Scalable Web Page Entanglement

Jason Rohrer
University of California, Santa Cruz
Department of Computer Science
Santa Cruz, CA 95064
+1-831-429-4294
rohrer@cse.ucsc.edu

December 3, 2002

Scalable Web Page Entanglement

[The human mind] operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain. It has other characteristics, of course; trails that are not frequently followed are prone to fade, items are not fully permanent, memory is transitory.

–Vannevar Bush, "As We May Think," 1945 [11]

ABSTRACT

We present a proxy-based system for augmenting the capabilities of the World Wide Web. Our system adds two-way association links and automatically removes these links when they break, while the existing web features only one-way links and lingering broken links. Several web augmentation systems have been developed in the past that add two-way links. Our key contribution is in terms of link management, which in our system is dynamic and completely automatic. Links between web pages are added and removed according to popular web traversal paths, freeing both page owners and readers from the burden of link creation and maintenance. Links can form between pages that do not link to each other at all, reflecting the fact that readers have associated these pages with each other—we described such pages as *entangled*.

We use variations on common peer-to-peer techniques to build a scalable system out of a dynamic set of proxy peers. Proxy-to-proxy communication takes place entirely over HTTP, ensuring compatibility with existing infrastructures.

A working implementation of our system is available at http://tangle.sourceforge.net/

1 INTRODUCTION

The World Wide Web, despite its popularity, is fraught with problems that limit its utility and burden the people that manage it. This paper focuses on two of these problems.

First, the web supports only one-way links. In general, if page A links to page B, we can assume that the contents of page A are somehow relevant to the contents of page B. Furthermore, if page A's link to page B is a popular means of entering B, we can feel quite certain that A's contents are relevant to the contents of B. However, page B may not necessarily link back to page A—in fact, the owner of B may not even realize that page A exists.

Next, we have the problem of static links on the web. For page A to link to page B in the first place, the owner of page A must know about page B, judge page B to be relevant, and manually add a link to page B. A similar series of steps must be carried out by the owner of page A to remove a link to page B, even in the case where B is removed from the web entirely. The owner of page A must also repeat this process if page B moves.

These problems are magnified with the growth of the web, since page owners cannot be expected to keep abreast of all relevant content, especially in the face of content that changes and moves. However, the web contains a massive untapped information resource: the navigation patterns of its users. Proper use of this information may help us deal with the problems cited above.

As users move from page to page in the web, we can assume that they are moving between pages that are somehow relevant to each other. We have developed a system that builds two-way links out of these transient page associations. We describe pages that have been linked together in this way as *entangled*. Along with overcoming the limitations of one-way links, these entanglements can track web traversal patterns as they change dynamically over time. Thus, our system can address both of the problems cited earlier.

The contributions made by this work are twofold. First, we present the novel idea of tracking user navigation patterns to add two-way association links between web pages—we refer to this general idea as *web page entanglement*. Second, we describe a proxy-based implementation of this idea that uses peer-to-peer techniques to achieve scalability.

2 WEB PAGE ENTANGLEMENT

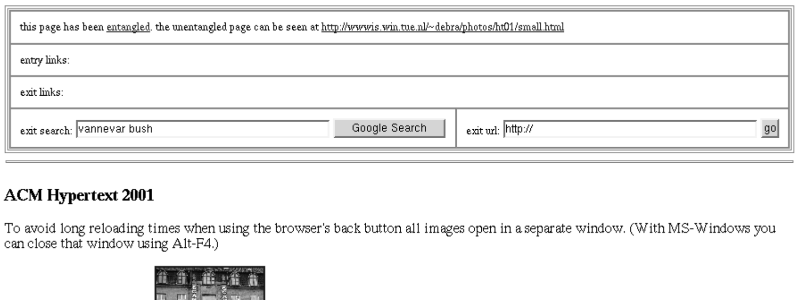
We can describe web page entanglement in general terms without yet delving into implementation details. Web page entanglement refers to automatic link addition and removal based on dynamic web usage patterns. For clarity, we first discuss the behavior of our entanglement system in response to the actions of a single user. A multi-user scenario is described toward the end of this section.

2.1 Single user scenario

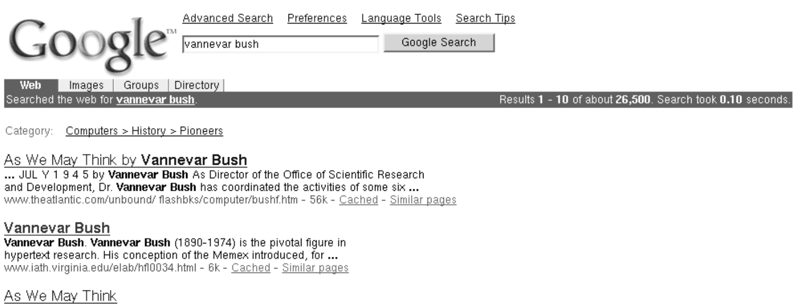
Whenever a web user exits one page by entering another, we can speculate that there is some kind of association between those two pages. For example, page A may link directly to page B, and the user may have followed such a link. The user may also have simply typed page B's URL into his or her browser after reading page A. Finally, the user may have performed a web search after reading page A and discovered page B as a result.

In the first case (a direct link), we have a definite association. In the latter two cases, we may or may not have an association. The user may have simply become bored of page A and moved on to page B. However, we can reasonably suspect that something in the contents of page A caused the user to think of page B or at least think of performing a search that resulted in page B. Thus, an association between A and B may exist that is not captured explicitly by links in the web.

