# Multi-way Rendezvous in the Web*

**USC TR 98-687**

John Heidemann Wenhui Zhao
University of Southern California/Information Science Institute

December 4, 1998

## Abstract

*Web Rendezvous* is the meeting of information publisher and subscriber, allowing the subscriber to create and maintain a list of links (indices) to publishers' pages. Rendezvous is difficult because pages may move and unauthorized publishers may seek to add their pages to the index (spamming). Current rendezvous approaches are work-intensive or restricted to very limited domains. We provide a taxonomy of web rendezvous methods. We identify automated methods which allow a subscriber to maintain an index of many pages with only a fixed amount of effort. Our methods leverage existing web search engines to be robust to page movement, and use encryption to prevent index spamming. We have designed and implemented two different rendezvous methods; we summarize our experience using them, finding that we can securely track index changes in a few days with publisher support, and eventually even without publisher support.

## 1 Introduction

One of the distinguishing features the web added to prior Internet information systems was the link (taken from earlier hypertext systems). Links allow a page to point to nearly any other resource on the Internet. Web pages often organize Internet information with lists of links (or *indices*), from simple employee home pages to hierarchical indices such as Yahoo.

As these lists grow, link maintenance becomes a serious problem. If the index author also creates the pages pointed to by the list, this maintenance is merely a chore. Tracking link changes becomes more and more difficult, though, as the index and page maintainers become more distant. Consider creating and maintaining a list of papers submitted to a conference: links will point to pages maintained many different groups, and these links will change over time as authors move. Without care, the quality of the index will quickly degrade as more and more links are broken. This is an example of the notification problem identified by Fielding [6].

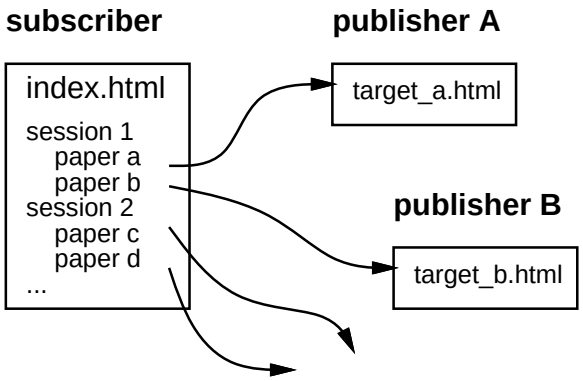


Figure 1: Defining the terms of web rendezvous.

This paper examines the problem of index creation and maintenance. We view this problem as arranging a *rendezvous* between information *publishers* (the creators of the pages pointed to by the index) and the *subscriber* (the index creator), as shown in Figure 1. Links in the current web are a very fragile way to rendezvous because they are one-way (subscriber to publisher) and manually maintained. Working within the constraints of the current web (HTML and URLs), we show how links can be made robust to movement of the subscriber and publisher, and how a subscriber can maintain an index of many pages with a constant amount of effort (adding new pages to the index does not require ad-

*The authors can be contacted at 4676 Admiralty Way, Marina del Rey, CA, 90292-6695, or by electronic mail to johnh@isi.edu and wzhao@usc.edu. John Heidemann's work was partially funded under the VINT project (DARPA grant ABT63-96-C-0054) and the LSAM project (DARPA grant J-FBI-95-185).

ditional subscriber effort). Our approach is based on including link information at both the subscriber and the publisher and automating link maintenance with existing web search engines. Search engines are frequently vulnerable to spamming, the association of irrelevant content with keywords, often for the purposes of advertising. We show how cryptographic techniques can avoid unauthorized rendezvous.

Although we describe our results in the context of the World-Wide Web, they could easily be generalized to other information systems where full-text search-engines exist. Similar approaches could also be used to manage symbolic links (also known as aliases or shortcuts) in file systems. Also, although we focus on multi-way rendezvous where a subscriber keeps links to many other pages, our approaches are also applicable to rendezvous between two people.

We begin with a taxonomy of rendezvous types and the problems of index creation and maintenance. We propose designs for several important cases of rendezvous and show how application of search engines and cryptography can address the index creation and maintenance problems. We have implemented several rendezvous mechanisms; we describe interesting and unexpected implementation details and our experiences using these services. Finally we compare our approach with others.

## 2 Rendezvous Classification

A rendezvous is the process of connecting an information *publisher* with a *subscriber* with a *link*. The link has a *source* in the subscriber's document and it points to a *target* in the publisher's document. Figure 1 shows a sample subscriber with two publishers. The subscriber maintains the source (index.html), while each publisher maintains their target (paper_a.html and paper_b.html).

