Searching for Privacy: Design and Implementation of a P3P-Enabled Search Engine

Simon Byers[1], Lorrie Faith Cranor[2], Dave Kormann[1], and Patrick McDaniel[1]

[1] AT&T Research
Florham Park, NJ
{byers,davek,pdmcdan}@research.att.com

[2] Carnegie Mellon University
School of Computer Science
Pittsburgh, PA
lorrie@cs.cmu.edu

Abstract. Although the number of online privacy policies is increasing, it remains difficult for Internet users to understand them, let alone to compare policies across sites or identify sites with the best privacy practices. The World Wide Web Consortium (W3C) developed the Platform for Privacy Preferences (P3P 1.0) specification to provide a standard computer-readable format for privacy policies. This standard enables web browsers and other user agents to interpret privacy policies on behalf of their users. This paper introduces our prototype P3P-enabled Privacy Bird Search engine. Users of this search service are given visual indicators of the privacy policies at sites included in query results. Our system acts as a front end to a general search engine by evaluating the P3P policies associated with search results against a user's privacy preference settings. To improve system performance we cache unexpired P3P policy information (including information about the absence of P3P policies) for thousands of the most popular sites as well as for sites that have been returned in previous search results. We discuss the system architecture and its implementation, and consider the work necessary to evolve our prototype into a fully functional and efficient service.

1 Introduction

As people increasingly use the Internet for shopping and other activities, the level of online privacy concern is rising [14]. Many web sites have attempted to address privacy concerns by posting privacy policies and participating in self-regulatory privacy programs. However, it remains difficult for Internet users to understand privacy policies [15], let alone to compare policies across sites or identify sites with the best privacy practices. The World Wide Web Consortium (W3C) developed the Platform for Privacy Preferences (P3P 1.0) Specification to provide a standard computer-readable format for privacy policies, thus enabling web browsers and other user agents to read privacy policies on behalf of their users [7]. However, the P3P user agents available to date have focused on blocking cookies and on providing information about the privacy policy associated with a web page that a user is requesting [8]. Even with these tools, it remains difficult for users to ferret out the web sites that have the best policies. We

have developed a prototype P3P-enabled search engine called Privacy Bird Search that offers users the ability to perform Web searches that return privacy policy information along side search results.

1.1 P3P and APPEL

The P3P 1.0 Specification defines a standard XML format for a computer-readable privacy policy called a *P3P policy*. Although P3P policies contain some human-readable elements, they consist mostly of multiple-choice elements, which facilitate automated evaluation. A P3P policy includes elements that describe the kinds of a data a web site collects, the purposes for which data is used, potential data recipients, data retention policies, information on resolving privacy-related disputes, an indication as to whether a site allows individuals to gain access to their own data, and other information.

P3P became an official W3C Recommendation in April 2002 and has since been adopted by nearly a third of the most popular (top 100) web sites [4]. P3P user agent software is built into the Microsoft Internet Explorer 6 (IE6) and Netscape Navigator 7 web browsers. In addition, a P3P user agent called AT&T Privacy Bird can be downloaded for free and used as an add-on to the IE5 and IE6 web browsers. Other experimental P3P user agents are also available. In addition, a variety of tools have been developed to help web site operators generate P3P policies.

W3C also produced a specification for a language called A P3P Preference Exchange Language (APPEL) that can be used to encode user privacy preferences. APPEL is not an official W3C Recommendation; however, it has been implemented in Privacy Bird and other P3P user agents. APPEL is an XML-based language in which privacy preferences are encoded as rules that can be used to evaluate a P3P policy and control user agent behavior [6]. For example, an APPEL ruleset might specify that access to a web site should be blocked if the site collects data for telemarketing purposes without providing opportunities to opt-out.