With web page entanglement, we represent such potential associations by adding two-way links between the associated pages. In our example, when a user exits page A by entering page B, a link is added from page A to page B and from page B back to page A. Future visitors to either entangled page are thus made aware of this association. In the case of the latter two traversal scenarios (URL to exit and search to exit), future visitors will be seeing an association that was completely invisible in the unentangled web. In the case



(a) User searches to exit page A.

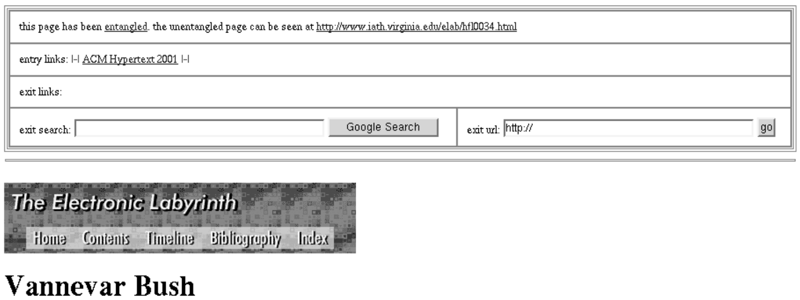


(b) User selects second search result (page B).

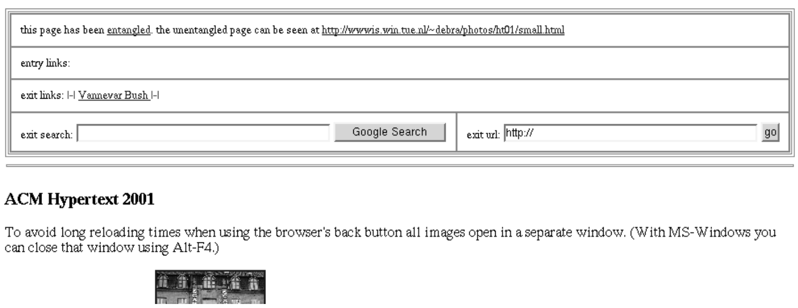
Figure 1: Example usage scenario (search to exit).

of a link traversal, future visitors to page A will simply be seeing an accentuated version of an existing web association—they will be informed that a previous visitor found the link to page B useful. On the other hand, future visitors to page B will be made aware of an inbound link from page A—such a link would have been invisible in the unentangled version of page B. Thus, for all three traversal scenarios, we capture some kind of association information that was invisible in the unentangled web.

For an example usage scenario in which a user's search entangles two previously unassociated pages, see Figures 1 and 2.



(a) Entry link from page A added to page B.



(b) Exit link to page B visible when page A revisited.

Figure 2: Result of the user actions seen in Figure 1.

2.2 Multi-user scenario

Building on the single user scenario presented so far, we now discuss the behavior of our entanglement system in response to the actions of many users.

As users move from page to page in the entangled web, the entanglement system builds two-way links from each traversal. Over time, if all of these links were preserved by the system, each page would amass an unwieldy collection of entry and exit links, many of which might be irrelevant. To deal with this issue, our system maintains only a small collection of links for each page and attempts to keep only the most relevant links, using traversal popularity to approximate link relevance.

Consider pages A and B that do not link to each other in the unentangled web. If many users independently perform a search after reading page A and move on to page B, we can surmise that these pages are strongly relevant to each other. Furthermore, if users see an exit link at the top of page A to page B and use it to move to B, we can surmise that they found the link to B useful when reading page A. Both user actions increase the traversal popularity of the two-way entanglement link between A and B, effectively increasing the system's estimation of how relevant these pages are to each other.

Link lists at the top of each page are sorted according to a popularity metric, with the most popular links appearing first in the list. As a link list fills up, the least popular links are removed from the end of the list. As many users pass through a particular page, the page gathers a list of the most popular ways to get to the page and the most popular places to go next.

In addition, an association made by a single user can effectively be judged over time by other users, vying in terms of popularity among the other association links on a page. Thus, even if a traversal would not have been popular in the unentangled web (for instance, if few people would tend to make a particular association), that traversal can become popular in the entangled web after a single user makes the association and others judge it to be a valuable association.

Results from a real-world multi-user scenario are described in Section 5.

3 IMPLEMENTATION DETAILS

3.1 Page filtering via proxy

We have developed a proxy-based web page entanglement system. Our proxy is web-based, meaning that it can be fully accessed and manipulated using the web, and it should not be confused with "web proxies" that are commonly used for caching and filtering purposes

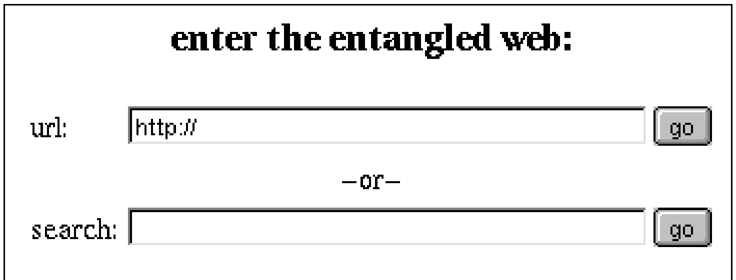


Figure 3: The entry user interface.

(these other proxies are often configured in a browser's preferences panel).