In the web today, the publisher makes data available. The subscriber somehow finds out about this information, determines a URL for it (the target), then adds this URL to an existing HTML page (the source).

There are several disadvantages with this process. First, it is very labor-intensive for the subscriber who must find the target, copy the link, and add it to his source. Finding the target is often difficult, particularly when the link crosses administrative boundaries. Copying the link and updating the source can be error-prone. WYSIWYG tools can alleviate the work required somewhat, but it is still costly. Second, this process doesn't scale for indices containing links to many targets. For example, imagine constructing a list of all papers appearing in a conference or all project pages in a business. The work just described for one link must be duplicated for each other link. Finally, manually established links often break if the source or target move.

This basic rendezvous process provided by HTML and URLs is fairly limited: the link is always manually created and maintained, it links a single data publisher to a single link subscriber, and the security and correctness of this link is manually verified. We can provide a much more useful rendezvous service if we generalize each of these link characteristics:

**expected targets** Is the link intended to point to a single target or many potential targets?

**source creation** The subscriber must always initiate the link source. When creating multi-target links, is publisher or subscriber interaction required for each target?

**target creation** The publisher must always author the contents of the link target. Is publisher or subscriber interaction required to secure the target-side of the link?

**security** Is the link created in a secure manner, or is it possible for third-parties to duplicate the source end of the link or add additional link targets?

Varying these parameters provide a variety of kinds of rendezvous. Table 1 summarizes the characteristics of several cases we explore in the next section.

With this taxonomy, several problems are apparent in multi-way rendezvous.

**index degradation** Both the subscriber and the publishers can move. When links are maintained manually, they will break over time if movements are coordinated with link updates. Over time the quality of the index will degrade.

**excessive subscriber work** Manual creation and maintenance of a rendezvous to $n$ targets requires $O(n)$ work by the subscriber. As $n$ grows this cost can become very expensive. A better system would require a constant amount of work by the subscriber.

**link forgery** A private subscriber may wish to limit the set of valid publishers which can participate in the index. Ideally, this is possible without manually verifying each target. Search engine spamming is an example of link forgery.

| type | expected targets | source creation | target creation | security | section |
|---|---|---|---|---|---|
| manual | one | — | subscriber | manual | 2 |
| CGI | many | publisher | — | none | 3.1 |
| CGI+password | many | publisher | — | password | 3.1 |
| keyword search | many | subscriber | — | none | 3.2 |
| magic-word | one/many | subscriber | publisher | none | 3.3 |
| magic-word+password | one/many | subscriber | publisher | password | 3.4 |
| magic-word+pub. key | one/many | subscriber | subscriber+publisher | pub. key | 3.5 |

Table 1: Some different types of rendezvous considered in this paper.

**link modification** A publisher must prevent unauthorized link modification once links have been added.

In the next section we consider these problems across different rendezvous approaches.

# 3 Rendezvous Design

We have already described manual rendezvous creation. This section looks at several approaches to automate rendezvous creation and maintenance.

To illustrate these designs we use the example of a conference chair wishing to make a list of all papers to appear at a given conference.

## 3.1 CGI Scripts

The Common Gateway Interface (CGI) provides a standard approach to extend web server services. A CGI program could remove the subscriber from the process of index creation and maintenance. The subscriber would create a CGI program to manage the index and announce this program to potential publishers. A publisher would then use this program to update the index to refer to new content.

CGI scripts have the advantage that avoids excessive subscriber work. They also suffer from index degradation; they require manual publisher intervention to accommodate publisher movement and have no mechanism to correct subscriber movement. Subscriber movement can be addressed by adding an edit capability to the CGI script, but this raises the problem of unauthorized link modification. Finally, link forgery remains a problem.

Addition of a password to the CGI program will prevent link forgery at the cost of password distribution. In many cases, the password can be broadcast with moderate security (for example, by announcing it at a conference or on a mailing list). A single password makes revocation costly, however, individual publisher passwords add linear subscriber work. Publishers need to know the location of subscriber to run the CGI program. If there are more than one subscribers making the index, publishers have to run each subscriber's CGI program individually. When a publisher moves, he can run the subscriber's CGI program to tell the subscriber his new location. However when the subscriber moves, he has to find a way to notify all publishers. Finally, CGI programs are somewhat difficult to install and maintain in many environments.

An example of the CGI approach can be seen in the Comprehensive Perl Archive Network (CPAN [4]). It employs passwords and thus linear subscriber work, but it is otherwise quite powerful.