1.2 Privacy Bird

AT&T Privacy Bird is implemented as an Internet Explorer browser helper object. The software adds a bird icon to the top right corner of the IE title bar. Users can configure Privacy Bird with their personal privacy preferences using a graphical user interface or by importing APPEL files. The preference interface allows users to select from preset high, medium, and low settings, or to configure their own custom setting. The user's preference settings are encoded as as APPEL rule set. At each web site a user visits, Privacy Bird checks for P3P policies. When Privacy Bird finds a policy, it uses an APPEL evaluation engine to compare the policy to the user's preferences. The Privacy Bird icon appears as a green "happy" bird at sites with policies that match a user's preferences. At sites with policies that do not match a user's preferences the icon appears as a red "angry" bird. The icon appears as a yellow "uncertain" bird at sites that have no P3P policy. A user can click on the bird to get a summary of the site's privacy policy, including the specific points where the site's policy differs from the user's preferences [9].

1.3 Related Work

A wide variety of web privacy tools are available that perform functions such as identifying web bugs, blocking cookies, reducing the amount of information transmitted by web browsers to web sites, and facilitating anonymous or pseudonymous browsing [8]. Several now-defunct dot coms offered privacy-related services including an electronic wallet linked to a privacy rating service (Enonymous) and a search engine dubbed "privacy friendly" because it did not have banner ads or cookies (Topclick). Neither of these services provided search results annotated with privacy information. Existing P3P user agents can make cookie blocking decisions based on P3P policies and display information about a site's privacy policies to users. However, none of these tools or services are designed to compare web site privacy policies or assist users in finding the sites with the best policies.

Tools and services are available to assist users in finding sites that match criteria unrelated to privacy. General web search engines find sites that match a user's text query. Google offers a SafeSearch feature in which sites with pornography or explicit sexual content are removed from search results. Shop bots and comparison shopping services find sites that sell a particular product, often offering the ability to compare these sites on the basis of price, reputation, delivery fees, and other criteria. However, currently none of these services offer comparisons based on privacy policies.

Studies have found that search engines are frequently used by most Internet users and that they serve as "gatekeepers" of Internet content [10]. Therefore, we believe that the search engine is the place where privacy policy information is likely to be of use most frequently.

After a user has conducted a web search and decided to visit a particular site, she has invested some time and effort and may be reluctant to turn away from that site even if she discovers that the site's privacy policy does not match her personal preferences. Without tools to assist her, she might have to visit several other sites before she finds one that has both the information or products she is looking for and a privacy policy that matches her preferences. In many cases such a site may not exist. As studies show that users typically do not visit more than two pages returned in a set of search results [13], it is unlikely that most users will undertake such a process to find a site that matches their privacy preferences.

A survey of Privacy Bird users showed strong interest in being able to do comparison shopping on the basis of privacy policies [9]. By adding Privacy Bird functionality to a search engine, we make it possible for users to determine which sites in their search results have policies that match their personal privacy preferences.

2 System Architecture

The Privacy Bird Search engine builds directly on the Google search engine service [2], and consists of four main architectural components: a policy acquisition module, a Google integration module, an APPEL evaluation engine, and a caching daemon. These components work in concert to acquire, maintain, and present a view of the P3P policies of sites returned by user search queries. This section gives an overview of the design and operation of this system.

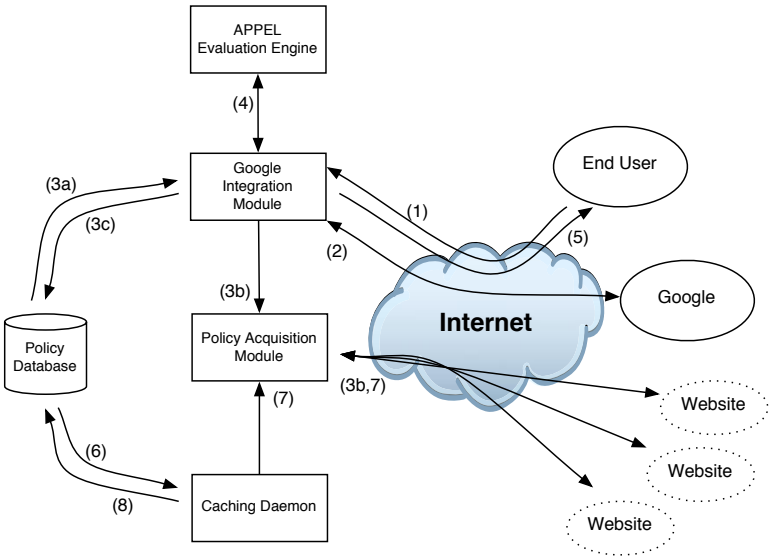

Fig. 1. Privacy Bird Search Engine Architecture - privacy evaluation results are generated by the evaluation of cached or real-time retrieved policies, and as directed by search results returned by the *Google* search engine.