In terms of existing systems, our web-based proxy is similar to Google Language Tools' translation service [3]. After the user enters a URL, the specified page is fetched by the proxy and filtered—all links are wrapped so that they point back at the proxy. For example, if the proxy running at `tangle.org:8080` is filtering the ACM homepage, and that page contains a link to the IEEE homepage, the URL from that link will be wrapped in the following manner (split into multiple lines and unencoded for clarity):

```
http://tangle.org:8080/request?
link_to_page_url=http://ieee.org
&link_from_page_title=ACM
&link_from_page_url=http://acm.org
```

When the user clicks on this wrapped link, the user's web browser makes a request to the proxy instead of to the IEEE server. The proxy extracts the destination from the wrapped `url`, fetches the page from IEEE, filters the page (wrapping its links), and then returns the page to the user's web browser.

Thus, after entering a single URL, the remainder of the user's browsing session can be mediated by the proxy. The entry interface for our web-based proxy can be seen in Figure 3. Both URLs and searches are supported as means of entry (the proxy wraps all links in the search results page).

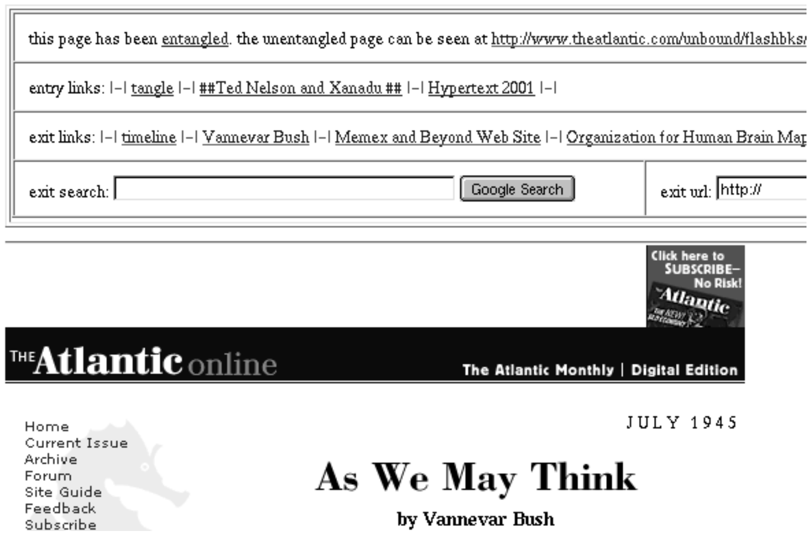


Figure 4: The user interface for an entangled page in detail.

Since all links are wrapped to point back at the proxy, the proxy can track popular entry and exit links for each page. Along with wrapping links, the proxy inserts a header at the top of each filtered page. This header features sorted lists of popular entry and exit links, as well as additional tools for exiting the current page: a search field and a URL field are provided. This header interface can be seen in the usage scenario presented earlier in Figure 1. A more detailed view of the header can be seen in Figure 4.

3.2 Link popularity

We would like to achieve a certain balance with our link popularity metric, and we describe this balance with the following four criteria. First, we want to maintain only a small collection of links for each page. Second, we want our popularity metric to be temporally sensitive: a link that underwent thousands of traversals during a flurry of interest long ago should not remain at the the top of the list indefinitely. Third, we would like to make it somewhat difficult for an attacker to flood a proxy with link traversals in order to push all existing links off of the list for a given page. Finally, we want our metric to be robust against transient fluctuations in the popularity of the most popular links (in other words, the most popular links should not be pushed completely off the list in response to a brief dip in their popularity).

3.2.1 Method

We have developed a simple link sorting mechanism that deals with all of these issues in an intuitive way. For each web page, we maintain an ordered list of N entry and M exit links. These lists are fully displayed in order as part of the user interface header. Whenever a link is traversed, the exit list on the source page and the entry list on the destination page are each changed in one of three ways. First, if the link is already on the list, the link moves up one position, exchanging places with the link above it. Next, if the link is not on the list, and the list is not yet full, the link is appended to the end of the list. Finally, if the link is not on the list, and the list is full, the link replaces the last link on the list.

N and M, the number of stored entry and exit links, are parameters in our system. Reasonable values for real-world usage scenarios and considerations for selecting good values are discussed in Section 5.

We should note the benefits of keeping separate lists of popular entry and exit links. If page A is very popular, and page B is less popular, an exit link on page A to page B may be at the bottom of page A's exit list or not on the list at all (since M other exit links on page A may be far more popular than the link to page B). However, the link from page A may be one of the most popular entry links on page B and may be at the top of page B's entry list. Thus, even if two pages are entangled, they stand alone in terms of the popularity of their entry and exit links—the ordering of links on page A does not directly affect the ordering on page B. This independence also plays an important role in system scalability, which we discuss in Section 4.

3.2.2 Analysis

The link sorting strategy presented above obviously meets our first criterion, as we are maintaining the absolute minimum number of links possible per page (those $N + M$ links that are actually being displayed). We have also met the criterion of temporal sensitivity, since a newly-popular link that receives N traversals close in time today can unseat a link from the top of the list that received thousands of traversals yesterday but very few traversals today. In order for an attacker to push all $N + M$ existing links off of the lists for a given page, he or she must execute $O(N^2 + M^2)$ false link traversals, so we have met the third criterion in a rather weak sense (such an attack would be easy to stage with a machine, but tedious to stage by hand). However, an attacker can push the most popular link off of either list with N (or M) carefully chosen false link traversals.