## 3.2 Keyword Search

*Keyword search* is the action of subscriber searching for some *keywords* to find the location of publishers. The keyword is chosen by the subscriber independently. Publishers are not involved in keyword search. Keyword search is used widely by most index creators. When a subscriber wants to index of a group of publishers, he will connect to some search engines and search for some keywords which he think appear in the publisher's page. Then the subscriber makes the index based on the search result. The connection between publisher and subscriber is minimal since there is no interaction between the publisher and subscriber.

Keyword search accomplishes our main goal: it is robust against index degradation. Subscriber can discover publisher movement by periodically re-searching for the keywords. Publishers do not care about subscriber's migration since publishers are not involved in the index making process. Keyword search is also robust to link modification.

Three problems preclude use of keyword search for general rendezvous. First, it can be difficult for subscriber to find a proper set of keywords which uniquely

locate all desired publishers. It is possible that there is no common keyword appear to all publishers' pages. Even if such keyword exists, subscriber has no way to know it.

Second, keyword search is vulnerable to link forgery. Although techniques to minimize spamming exist, they are at best imperfect as shown on many current search engines. In the example scenario, authors of rejected papers could covertly add them to the index of accepted papers, and completely unrelated pages (perhaps selling things) could be added as well, thus destroying the value of the index.

An additional problem is that search engines often return stale links, links to pages that no longer exist. We would like to filter these out.

We have implemented an automatic keyword search service and describe our experiences in Section 5. Although it is very valuable when publishers are not involved, it is clearly not suitable for a general purpose rendezvous mechanism.

### 3.3 Magic Word Search

*Magic Word Search* modifies keyword search with publisher participation. Publishers puts a magic word, a sequence of characters unique on the web, in each of their targets, allowing the subscriber to locate all publishers by searching for this magic word. The magic word is thus the connection between the publisher and subscriber. It may be distributed by the subscriber to all publishers or it may be an agreement among all publishers and distributed to subscribers. In Section 4 we describe how we automatically chose these words; in practice a long enough string of random characters is sufficient.

Like keyword search, magic word search is resistant to index degradation and link modification. Periodic automatic searches update links broken due to subscriber of publisher movement. Since the rendezvous between subscriber and publisher is based on a search for the magic word, the success of rendezvous depends on the frequency and completeness of search engines. An ideal search engine which will locate all pages containing the magic word does not exist, but in Section 5 we evaluate the effectiveness and performance of current search engines at this task and suggest how they could be improved for our needs.

The major problem with magic word search is link forgery. Just as with simple keyword search there is nothing to prevent unauthorized publishers from stealing magic words and adding forged links to the index. Thus magic word search is a general solution only if the user population is trusted as perhaps on a private web or intranet.

In Section 4 we describe the program we implemented to generate magic words. With our implementation of keyword search this provides an implementation of rendezvous with magic word search.

### 3.4 Magic Word Search with Secret Key Encryption

In order to solve the authentication problem in magic word search, we can add *secret key encryption*. A secret key (implemented as a password) is shared by publishers and subscribers. Publishers place, in addition to the (clear-text) magic word, an encrypt version of some known constant and the URL of their page.

To find valid publishers, the subscriber first searches for all pages with the magic word. To prevent link forgery we then validate the page by decrypting the encrypted block with the secret key. Failure to decrypt indicates a forgery. Successfully decrypting the known constant indicates possession of the secret key, but does not indicate *who* had the key (after all, a clever forger could have copied the encrypted text from a valid publisher). We therefore fetch the page pointed to by the encrypted URL and consider that a candidate for adding to the subscriber's index. Thus copying crypt text from valid target A into a forged target F results in only additional links to A, not to F. We then suppress duplicate links. Security for this process assumes that DNS and the web server, the mechanisms for URL resolution, are secure.

As described, this algorithm will correctly identify pages and reject forgers who copy the encrypted block from a good page. However, this does not provide protection against index degradation: is a publisher moves a page without adjusting the encrypted block the valid but moved page will be rejected as a forgery. We expect users to recompute this block when moving the page as a matter of course, but we would prefer to not require that. We can solve this in two ways: first, if we can relax the rule about URL comparison to allow some prefix matches to be considered correct. If the required prefix matches only URLs controlled by the user or trusted friends then it protects against spamming while allowing some page movement without recomputation. Second, if we include a checksum in of the page contents (without the encryption block) in the block than we can recognize and accept unmodified pages that occur anywhere in the net.