The logical information flow and components of the Privacy Bird Search engine are illustrated in Figure 1. Users submit queries to the service through a search page provided by the Google integration model (step 1 in the figure). The integration model redirects the queries to Google, which returns results (2). The Google integration module checks the local cache for privacy policies associated with returned links. If available, the policies are acquired directly from the policy database (3a). If not, the policy is acquired directly from the link's parent website using the policy acquisition module (3b) and placed in the database for future use (3c). The policies are evaluated using the APPEL evaluation engine, and results returned to the Google integration module (4). Finally, the search results are annotated with a red, yellow, or green bird (depending on the evaluation results) and returned to the original end user (5).

Working independently of user queries, the caching daemon maintains the freshness of the policy database. The daemon periodically queries the database for all expired policies (6), and uses the policy acquisition module to refresh them (7). Once re-acquired, they are pushed back into the policy database (8).

We use the automated P3P policy acquisition tool reported in [4] to obtain P3P policies, and refer interested readers to that publication for further details. The remainder of this section briefly describes the design of the other core components.

Fig. 2. Search Page

2.1 Google Integration

The Google integration module accepts user queries, submits them to the Google search engine, and returns annotated results to the user.

Depicted in figure 2, users enter search queries using a Google-style interface hosted on our server. The integration module submits queries to the Google search engine and retrieves encoded results. The integration module then checks each URL in the search results to see whether it has a corresponding entry in the local P3P policy cache. If no entry is found, it attempts to obtain a P3P policy directly from the web site. Next the policies are evaluated and the results annotated and presented to the user in a Google-style results page.

Illustrated in Figure 3, our current system simply places an appropriate Privacy Bird icon next to each returned link. However, other presentation choices may be desirable. For example, one may wish to reorder the links so that those with green birds are presented first. In the extreme, one may eliminate all red or non-green birds entirely. We consider the social and political implications of different result presentations in Section 5.

All of these tasks are performed by website scripting. The Google integration module simply joins the services of Google, the evaluation engine, and the caching daemon. While intuitively simple, this requires some complex processing of the dissimilar artifacts used by each service. We consider the coordination of these services in depth in Section 3.1

We have also added an advanced searching feature that causes the Google integration module to return to the user any P3P policy information it has cached for a given site. The prefix “p3p:” followed by a host and domain name signals a search for P3P policy information.

As shown in Figure 4, a P3P policy search returns information about the location of a site’s P3P policy reference files, the content of any P3P headers, and the site’s cached P3P policies. In addition, the results page includes a hyperlink that submits the

Fig. 3. Results Page

site's policy directly to the W3C P3P Validator [3] to facilitate checking of policy syntax. This advanced feature has been designed primarily for use by web site developers and researchers.

[3] W3C maintains a free P3P validation service at `http://www.w3.org/P3P/validator.html`. This service can be used to check the syntax of P3P policies and policy reference files, and to verify that all P3P policies are properly located and referenced. This service is quite useful for debugging P3P-related problems on web sites.

