cached on the server and may eliminate some accesses over the Internet, there is insufficient locality in that cache for it to eliminate a significant fraction of requests.

Alternatively, *w3newer* could be run on the set of pages that have been saved by the *snapshot* daemon. Regardless of how many users have registered an interest in a page, it need only be checked once; if changed, the new version could be saved automatically. Then a user could request a list of all pages that have been saved away, and get an indication of which pages have changed since they were saved by the user.

Adding this functionality would be useful, since it would offer economies of scale. In fact, it could be further extended to be integrated with a "web crawler" and track modifications to pages *pointed to* by pages specified by the user. Following links recursively is inappropriate for tools run by every user individually but would be feasible for a centralized service. It would have the advantage of handling multiple styles of pages, for example:

Virtual Library pages Pages with pointers to many other pages scattered throughout the W^3 could have each link followed to indicate when the referenced pages have been modified, thus eliminating the need for a user to include many pages of interest separately.

Collections of related pages Many times, a "home page" refers to a number of other pages, both within the same namespace and external. By following the internal pages automatically, a single entry in one's hotlist could result in notification whenever any of those pages is modified. *HtmlDiff* could in turn be invoked recursively, as described above in Section 5.3.

On the other hand, centralized tracking of modifications would have the disadvantage of being decoupled from a given user's W^3 browser history; i.e., if a user views a page directly, the *snapshot* facility would have no indication of this and might present the page as having been modified. Java might be suitable for conveying that information to the server.

Modifications to support server-side tracking of modifications, including hierarchical tracking, are nearly complete.

8.4 Interactions with CGI scripts

Because AIDE can handle arbitrary URLs, it can interact with CGI scripts that use the GET protocol by passing arguments to the script as part of the URL. However, services that use POST cannot be accessed, because the input to the services is not stored.

Both *w3newer* and *snapshot* would have to be modified to support the POST protocol, in order to invoke a service and see if the result has changed, and then to store away the result and display the changes if it has. The interface to AIDE to support POST is unclear, however. A user could manually save the source to an HTML form and change the URL the form invokes to be something provided by AIDE. It, in turn, would have to make a copy of its input to pass along to the actual service.

Instead, the browser could be modified to have better support for forms:

- It should store the filled-out version of a form in its bookmark file, so the user could jump directly to the output of a CGI script.
- It should be able to pass a form directly to AIDE, along with the URL specified in the FORM tag, so that the output could be stored under RCS.

9 Conclusions

AIDE combines notification, archiving, and differencing of W^3 pages into a single cohesive tool. It achieves economies of scale by avoiding unnecessary HTTP accesses, saving pages at most once each time they are modified (regardless of the number of users who track it), and using RCS as the underlying versioning system. Automatic generation of differences within the HTML framework provides users with the ability to see both insertions and deletions in a convenient fashion.

In the general setting of the W^3 and document retrieval, AIDE benefits two communities: users of the W^3 no longer have to browse to find pages of interest that have changed; HTML providers no longer have to create suitably marked-up pages to show "what's new".

While such automation is clearly helpful in this general context, we expect that AIDE will be a critical part of more focused uses of the W^3, especially in areas involving collaborative and distributed work.

Several issues still need to be addressed. In particular, many of the complications of AIDE could be avoided by better integration with W^3 browsers and servers. The increasing availability of distributed, hierarchical HTTP repositories such as Harvest [1] will also be both an opportunity and a challenge for scalable notification mechanisms and version archives.

For more information on AIDE, see URL http://www.research.att.com/orgs/ssr/people/douglis/aide.

Acknowledgments

Robin Chen, Steve Crandall, John Ellson, P. Krishnan, Mark Rajcok, Herman Rao, and the USENIX referees provided comments on earlier drafts of this paper. Brooks Cutter wrote the version of *w3new* from which the *w3newer* script is derived. Thanks also to David Ladd for numerous discussions about *HtmlDiff*; to Rich Brandwein, Robin Chen, John Ellson, and Herman Rao for discussions about HTTP and HTML; and to the many colleagues who tried out AIDE and provided feedback. Finally, Charles Babbage invented the first computer and called it the "Difference Engine," a term we appropriated for a new context.

References

[1] C. Mic Bowman et al. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Dept. of Computer Science, University of Colorado–Boulder, March 1995.

[2] K. Church. Char_align: A program for aligning parallel texts at the character level. In *Association for Computational Linguistics*, pages 1–8, 1993.

[3] Ward Cunningham. WikiWikiWeb. http://c2.com/cgi-bin/wiki.

[4] B. B. Cutter III. w3new. http://www.stuff.com/~bcutter/programs/w3new/w3new.html.

[5] Roome W. D. 3DFS: A time-oriented file server. In *Proceedings of the USENIX 1992 Winter Conference*, pages 405–418, January 1992.

[6] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, December 1992.

[7] James S. Gwertzman and Margo Seltzer. The case for geographical push-caching. In *Proceedings of the Fifth Workshop in Hot Topics in Operating Systems (HOTOS-V)*, pages 51–55, Orcas Island, WA, May 1995. IEEE.

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

[9] J. Howard et al. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[10] 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.

[11] Java. http://www.javasoft.com/.

[12] M. Nelson, B. Welch, and J. Ousterhout. Caching in the Sprite network file system. *ACM Transactions on Computer Systems*, 6(1):134–154, February 1988.

[13] M. Newbery. Katipo. http://www.vuw.ac.nz/~newbery/Katipo.html.

[14] A standard for robot exclusion. http://web.nexor.co.uk/mak/doc/robots/norobots.html.

[15] David S. Rosenblum and Balachander Krishnamurthy. Generalized event-action handling. In Balachander Krishnamurthy, editor, *Practical Reusable UNIX Software*, chapter 9. John Wiley & Sons, New York, 1995.

[16] Specter, Inc. Webwatch. http://www.specter.com/users/janos/webwatch/index.html.

[17] Sun Microsystems. *The HotJava Browser: A White Paper*. Available as http://java.sun.com/1.0alpha3/doc/overview/hotjava/browser-whitepaper.ps.

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

[19] Url-minder. http://www.netmind.com/URL-minder/URL-minder.html.

[20] Warmlist. http://glimpse.cs.arizona.edu:1994/~paul/warmlist/.

Author Information

Fred Douglis is a member of technical staff at AT&T Bell Laboratories. His research interests include the W^3, mobile and distributed computing, and file systems. He received a B.S. from Yale University (1984) and the M.S. (1987) and Ph.D. (1990) degrees from the University of California, Berkeley, all in Computer Science. Email: *douglis@research.att.com*.

Thomas Ball is a member of technical staff at AT&T Bell Laboratories. His research interests include programming languages, software tools, techniques for efficiently monitoring system and program behavior, and software and system performance visualization. He received a B.A. from Cornell University (1987) and the M.S. (1989) and Ph.D. (1993) degrees from the University of Wisconsin-Madison, all in Computer Science. Email: *tball@research.att.com*.