Some of the contents encrypted are publicly known. To protect against attacks using this known text we

add a random amount of padding before the URL. Thus the final algorithm is as follows:

1. Publishers and subscribers choose magic word: $M_1$, padding $P$ and secret password: $K_1$. The publisher chooses page contents *text*.
2. Publisher computes *cksum* over *text* and puts

$$text, M_1, \{P, URL, cksum\}_{K_1}$$

into the target page.

3. Subscriber connects to search engines and searches for the magic word: $M_1$. For each page he extracts the URL For each $URL_{find}$ returned from the search engines, do the following:

Subscriber downloads the page content: *text*, *eblock* for the $URL_{find}$.
If *text* does not includes $M_1$ then
discard the $URL_{find}$.
Else
Decrypt the encrypted block *eblock* with $K_1$ to get $URL_{clear}$ and $cksum_{clear}$.
If $URL_{clear}$ prefix matches $URL_{find}$ then
add $URL_{find}$ to index
Else if $cksum_{clear} = cksum over text$ then
add $URL_{find}$ to index
Else discard $URL_{find}$
End If
End If

Magic word search (Section 3.3) provided links which are robust to index degradation and link modification. The addition of secret key encryption meets the final goal of preventing link forgery. Excessive subscriber work is avoided if the secret key can be distributed efficiently. Key distribution is an area of active research and absolute security may not be possible without requiring the subscriber do specific work for each client. In practice, however, there are many cases where sufficiently private (but not absolutely secure), or relatively efficient solutions are possible. For example, efficient relatively secure opportunities for key broadcast exist if the subscriber and publishers are at a common meeting, or if a private e-mail list exists. Alternatively, secure but only relatively efficient approaches to distribution might include piggybacking the secret key on an existing per-publisher message. In our example, this could be done by including the secret key in the paper acceptance message sent by the program chair. We explore another example with our final algorithm in the next section.

We have implemented magic word search with secret key encryption and describe our experiences in Section 4.

## 3.5 Magic Word Search and Public Key Encryption

The magic word search and secret key encryption algorithm needs a secure channel to distribute secret key. We can avoid the requirement of secure channel if the subscriber knows the *public key* of all publishers. Rather than encrypting the known content and URL with a shared secret key, the publishers can do this encryption with the private key portion of their public key pair. The algorithm is as follows:

1. Publishers and subscribers initialize magic word: $M_1$.
2. Publisher computes *cksum* over *text* and puts

$$M_1, \{URL, cksum\}_{K^{-1}}$$

into the target page ($K^{-1}$ is the private part of the publishers public key pair)

3. Subscriber connects to search engines and searches for the magic word. For each $URL_{find}$ returned from the search engines, do the following:

Subscriber downloads the page content: *text*, *eblock* for the $URL_{find}$.
If *text* does not includes $M_1$ then
discard the $URL_{find}$.
Else
Decrypt the encrypted block *eblock* with $K$ to get $URL_{clear}$ and $cksum_{clear}$.
If $URL_{clear}$ prefix matches $URL_{find}$ then
add $URL_{find}$ to index
Else if $cksum_{clear} = cksum over text$ then
add $URL_{find}$ to index
Else discard $URL_{find}$
End If
End If

The algorithm requires that the subscriber know each publisher's public key in advance. This requires the subscriber do an $O(n)$ amount of work, violating our goal of avoiding excessive subscriber work. The magic word searching with public key encryption therefore only meets our goals if the subscriber already knows the publishers' public keys for some other reason. In our example this would be the case if the authors already were required to submit their public keys with their papers.

We have not implemented magic word search with public keys.

# 4 Implementation

Our implementation consists of several programs for the publisher and subscriber and a library for interacting with search engines.

Our implementation is in Perl with calls to PGP for encryption. We use the `WWW::Search` library [8] to interact with web search engines. WWW::Search provides a Perl API to querying web search engines. A common front-end drives searches, while separate back-ends interact with 24 different search engines (as of November 1998).

## 4.1 Keyword Rendezvous

Our keyword-based rendezvous is provided by the program `AutoSearch`. A user runs it the first time, specifying the set of keywords to search for. It is then run periodically (for example, from a cron job under Unix or as a scheduled event under Windows). Each time it is run it updates a list of pages with the required keywords. Output is an set of HTML files, one for the master list and an addition file with the changes from each run. Output format can be specified by a template.

AutoSearch uses any search engine supported by the WWW::Search library. It can use advanced searches when supported by the search engine and WWW::Search. Therefore keywords can be not only text in web pages, but also particular parts of web pages such as URLs referring to other pages.