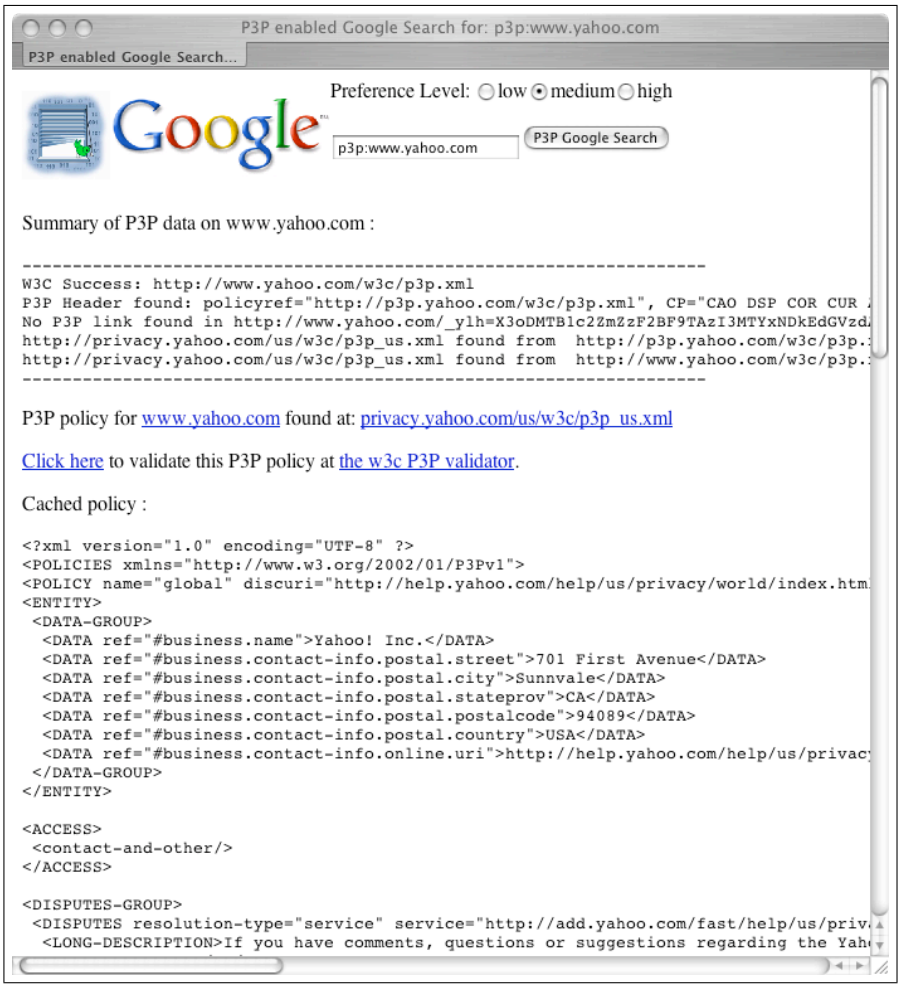

Fig. 4. P3P Policy Query Result Page

2.2 APPEL Evaluation Engine

A simpler P3P-enabled search service might establish a standard set of privacy preferences and evaluate all P3P policies against these preferences. However, this would eliminate one of the truly attractive features of P3P, choice. The APPEL evaluation engine gives us the ability to evaluate web site P3P policies against any APPEL-encoded privacy preference set without having to change any hard-coded rules.

For the purpose of demonstrating the feasibility of this concept in our prototype, we implemented an interface that includes three privacy settings, corresponding to three APPEL rulesets. In the future we plan to expand our interface to allow users to create or

import rulesets. These rulesets could be maintained on the server[4] or placed in a cookie on the user's computer.

The three Privacy Bird Search settings are:

- *Low*: Trigger a red bird at sites that collect health or medical information and share it with other companies or use it for analysis, marketing, or to make decisions that may affect what content or ads the user sees. Also trigger a red bird at sites that engage in marketing but do not provide a way to opt-out.
- *Medium*: Same as low, plus trigger a red bird at sites that share personally identifiable information, financial information, or purchase information with other companies. Also trigger a red bird at sites that collect personally identified data but provide no access provisions.
- *High*: Same as medium, plus trigger a red bird at sites that share any personal information (including non-identified information) with other companies or use it to determine the user's habits, interests, or other characteristics. Also trigger a red bird at sites that may contact users for marketing or use financial or purchase information for analysis, marketing, or to make decisions that may affect what content or ads the user sees.