Analysis of the final criterion, that of robustness, is more complicated. To simplify the analysis, we will consider only robustness for the most popular link: we would like to know how easily it can be pushed completely off the list by random fluctuations in its popularity. Let us call the most popular link in the list $\hat{l}$. We assume that link traversals during such a fluctuation are uniformly distributed at random across all links in the list. Thus, for N such links, each link (including $\hat{l}$) will be hit by a particular traversal with probability $1/N$. For $\hat{l}$ to move down from the top of the list in response to a random traversal, the link immediately below it needs to be traversed, so the probability of $\hat{l}$ moving down is $1/N$; it will stay at the top of the list with probability $(N-1)/N$. Once $\hat{l}$ is in the middle of the list, it can move up with probability $1/N$ (if it is traversed), move down with probability $1/N$ (if the link below it is traversed), or stay where it is with probability $(N-2)/N$. If $\hat{l}$ reaches the end of the list, it can move up with probability $1/N$ (if it is traversed), fall off of the list with probability $1/N$ (if a new link is traversed to replace it), or stay where it is with probability $(N-2)/N$.

This probability system forms a Markov chain. We would like to find the expected number of random traversals needed to push $\hat{l}$ off of the list, which is the same as the expected number of time steps needed to reach the end of the Markov chain. Simulating this chain produced the following approximate form for the expected value:

$$E[\text{number of traversals before } \hat{l} \text{ falls off of the list}] = \frac{N^3}{2}.$$

Thus, during a transient fluctuation in popularity, $O(N^3)$ traversals are needed to push the most popular link completely off of the list.

Temporal sensitivity and fluctuation robustness are competing goals, and our system design favors temporal sensitivity somewhat. Though many random traversals are needed to push the most popular link off of the list, only one traversal is needed to unseat it from the top position on the list (though if traversals are uniformly distributed at random, we would expect N of them to occur before this happens).

For example, if the most popular link has been traversed 1000 times, and the next most popular link has been traversed only 10 times, a single traversal across the next most popular link will move it to the top of the list. In this case, a link that has received only 11 traversals would be positioned in the list above a link that has received 1000 traversals. This behavior does not seem desirable, though we point out that if the second link in the list (that with 1000 traversals) really is more popular, this unbalanced state will be short-lived, and the more popular link will soon receive a traversal that moves it back to the top of the list.

3.3 Broken links

Dealing with broken links in our system is trivial. When a user clicks on a header link referring to a page that no longer exists, the proxy detects that the link is broken (by detecting a "not found" error or an unresponsive server) and removes the link from its database. Thus, at most one user will experience each broken header link.

4 SCALABILITY

For the sake of clarity, we have thus far been discussing our entanglement system as if a single proxy filters every page on the web. Two scalability issues would plague such a single-proxy system. First, should our system become popular, a single proxy would not be able to handle the request load generated by all of the clients using it. Second, even if we had a manageable request load, a single proxy would have difficulty storing popular link information for every page on the web.

We deal with these scalability issues by building a distributed system of proxy peers. Each proxy in the system serves as an expert for a particular subset of pages on the web, tracking popular entry and exit links only for these pages. When a user exits one page and enters another, the proxy for the source page hands the user off to the best known expert proxy for the destination page. Thus, as a user moves from page to page, he or she gets passed from proxy to proxy.

Responsibility for pages is distributed among the proxies using a cryptographic hash algorithm. The URL of a page is hashed, as are the addresses of each proxy, and the proxy with the closest hash is designated as the expert for that page. The range of the hash function can be thought of as a ring, with each proxy in the system serving as the expert for a particular contiguous section of this ring space: all hashed URLs that land in a proxy's section should be filtered by that proxy. As long as the hash function uniformly distributes both URLs and proxy addresses throughout the hash ring space (which is a guarantee made by many cryptographic hash functions [18]), we get a probabilistic guarantee of URL distribution—we can expect each proxy to be responsible for an equal portion of the URLs. Furthermore, we can expect that the URLs with the heaviest load (in other words, the most popular web pages) will be uniformly distributed among the proxies, giving us a probabilistic load balancing guarantee. See Figure 5 for a graphical representation of this hash space.

4.1 User hand-off issues

The naive strategy for passing a user to the best expert proxy in response to a link traversal would be to wrap each link so that it points at the best proxy for that link. This strategy is certainly efficient: after filtering the current page for the user, the current

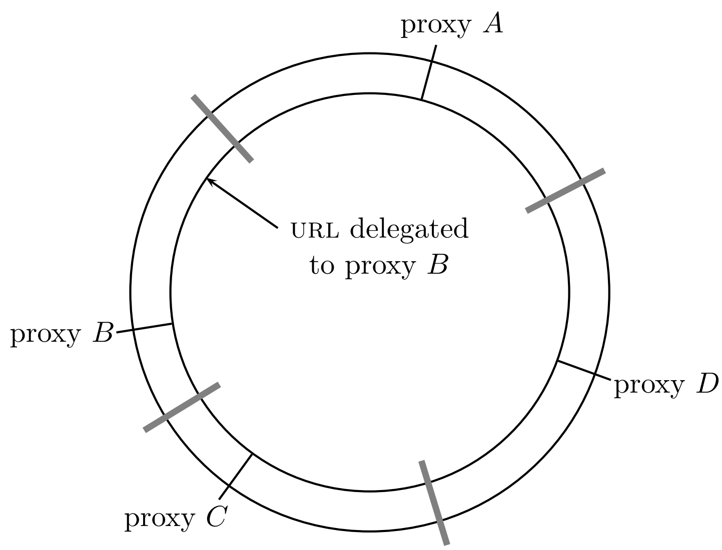


Figure 5: The ring hash space. Gray lines separate the proxies' regions of expertise.

proxy, say A, is unlikely to experience load from the user again, as the browser is responsible for connecting to the next proxy when fielding a link click. The next proxy will correctly receive entry link information from the contents of the wrapped link. However, with this scheme, proxy A will miss the exit link information present in this link traversal. Therefore, some kind of proxy-to-proxy communication is necessary during a link traversal so that both proxies are informed about the traversal.