## 4.2 Secure Rendezvous

Secure rendezvous is provided by the magic word with secret key encryption algorithm and is implemented by three programs.

First, the subscriber runs `MakeKey` to generate the magic word and a secret key. Currently both a pseudo-random strings of eight letters, but they can be made arbitrarily long to probabilistically insure uniqueness. The subscriber distributes these to each publisher through out-of-band, secure means.

The publishers use the program `MakeHtml` to annotate a web page they wish to publish with the information provided by the subscriber. The magic word is placed in a meta tag for search engines to find it. The URL is encrypted with the secret key. Publishers have some control of the formatting and ordering of their link on the subscribers page; they optionally specify the text to be used for the link and a sorting key. Finally, to encourage rapid discovery rendezvous with the new page, MakeHtml automatically submits the target to several search engines.

Finally, the publisher constructs an empty index page and then runs the program `AutoLink` periodically. AutoLink queries one or more search engines and adds links to the index. Unlike AutoSearch, AutoLink can take the union of the results of multiple search engines.

# 5 Evaluation

We have designed and implemented two web rendezvous mechanism to prevent the problems outlined in Section 2: index degradation, excessive subscriber work, link forgery, and link modification. This section examines our keyword search mechanism and our magic word search with secret key encryption to evaluate how well they work and how fast they respond to changes. Using this experience we reflect on how we customize a search engine to better support web rendezvous.

## 5.1 Keyword-based rendezvous

We have used keyword-based rendezvous to maintain several lists of pages related to several technical subjects in web development (load balancing, persistent connections, etc.) and lists of pages that pointed to a subtree of the web. Each list was automatically maintained for about a year. Some are currently active while others are discontinued.

We examined our uses of keyword-based rendezvous to evaluate how effective it is at maintaining indices. The first problem with keyword-based searches is selecting the keywords. Our initial applications were to track research on web technologies such as caching and performance from 1996-1997. Searches on these very terms quickly became very long, exceeding both the number of results permitted in our search-engine query and our patience at examining the results. A later application was to index all pages that reference a particular web server and URL prefix (using the AltaVista url: advanced search option). This more constrained search has been much more effective.

An effective user interface for large keyword-based searches is critical to making them useful. In addition to the basic index listing all pages that match the query we provide a page summarizing changes from the most recent run. A serious problem with accurately reporting changes to an index is that not all web search engines return complete or consistent results. Frequently many targets will be missing one week to be found again the next week, resulting in redundancy

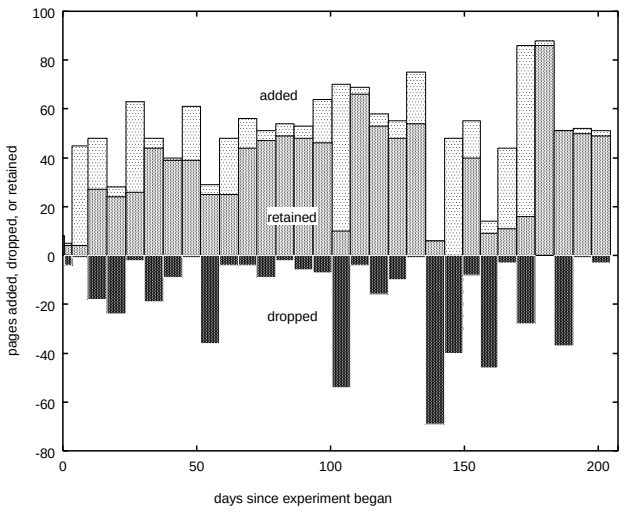


Figure 2: Keyword-based rendezvous results over time.

in the weekly change report. Figure 2 shows weekly search results illustrating this problem. For example, just before day all targets were missing, most of these reappeared over the next two weeks. We discuss reasons for this problem and possible solutions below in Section 5.3.

We did not observe problems with link forgery in our uses of keyword search. However, our uses were quite modest; we expect spamming to be in proportion to the visibility of a given query.

We did not directly measure time for a particular target to appear in our index for keyword-based rendezvous. We used keyword rendezvous for opportunistic target discovery rather than for tight collaboration between publisher and subscriber. Time for their targets to be added to an index is dependent on how quickly they are found by a search engine. Since information publishers do not know about subscribers we cannot assume they explicitly submit their pages for early searching.