Currently, P3P policies are evaluated in response to each end user query. The red, yellow, or green bird result is used for annotation, but not stored beyond that request, i.e., there is no attempt to persistently store evaluation results. This misses an opportunity to optimize request processing costs, but as yet we have not seen evaluation as a limiting factor. We expect that this decision will effect the future scalability of the system, and will be revisited as needs dictate.

2.3 The Caching Daemon

The Privacy Bird caching daemon maintains the P3P policy database. Called `pb_daemon`, this daemon runs in the background and constantly scans the Internet for website P3P policies. Policies are refreshed as they become stale, and new site policies are discovered and subsequently monitored as directed by end-user queries. In this way, the service learns from users which policies it should be monitoring.

The P3P policy database is simply a collection of ASCII files containing the P3P policies of the monitored sites. A sub-directory is created for each site whose policies are being monitored. Each subdirectory contains all P3P policies, reference files, and a single informational file named *state*.

The *state* file contains a single line with the current state of the P3P-related files associated with the monitored site. The encoded data indicates whether the site is a "static" entry (see Section 3.3), the number of hits since its last refresh, the time of its next refresh, the time of its last reference, and a flag indicating that it should (or should not) be purged from the cache as soon as possible.

[4] The ruleset could be mapped to a unique identifier held in a user cookie. This would eliminate the need to communicate the potentially large policy, and allow policies to be used across browsers.

The P3P *EXPIRY* element dictates how long a downloaded policy should be considered valid. This tells the daemon exactly how long it can continue to use a downloaded policy, and when is should be discarded. As directed by the P3P specification, where EXPIRY is not set, a default expiration of 24 hours is assumed. As expected, the caching daemon holds a policy for this period if no EXPIRY is specified. Non-existence of policies are also cached in a similar way, save that no policies are stored in the directory. A site with no P3P policy is checked once every 24 hours to see if they have added one.

`pb_daemon` constantly scans the database for new entries, and purges old policy files and refreshes others dictated by EXPIRY information, if any. New sites are detected by periodically scanning the local database. The Google integration module stores acquired policies in the database. The policies are subsequently discovered at the next scan by the daemon. Because discovery occurs via the filesystem, the daemon need not directly communicate with the other components of the architecture. This vastly simplified the construction of the tool as it obviated the need for building specialized protocols for inter-process communication.

3 Implementation

We have implemented a prototype version of our Privacy Bird Search engine. The prototype has only basic user interface features and has not yet undergone performance testing. Eventually, we plan to evolve this prototype into a service that we can make available to the public via the Privacy Bird web site.

The following considers some of the low level implementation issues and challenges we faced during the construction of the Privacy Bird Search engine.

3.1 Google Integration Module

The Google integration module is built directly upon the Google search engine API [3]. Based on the SOAP [16] and WDSL [5] standards, this API provides a programmatic interface to the Google search engine. We found the Google API to be both well documented and easy to use. It allowed us to quickly integrate its service directly with our perl implementation of the integration module. In essence, this API reduced the job of implementing an Internet search to a quick and rather painless exercise.

The Google integration module is written entirely in perl. User cookies containing privacy preferences are decoded using simple perl subroutines and results recorded in process-local data structures. Call-outs to the APPEL evaluation engine allow us to access the evaluation tools, and results are again stored in the local data structure. The results of the Google query are extracted from the documented API structures, and used in the presentation functions.

Because of users' familiarity with it, the current implementation models the results after the Google results page. The Google integration module simply merges the search results with template files to generate the HTML source returned to the end users. This will allow us to quickly alter the feel of the results page as needs and user desires dictate. We plan to experiment further with different presentations as the prototype matures.