Our solution is simple and relies on the existing infrastructure. A proxy wraps all links in a page as specially designated *exit links* that point back to itself. When proxy A receives a request for one of these exit links, it first notes the exit traversal in its popular link database. Then proxy A looks up the best proxy, say B, for this link (using the hash mechanism described earlier), generates

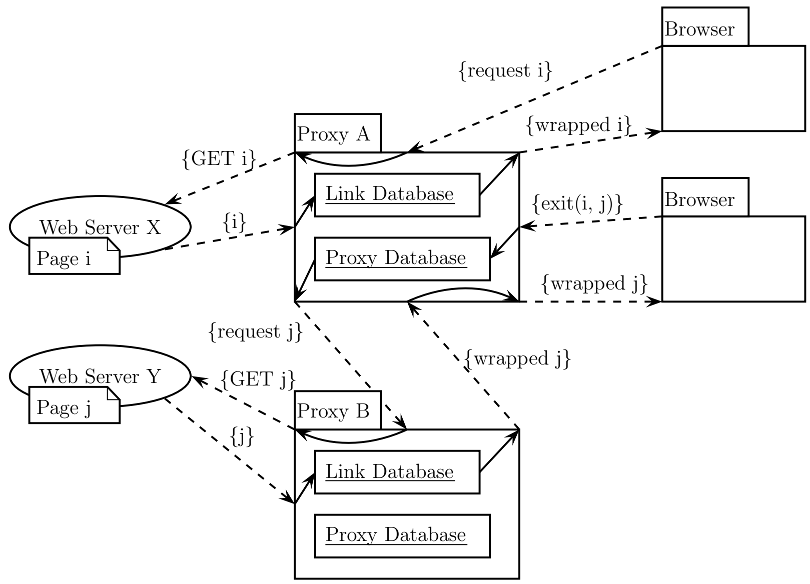


Figure 6: The implementation architecture with information flow. Dotted lines represent messages, and solid lines represent internal action paths.

an HTTP request to proxy B for the page, and passes the fetched page contents to the user's browser unmodified (proxy B will add an appropriate header interface to the page contents and filter all links in the page).

Thus, proxy-to-proxy communication is rather transparent. In fact, the next proxy cannot tell the difference between a pure user request and a forwarded exit link traversal. This pure HTTP approach is also compatible with the existing infrastructure. Other more efficient approaches might be proposed using redirects, but such approaches make it difficult to detect proxy failure, which we discuss in the next section. Figure 6 shows the implementation architecture and the information flow during a user hand-off.

4.2 Adding and removing proxies

We should note that, in terms of scalability, our goal is not to support a world-wide system of proxies managed by separate individuals and organizations. Such a system would face too many security issues to be practical: we would need to trust each proxy to operate correctly for its randomly-assigned subset of web pages. Thus, we are aiming to support a closed network of proxies that are all controlled by a single organization. Search engines and web history caches have traditionally been controlled by single organizations, yet they have been able to scale to meet the world-wide demands of the entire web.

Even in such a closed network, proxies may be frequently added and removed in the case of failures or when scaling the system to match changes in demand. Learning about failed proxies is trivial. When proxy A requests a page from proxy B in response to a link traversal, A can easily detect that B has failed and remove B from its hash database. Proxy A would then issue a request to the next-best proxy in its hash database for the page.

Mechanisms for learning about newly added proxies are not as obvious. Our approach allows existing proxies to learn about a new proxy slowly over time. Each time proxy A requests a page from proxy B in response to a link traversal, A selects a proxy at random from its hash database, say proxy C, and includes the address of C as an argument in the request URL. Thus, during each link traversal, the next proxy is informed about one proxy that is known to the current proxy. As a pair of proxies pass link traversals between each other over time, we can expect the contents of their proxy hash databases to converge.

To add a new proxy to the system, we simply need to inform one existing proxy about the new proxy (we can achieve this with a properly formatted HTTP request). Over time, as the system is used, information about this new proxy will spread, and proxies throughout the system will start delegating pages to the new proxy. Upon insertion, the new proxy "takes over" a section of the hash space, though existing proxies will temporarily delegate pages in this section incorrectly until they learn about the new proxy.

Our focus is on simplicity, not perfect consistency. Whenever a new proxy is added or an existing proxy is removed, some link information will be lost from the system. However, the lost link information will be uniformly distributed throughout the web and will have at most a minor effect on each web site (which is likely comprised of many web pages). Furthermore, since all link information in our system has been gathered automatically, small portions of this information are themselves neither all that valuable nor difficult to replace—collecting them required no human effort.

5 REAL-WORLD USAGE PATTERNS

This section describes results culled from an experiment involving a reasonably large group of users. We first describe our experimental setup.

To maintain a balance between link list accuracy and screen space usage, we set the maximum number of exit links and entry links for each page to 10 (in other words, $N = M = 10$). Maintaining too few links at the top of each page decreases the accuracy of our popularity metric, since small fluctuations in the popularity of a link can force it completely off of the list. However, each additional link in the list takes up more screen space. Users would certainly be annoyed by an entanglement header that filled an entire screen and forced them to scroll to see the top of a page's contents. With 10 entry links and 10 exit links, these link list take up about four lines total on average, which seems permissible. Referring back to the analysis in section 3.2.2, we would expect 500 random link traversals to occur before the top link is pushed completely off of the list.