## 5.2 Magic Word Search

Although keyword-based rendezvous is useful when the publishers are not involved, we believe that a secure algorithm will be preferred when both parties are aware of the rendezvous. This section evaluates how long it takes to set up a new rendezvous using magic word search and how rapidly it adapts to target movement.

In our experiments we used our implementation of secure rendezvous (Section 4.2) with queries to four search engines, AltaVista, Infoseek, HotBot and Lycos.

| | case 1 | | | | case 2 | | | |
|---|---|---|---|---|---|---|---|---|
| **day** | **AV** | **Is** | **rv.** | **act.** | **AV** | **Is** | **rv.** | **act.** |
| 1 | 0 | 0 | 0 | 13 | 0 | 0 | 0 | 10 |
| 2 | 9 | 0 | 9 | 13 | 10 | 0 | 9 | 10 |
| 3 | 13 | 0 | 9 | 10 | 11 | 0 | 9 | 9 |
| 4 | 13 | 0 | 9 | 10 | 11 | 0 | 9 | 9 |
| 5 | 13 | 0 | 9 | 10 | 11 | 0 | 9 | 9 |
| 6 | 12 | 0 | 10 | 10 | 10 | 0 | 9 | 9 |
| 7 | | | | | | | | |
| 8 | | | | | | | | |
| 9 | 12 | 0 | 10 | 10 | 7 | 1 | 7 | 9 |
| 10 | 10 | 0 | 10 | 10 | 7 | 1 | 7 | 9 |
| 11 | | | | | | | | |
| 12 | 12 | 10 | 10 | 10 | 6 | 9 | 9 | 9 |
| 13 | 12 | 10 | 10 | 10 | 10 | 9 | 9 | 9 |
| 14 | | | | | | | | |
| 15 | 12 | 10 | 10 | 10 | 10 | 9 | 9 | 9 |
| 16 | | | | | | | | |
| 17 | | | | | | | | |
| 18 | 13 | 10 | 10 | 10 | 8 | 9 | 9 | 9 |
| 19 | 12 | 10 | 10 | 10 | 10 | 9 | 9 | 9 |
| 20 | 13 | 10 | 10 | 10 | 10 | 9 | 9 | 9 |

Table 2: **Results of secure rendezvous over time.** Abbreviations: AV, AltaVista; Is, Infoseek; rv., secure rendezvous; act., actual pages existing. Data was not taken on blank days.

We examined two cases with separate subscribers. In the first case, we created 13 pages publishing them with MakeHtml. On the third day we withdrew three pages and moved one to a new location. In the second case, we created 10 pages and withdrew one on the third day. Each time we created or moved a page we submitted its URL to the search engines.

Table 2 shows the results of this experiment. For each case we show how many pages were returned by AltaVista (AV), InfoSeek(Is), secure rendezvous (rv.), and the actual number of pages containing the magic word (act.) for most of the first 20 days. HotBot and Lycos did not return any pages for the duration of our experiment. From the experiment we can see that the delay between submitting a page it appearing in a search is either 1–2 days, 9–12 days, or more than 20 days for these search engines. A secure rendezvous service will therefore have at least a 1–2 day latency when updating an index.

Two other surprising features appear in this data. First, AltaVista keeps links to stale pages around for the duration of the experiment (even after two months). Second, as observed in keyword-based rendezvous, AltaVista searches are not necessarily stable

from day to day. This instability reduced the targets reported by rendezvous in days 9 and 10 of case 2. We discuss ways to work around these problems in the next section.

From this experiment we conclude that secure rendezvous can be used when a few days of latency are allowed between publisher and subscriber. The amount of latency is very dependent on the search engines used. In these cases a small amount of extra publisher effort (running MakeHtml to update the security information and submit the new URL) allows the subscriber to provide a secure index with no additional effort. The main advantage of rendezvous is that even if the publisher isn't vigilant in running MakeHtml, the subscriber will eventually find out about the new location. (Although within the limitations of the matching rules.)

### 5.3 Search engines and rendezvous

We next consider the characteristics of an ideal search engine for web rendezvous, some reasons why existing engines are not ideal, and approaches applications can use to work around these differences.

For web rendezvous, an ideal search engine would have a very rapid time between submission and indexing, it would rapidly purge stale pages, and it would report consistent results over time. Accomplishing all of these goals is not easy, so we should mention that it does *not* need to index the full-text contents of the web for magic-word rendezvous. Our purposes require only indexing the presence and contents of any magic words.