3.2 APPEL Evaluation Engine

While the evaluation of APPEL is intuitively simple, its implementation in software is complex and often difficult to debug [8]. The Privacy Bird APPEL evaluation engine first parses a P3P policy and an APPEL ruleset. Then it normalizes both policy and ruleset in several ways, including removing comments and white space characters and inserting default attribute values for attributes that have been omitted. Because P3P includes a somewhat complicated data model in which data elements may either be enumerated (for example, #business.contact-info.telecom.telephone.number) or identified by category (for example physical contact information), the APPEL engine must expand all data references so that rules about data categories can be applied to policies that include data elements and vice versa. The APPEL engine then applies each APPEL rule to the P3P policy in order to determine whether any of the rules fire. Evaluation of each rule involves an eight-part test that is applied recursively. The Privacy Bird APPEL engine collects description strings associated with each rule that fires and returns them to the calling application.

As the preceding discussion indicates, building an APPEL evaluation requires enormous domain knowledge and testing. Thus, rather than implementing a new APPEL evaluation engine, we have extracted the APPEL evaluation modules from the AT&T Privacy Bird software package [1]. However, this decision introduced an entirely different set of problems.

AT&T's Privacy Bird is a helper tool for Microsoft's Internet Explorer browser and requires a Windows operating system. We designed Privacy Bird Search entirely on the UNIX platform. Hence, we needed to port the code to a UNIX platform. The APPEL evaluation modules in Privacy Bird use Windows native libraries, which complicated the move to UNIX. For example, the Windows `String` API was used widely in the APPEL evaluation code. Similar to the `string` object in the ANSI Standard Template Library [12], the Windows `String` objects provide APIs for the safe and efficient manipulation of resizable buffers of alpha-numeric characters. Because this API was not available on UNIX, we had to build a custom string object and replace every `String` API call with an equivalent one. This required a fairly deep understanding of a large portion of code itself.

Because the original modules were integrated directly into the browser, we had to construct a new interface to the evaluation engine. For flexibility, we concluded that a command utility was the best way to access the evaluation engine. This lead to the following simple interface:

`appel_eval` *[user policy] [site policies]*

The *user policy* contains the APPEL privacy preferences. Based on user preferences or by default, the current Google integration module currently selects an APPEL policy encoding one the of three privacy preference profiles (e.g., high, medium, or low). The *site policies* are the collection of P3P related files retrieved directly from a site under consideration. The result of the APPEL evaluation is a Privacy Bird decision, which is printed to standard output in numeric form (e.g., 1, 2, or 3).

3.3 The Caching Daemon

The caching daemon is written entirely in perl. A number of APIs for the manipulation and use of the files created and maintained by the caching daemon (e.g., state files, P3P policies) are directly implemented in the `P3PSEARCH` perl module.[5] The daemon itself is implemented in an executable `pb_daemon` perl script. As mentioned previously, `pb_daemon` does not communicate directly with the other components of Privacy Bird Search (e.g., via interprocess communication), but simply maintains the on-disk cache of P3P policies.

There are two classes of sites in the caching daemon. *Static* sites are those that are deemed important enough (e.g., appear frequently in searches) that fresh copies of the policies should always be maintained. Other non-static sites are those that the service acquires, but are free to be ejected from the cache should resource constraints mandate it. Static sites are identified in the caching daemon configuration as a single file of URLs, where each line contains the root of the site to which the policy applies. This file is read at startup and processed as described below.

The caching daemon is started using the following command line arguments:

`pb_daemon [-r` *refresh rate*`] [-d` *repository*`]`

Where the *refresh rate* is the rate at which the daemon rescans the P3P policy database, and the *repository* is the path to the the root of that database. The refresh rate defaults to 30 seconds, and the repository defaults to *./p3p_search_repository*)

The caching daemon operates as follows:

1. The daemon begins by initializing the list of static sites by reading the URLs included in the local (`static_urls.dat`) configuration file. Where a directory does not exist for a configured static site, one is created in the database. Where one exists, the local state file is read and its contents noted.
2. The daemon scans the database for directories not associated with statically defined sites. As before, the local state file is read and its contents noted.
3. The daemon queues the "stale" or missing policies for refresh (those entries which have not been refreshed in the last 24 hours, those with explicit and expired EXPIRY values, or those for which the daemon has no current information).
4. The policies are refreshed in the order in which they were queued. The policies and update state file are written to the appropriate directory once this process is completed. If the files cannot be retrieved, the stale files are removed and an empty state file written.
5. The main thread wakes every *refresh rate* seconds, rescans the database, and queues and refreshes the stale policies. This loop continues indefinitely.