Because this experiment focused on usage patterns and not on system scalability, all results were culled from a single proxy that was not connected to any other proxy peers.

In response to front-page publicity on a major news service [21], our project page received over 11,000 user visits during a 24-hour period. From traversals made by the users during this period, our system amassed entry link lists for 3069 pages and exit link lists for

1432 pages. Pages with link lists varied in terms of the number of links in their lists (between 1 and 10), though we gathered a total of 6407 links. By the end of the experiment, 18 pages had full entry link lists, and 25 pages had full exit link lists. A histogram for the number of entry and exit links associated with the pages is shown in Figure 7.

A plot of the same histogram on log-log axes can be seen in Figure 8. Note that both histogram curves are nearly linear in this log-log plot, indicating that the distribution of link list sizes among pages follows a power-law [4]. This observation might not seem surprising given the recent work that has associated power-law distributions with internet connections [15] and web link topologies [5].

However, this power-law results does tell us something new, since our system is tracking user associations, not just web links. We can view links in the unentangled web as explicit associations made by page owners, and the distribution of these links has been shown to follow a power-law [5]. We might assume that the power-law describes some aspect of the act of linking, but our result shows that the power-law describes the underlying associations, which are merely *expressed* by owner-created links in the unentangled web.

Note that our frequency curves have particularly heavy tails that diverge from a power-law distribution. This is primarily due to the way that our data have been binned: any page that had more than 10 entry or exit links associated with it had its link list capped to the 10 most popular links and thus contributes to the frequency of the last bin. We conjecture that the underlying association distribution has tails that are indeed power-law.

6 RELATED WORK

The World Wide Web [7] is obviously the most widely adopted hypertext system ever, though it was certainly neither the first nor the most feature-rich. The origins of hypertext have been attributed to the ideas of Vannevar Bush in the 1940s [11] and the work of Nelson in the 1960s [19]. In its original conception, hypertext featured two-

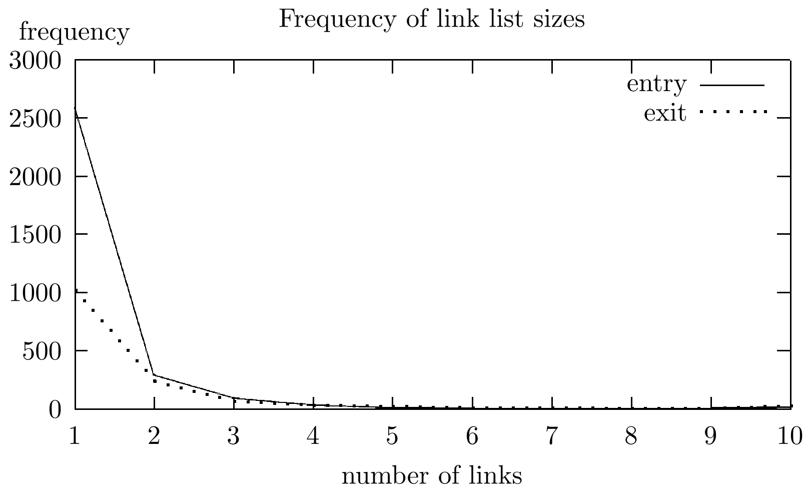


Figure 7: A histogram of the link list sizes.

way links that could be modified by both readers and writers. With its one-way, owner-created, static links, the current web certainly pales in comparison to the original hypertext vision. Recent work by Nelson describes this discrepancy in detail [20]. These weaknesses have motivated a great deal of research on improving the behavior of web systems.

The web-based hypertext encyclopedia Everything2 [1] is similar in several ways to the entanglement system presented here. Each node in the Everything2 system contains a list of popular two-way links that are culled from both link traversals and searches. Everything2 seems to come much closer to fulfilling the hypertext vision than the web does. However, though Everything2 is web-based, it is a closed hypertext system, and its link tracking mechanism does not work for the web at large. In fact, the Everything2 system forbids links to the outside web and provides no mechanism for them.

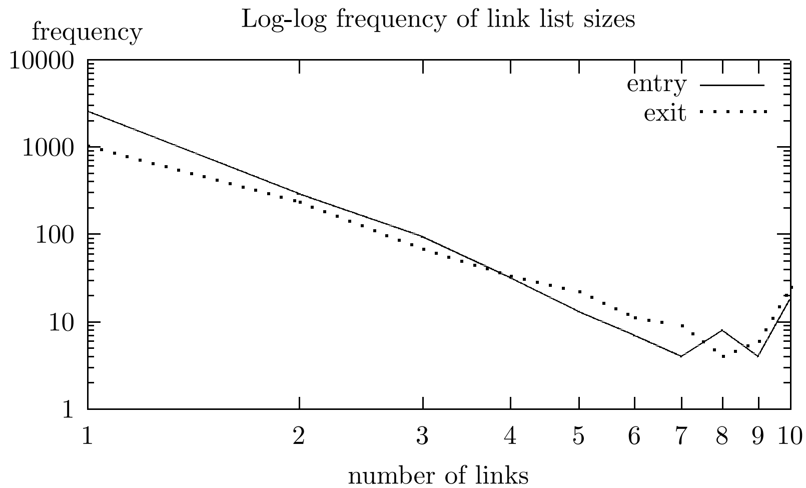


Figure 8: A histogram of the link list sizes plotted on log-log axes. A power-law distribution is evident.