Search engines do not meet this ideal for several practical reasons. With many pages in the web, engines cannot immediately fetch new pages to index, and even with page data they likely stage updates to the index database. Lawrence and Giles present several reasons search engines are slow to purge stale pages [11]. Finally, engines return inconsistent results because they are optimized for output to humans rather than other programs. Some search engines such as Inktomi's HotBot intentionally return partial results to achieve high availability or load balance [7]. For most humans, 100 hits quickly are more valuable than 200 hits in minutes or the message "cannot process your request".

Fortunately, secure rendezvous mechanisms can work around the later two problems. The secure algorithm includes a page validation step where we found pages are include the magic word and encrypted URL. Stale pages will fail this step. We can accommodate inconsistent results by treating the prior runs as input into the current run. In effect, we do a meta-search including ourselves as one of the sources.

The problem of rapid indexing at the search engine is more difficult. Creation of a search engine dedicated to web rendezvous would help. The database portion of a dedicated rendezvous search engine could be much smaller other engines, and priority could be given to submitted requests, thus allowing rapid update and indexing.

## 6 Related Work

### 6.1 Broken Link Detectors

There are both commercial and academic programs to automatically detect broken links in WWW. MOMSpider uses robot to traverse all links in a web site [5]. Broken links are reported to the maintainer. The broken link detectors only detects broken links automatically. Fixing the broken links still need to be done manually.

### 6.2 Persistent Names

There are a number of research groups investigating the issue of providing a Persistent Names for Internet resources.

The Internet Engineering Task Force (IETF) has been working on the issue of devising a general naming scheme for Internet resources. They introduce a global-unique, location-independent naming scheme, Uniform Resource Names (URNs) [3]. URNs separate object identification from location with the goal of providing persistent logical names for objects. As URN systems are developed and standardized, we expect them to provide stable names to web objects

WebLinker is a similar scheme which uses Local Resource Names (LRNs) as the logical name of object within a site [1]. All references to the objects of the site are through the LRNs. When an object is accessed, the LRNs is resolved by a linker server residing on that site. LRNs provide logical name to objects in one site. However if the site moves to a different server, the LRNs can not be resolved correctly. The site needs a URN as a logical name.

Persistent Uniform Resource Locator (PURL) is another scheme to augment URLs [12]. Instead of pointing directly to the physical location of an Internet Resource, a PURL points to a intermediate resolution service. The resolution service return the actual URL associated with the PURL to the client. The client can then continue the Web transaction in normal fashion.

Unfortunately, the persistent names approach provides only one component of a solution. Persistent

names identify individual object; they cannot point to multiple objects. A URN for the business office page would not adapt to the split of the business office into accounting and human resources, for example. In addition, many persistent-names systems simply provide a level of indirection: resolving a persistent name implies mapping it to a location. This approach moves the link maintenance problem from the index maintainer to the maintainer of the persistent name. finally, persistent names do not aid the task of index creation: each resource must be gathered and manually added to the index.

### 6.3 New Link Schemes

There are some other research groups aims to avoid the broken link problem by inventing new link schemes.

In W3Object Model, web resource are represented as W3Object which encapsulate internal state and well defined behavior [9]. Accessing W3Object is through the interface. W3Reference, a first-class referencing object, is used for addressing. W3Reference contains the location of an object. Referential integrity is guaranteed by *forward referencing*. An object is replaced with a forward reference at migration time—invocation can be redirected through the forward reference. However the chain from reference to the object maybe broken due to space crash or network partition. W3Object uses name service and callback to guarantee referential integrity. Reference holder register the object with some name servers. When object migrates, it informs the name server. Reference holder can get the path to the object from name servers. The reference holder itself can also register a callback to the object. The object inform the reference holder directly when it migrates.

In W3Object model, indirection invocation will traverse through a chain. The performance at invocation time is affected. Although short-cutting may be used to decrease indirections, it will take long time for reference holder to point to the object directly. It is difficult to deploy the name server in world wide web. The name server approach is somewhat similar to our work. Reference holder uses name server to access object. In web rendezvous approach, subscriber use existing search engines to access publisher. There is no extra work in construction of the name server in our Web Rendezvous model.

The Hyper-G system from University of Technology, Graz, Austria has a superficially similar architecture to the World Wide Web: servers provide documents to client browser [2]. Unlike WWW client, Hyper-G clients talk to a single Hyper-G server for the entire session. If information from remote server is needed, local server fetch the object and passes it to the client. Each Hyper-G server maintains its own document management system. Links in Hyper-G are not stored within documents but in a separate link database. The referential integrity between local resource is achieved by updating or deleting links to a resource when it is removed. Referential integrity for remote resources is maintained by propagating update information to remote servers using a flooding algorithm [10]. The update information contains deletion or migration of objects and creation or deletion of links.

All Hyper-G server are involved in the remote server update mechanism since the information is broadcast to all servers. The network traffic is proportional to the number of servers. It can not scale to the rapid growing World Wide Web.

The new link schemes may prevent broken links if widely deployed. However, switching the link structure in current World Wide Web to the new link schemes may require extensive work. In addition, the new link schemes can not solve the multi-way rendezvous problem. In the index making problem, subscriber still need excessive work to locate publishers.

### 6.4 Automatic Home Page Finders

Several groups have independently developed automated indexes of local pages. For example, some organizations automatically maintain lists of home pages or projects. Most of the examples we've seen appear to search an external database (for example, the list of the organization's home directories) or use CGI scripts. Thus these approaches either fail to address the problems of link forgery and modification or avoid these problems by assuming a trustworthy user community or with a solution which is limited to a single organization. We are not aware of an implementation which addresses all of the problems presented here.

## 7 Conclusions and Future Work

We have describe the problem of web rendezvous: allowing a subscriber to find links to a number of publisher's web pages about a given topic. Successful rendezvous faces several problems, particularly the gradual erosion of links over time as pages move (index degradation), placing too much work on the subscriber, and spamming (link forgery).

We suggested seven solutions to the rendezvous problem, and implemented two: keyword rendezvous,

which does not require publisher involvement, and secure rendezvous, which uses encryption to prevent link forgery. These approaches are somewhat complementary because of these trade-offs, but we believe that secure rendezvous is better when possible.

We evaluated both of keyword and secure rendezvous, examining utility and timeliness. With publisher involvement, secure rendezvous can update links in 1–2 days. Robust implementation of rendezvous depends on understanding of existing search engines. Based on our experiences we suggested ways rendezvous can be accomplished with existing search engines, and ways future engines can better support this kind of application.

We believe that wider use of web rendezvous can result in easier subscriber maintenance of large indices.

## Acknowledgments

Several people have made substantal contributions to web rendezvous. Joe Touch and Kedar Jog had the idea and did the initial implementation for keyword-based rendezvous. William Schedding implemented the second version of keyword-based rendezvous using the WWW::Search library. The authors also thank Ted Faber and Joe Bannister for comments about the paper.

## References

[1] Nikos Drakos Alberto Aimar, James Casey and etc. Weblinker a tool for managing WWW cross-references. *Computer Network and ISDN systems*, 28(1&2):99–107, Dec 1995.

[2] Keith Andrews, Frank Kappe, and Hermann Maurer. Serving information to the web with Hyper-G. In *Proceedings of the Third International World Wide Web Conference*, Darmstadt, Germany, April 1995. In Computer Networks and ISDN Systems, 27(1995).

[3] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (URI): Generic syntax. RFC 2396, Internet Request For Comments, August 1998.

[4] CPAN. Comprehensive perl archive network. http://feenix.metronet.com/perl/CPAN.html.

[5] Roy T. Fielding. Maintining distributed hypertext infostructures: Welcome to MOMspider's web. In *Proceedings of the First International World Wide Web Conference*, Geneva, Switzerland, May 1994.

[6] Roy T. Fielding, E. James, Jr. Whitehead, Kenneth M. Anderson, Gregory A. Bolcer, Peyman Oreizy, and Richard N. Taylor. Web-based development of complex information products. *Communications of the ACM*, 41(8):84–92, August 1998.

[7] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-based scalable network services. In *Proceedings of the 16th Symposium on Operating Systems Principles*, pages 78–91, St. Malo, France, October 1997. ACM.

[8] John Heidemann. Www::search. http://www.isi.edu/lsam/tools-/WWW_SEARCH/index.html, October 1996.

[9] D. B. Ingham, S. J. Caughey, and M. C. Little. Fixing the broken-link problem: The W3Objects approach. In *Proceedings of the Fifth International World Wide Web Conference*, pages 1255–1268, Paris, France, May 1996.

[10] F. Kappe. A scalable architecture for maintaining referential integrity in distributed information system. Technical report, Graz University of Technology, Austria, 1995. ftp://ftp.iicm.tu-graz.ac.at/pub/papers/p-flood.pdf.

[11] Steve Lawrence and C. Lee Giles. Searching the world wide web. *Science*, 280:98–100, 3 April 1998.

[12] PURL. Persistent uinform resource locator. http://purl.oclc.org.