6.1 Adding hypertext features to the web

With the advent and popularity of the web, a great deal of work has been done to retrofit it with some of the functionality present in older hypertext systems.

Anderson [6] presents an applet-based approach to augmenting the web with hypertext features. His approach is similar to ours in that all links are wrapped to point back to the hypertext service, forcing all pages to pass through the service before being displayed to the user. Links can be maintained manually using the applet.

Carr *et al.* [12] present a proxy-based system for web augmentation based on Microcosm. Their system supports unmodified web browsers and servers, and unmodified browsers can be used to create links. However, they recommend the use of a browser plug-in, noting that using web-based forms for link maintenance is tedious. This system supports Microcosm's notion of *generic links*, or links that can be anchored to keywords. Generic links appear wherever their keywords appear, even in documents created in the future.

Grønbæk *et al.* [17] describe another proxy-based system for adding links to the web. Unmodified browsers can only view augmented links in their system, and a plug-in is required to modify links.

All three of the augmentation systems described above support only static links that must be added, removed, and edited manually—they provide no mechanism for automatic link maintenance. Microcosm's generic links might be seen as a dynamic link mechanism, since one link object will be applied automatically throughout the entire document corpus. However, the underlying link objects themselves are static: a generic link must be created, destroyed, and edited by hand.

6.2 Tours and trails

Guided tours are a popular feature in traditional hypertext systems, with one of the first implementations attributed to Trigg [23]. Using guided tour functionality, users can record their hypertext traversal paths in a form that can be replayed to themselves or others in the future. Furuta *et al.* [16] use a proxy-based system to add guided tours to the web. With their system, tours can be viewed using an unmodified web browser, though tours can only be created using a separate authoring tool. Also, the authoring tool does not record paths automatically: the user must create each step in the path by manually entering a URL. This work is similar to our web entanglement system, which can be seen as a web-wide, automatic, guided tour system that shows popular web traversal paths. Of course, as the user follows a series of popular entanglement links in our system, he or she is not following a path created by an individual user, but a well-worn path created by the population as a whole.

The MEMOIR system described by De Roure *et al.* [13] tracks users' trails as they navigate through an information space. Trails are compared, and users are informed about similar trails that have been explored by others in the past. MEMOIR treats trails as first class objects that can be saved and manipulated by users. Both MEMOIR and our entanglement system are inspired by the ideas of Bush mentioned earlier, though our system has no explicit notion of trails. Our system captures population-wide traversal patterns and does not focus on adapting to the context of the current user.

6.3 Dealing with web complexity

With the exponential growth of the web, sifting through its vast information space has become troublesome. Search engines such as Google [10] have used the information present in link structure to sort search results, providing much more usable search facilities. In addition to the problem of navigating the web, there is also the problem of maintaining its links. Other web augmentation work has focused on automatically adding links to the web.

El-Beltagy *et al.* [14] describe a system that adds links to documents based on both document and user context. Links are extracted from documents using automatic context analysis and stored in link databases. As users browse the web, the path of each user is analyzed automatically to determine that user's current interest context. Links are added to the pages viewed by each user based on that user's suspected interests. This context-based approach does not take advantage of the information present in user traversal patterns for link discovery. However, using text-based page comparison, their system can create links between pages that do not link to each other at all, so it is in some ways similar to the entanglement system presented here.

Other work on automatic link generation has also focused on context-based approaches in which similar documents are linked together. Wilkinson and Smeaton [24] provide a good overview of this research area.

De Bra and Calvi [9] present a system architecture that builds a model for each user based on past browsing habits. Using these models, their system shows, hides, or annotates links and content to adapt to the current user. However the underlying link structure is static, and their system does not have a mechanism for automatic link discovery.

6.4 Adaptive hypertext networks

Bollen and Heylighen [8] present an adaptive hypertext system that is closely related to the entanglement system described here. Their system restructures hypertext link networks based on the traversal patterns of the user population. Each link is associated with a weight, and user traversals modify link weights according to various learning rules. On each page, links are presented in sorted order based on their weights. Their experiments demonstrate that the most relevant links for a page rise to the top of the list over time.

Bollen and Heylighen criticize other adaptive hypertext systems for maintaining a static underlying link structure, claiming that their system has a dynamic link structure. However, the link structure in their system is in fact static: the changing weights affect the order of presentation for the links, but links are never added or removed.

For example, consider the 150-node experimental system that they present. Each node links to every other node in the system, though users can only view these lists of 149 links in sets of 10. Links are initialized with small random weights and allowed to adapt over time according to user navigation patterns. Of course, since all possible links are present in their system, the weights can evolve to match any possible all-to-all weighted graph structure.

Though using a static all-to-all link collection will work in a small, experimental network, it will certainly not work for a system as large as the web, which contains billions of nodes to date [2]. The authors do not discuss this issue, though their system could perhaps be adapted to larger networks by manually creating a small list of relevant links for each page and allowing the weighting system to adaptively order each link list independently.

Regardless, Bollen and Heylighen's system provides no mechanism for adding or removing links adaptively. Thus, if the population's associations do not match those of the initial link list author, their associations will never be represented by the links on that list. Our system deals with this issue by using all forms of page traversal (including search) for link discovery. Furthermore, our system's data structures are designed to work with a hypertext network as large as the web.

6.5 Peer-to-peer aspects

The use of a hash algorithm to distribute responsibility for web pages among a set of proxies is similar to the techniques used by Stoica *et al.* for Chord, their distributed data location system [22]. In Chord, both data blocks and server addresses are hashed, and each data block is stored at the server whose hashed address is the successor of the data hash in the hash space. Chord provides mechanisms for hopping through the hash space to locate a particular data item, ensuring that the data item can be found if it exists in the system, even in the presence of failures. Our page entanglement system differs in that low latency is more important than perfect consistency—we cannot afford to hop through the proxy network in search of the best proxy. Thus, in locating the best proxy for the next web page requested by the user, we make at most one hop. If the current proxy does not know about the best proxy for a page, the next-best known proxy will be used.

7 CONCLUSION

We have presented a simple, usable, scalable system for adding association links to the web based on a previously untapped information resource, user navigation patterns. Our system's simplicity is rooted in our use of an existing mechanism, the HTTP request, for transparent browser-to-proxy and proxy-to-proxy communication. Our web-based proxies and unobtrusive user interfaces make our system usable for anyone with a web browser and basic web-browsing skills—no change in browsing habits is necessary to use our system. We achieve scalability through an adaptation of peer-to-peer techniques, supporting seamless removal and addition of proxies at runtime.

With the use of our system, the web moves one step closer to fulfilling the visions of the early hypertext pioneers. Recalling the static linking mechanisms that they proposed, we might even say that our system helps the web move one step beyond some of the limitations inherent in their visions.

A working implementation of our web page entanglement system can be accessed at

`http://tangle.sourceforge.net`

8 ACKNOWLEDGEMENTS

We are grateful to Jim Whitehead for his valuable thoughts and comments. We also thank the anonymous reviewers for their critiques.

REFERENCES

[1] Everything2. http://www.everything2.com.

[2] Google. http://www.google.com.

[3] Google language tools. http://www.google.com/language_tools.

[4] ADAMIC, L. A. Zipf, power-laws, and pareto—a tutorial. Tech. rep., Xerox Palo Alto Research Center, 2000. http://www.hpl.hp.com/shl/papers/ranking/.

[5] ALBERT, R., JEOUNG, H., and BARABASI, A.-L. The diameter of the World Wide Web. *Nature*, 401, 1999, 130–131.

[6] ANDERSON, K. M. Integrating open hypermedia systems with the World Wide Web. In: *Proceedings of the 1997 ACM Conference on Hypertext*, 1997, 157–166.

[7] BERNERS-LEE, T., CAILLIAU, R., GROFF, J.-F., and POLLERMANN, B. World-Wide Web: The information universe. *Electronic Networking: Research, Applications and Policy*, 1(2), 1992, 74–82.

[8] BOLLEN, J., and HEYLIGHEN, F. A system to restructure hypertext networks into valid user models. *The New Review of Hypermedia and Multimedia*, 4, 1998, 189–213.

[9] BRA, P. D., and CALVI, L. Aha! an open Adaptive Hypermedia Architecture. *The New Review of Hypermedia and Multimedia*, 4, 1998, 115–139.

[10] BRIN, S., and PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7), 1998, 107–117.

[11] BUSH, V. As we may think. *The Atlantic Monthly*, 176(1), 1945, 101–108.

[12] CARR, L. A., DE ROURE, D., HALL, W., and HILL, G. Implementing an open link service for the World Wide Web. *World Wide Web*, 1(2), 1998, 61–71.

[13] DE ROURE, D., HALL, W., REICH, S., PIKRAKIS, A., HILL, G. J., and STAIRMAND, M. Memoir—an open framework for enhanced navigation of distributed information. *Information Processing and Management*, 37, 2001, 53–74.

[14] EL-BELTAGY, S., HALL, W., DE ROURE, D., and CARR, L. Linking in context. In: *Proceedings of the 2001 ACM Conference on Hypertext*, 2001, 151–160.

[15] FALOUTSOS, M., FALOUTSOS, P., and FALOUTSOS, C. On power-law relationships of the internet topology. In: *Proceedings of the 1999 ACM SIGCOMM Conference*, 1999, 251–262.

[16] FURUTA, R., SHIPMAN III, F. M., MARSHALL, C. C., BRENNER, D., and HSIEH, H.-W. Hypertext paths and the World-Wide Web: Experiences with Walden's Paths. In: *Proceedings of the 1997 ACM Conference on Hypertext*, 1997, 167–176.

[17] GRØNBÆK, K., SLOTH, L., and ØRBÆK, P. Webvise: Browser and proxy support for open hypermedia structuring mechanisms on the WWW. In: *Proceedings of the Eighth World Wide Web Conference*, 1999, 253–268.

[18] MENEZES, A. J., VAN OORSCHOT, P. C., and VANSTONE, S. A. *Handbook of Applied Cryptography.* CRC Press, 2002.

[19] NELSON, T. H. The hypertext. In: *Proceedings of the World Documentation Federation Conference*, 1965.

[20] NELSON, T. H. Xanalogical structure, needed now more than ever: parallel documents, deep links to content, deep versioning, and deep re-use. *ACM Computing Surveys*, 30(4es), 1999.

[21] ROHRER, J. Web page entanglement. *Slashdot*, 2002. sid=02/11/10/1858259.

[22] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., and BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In: *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001, 149–160.

[23] TRIGG, R. H. Guided tours and tabletops: Tools for communicating in a hypertext environment. *ACM Transactions on Office Information Systems*, 6(4), 1988, 398–414.

[24] WILKINSON, R., and SMEATON, A. F. Automatic link generation. *ACM Computing Surveys*, 31(4es), 1999.