The caching daemon does not refresh P3P policies directly. Rather, it forks a process for each site to be updated. This has the advantage that the daemon can easily parallelize updates. The current implementation forks a configurable number of update processes (by default, 5). Process termination is detected via the `SIGCHLD` signal, and the results obtained by rescanning the refreshed directory.

[5] The `P3PSEARCH` module also contains the APIs used by the Google integration module to acquire P3P policies from the end websites.

A key question for any such system is when to purge stale or unused site data. Some site policies may not be used frequently, or again. Hence, it is wasteful of resources to maintain the data. We are now considering several cache ejection strategies. The most obvious strategies would be to cap the disk space usage and employ a commonly used cache ejection discipline when the usage is exceeded (e.g, least frequently used, least recently used).

A second, possibly more appropriate, approach would implement a neglection threshold: any policy which is not used for some period of time should be ejected. This would not only save disk space, but reduce the overhead of refreshing P3P policies and the associated state maintenance costs. An interesting question is whether to eject at all, as currently there are probably fewer than 5,000 P3P-enabled web sites on the Internet. Of course, as the use of P3P grows, this will be become a more important issue.

4 Performance

The current version of Privacy Bird Search is a prototype that has not yet undergone any performance optimizations other than the introduction of a caching daemon. It is implemented on a single 300 MHz UltraSPARC. The following discussion broadly considers the performance of the architecture and ways it can be improved. Ultimately, the performance must support scaling to a very large number of users.

While the current implementation is stable, the performance is less than optimal. The amount of time to return query results is impacted mostly by the time it takes to process a Google search request using the Google API, the time it takes to fetch P3P policies from web sites or from our cache, and the time it takes to evaluate P3P policies against user privacy preferences. It takes approximately 400 milliseconds to complete a search request using the Google search API (i.e., total first request to last response byte, with 30 search results returned). This response time will be affected by network conditions. The time it takes to fetch P3P policies from web sites varies considerably depending on web server performance and network congestion. Evaluating a single P3P policy is fairly quick, taking about 180 milliseconds to complete. However, if most of the 30 search results have P3P policies, this can add a delay of several seconds. Most of the cost of each evaluation is in launching the perl interpreter and disk I/O. The actual processing time by the APPEL evaluation engine is about 16 milliseconds.

We have not conducted a rigorous performance study. However, we have timed a number of search queries to get a feel for where the biggest performance costs are and where we might focus our optimization efforts. For example, searching for the term “lorrie cranor” can take a little over 25 seconds to return 30 results if no policies are previously cached. The same query takes about 6.4 seconds where policies are cached. The overwhelming amount of time spent in the uncached test is spent fetching P3P policies. In our current prototype policies are fetched serially—clearly, fetching P3P policies in parallel would improve performance considerably. Not only would this reduce total acquisition time, but it would allow us to evaluate policies while waiting for others to be returned. To further improve performance would require reducing policy evaluation time. Again parallel processing would improve performance considerably. In addition, if all the code were binary we could reduce the substantial overhead associated

with launching the perl interpreter used to wrap the call to the binary evaluation tool. Minor performance gains could be achieved through optimization of the complicated APPEL evaluation code, including use of an XML parser optimized for this task.

Alternatively, caching evaluation results for each of our standard APPEL rule sets would improve performance, especially if caching was optimized to minimize disk I/O time. Each time the caching daemon retrieves a new policy the APPEL evaluation engine could be used to evaluate that policy for each of our standard APPEL rule sets (and perhaps also a set of popular user-defined APPEL files). The results of each evaluation could be stored as single bits.

When the policy cache is used, the performance of our prototype is reasonable for a prototype, but not for a production system. However, we believe the approaches outlined here will ultimately result in a stable and scalable system.

Note that because our implementation uses the Google API (as opposed to being integrated directly into a search engine) we had to build our own policy cache. A search engine may be able to implement privacy enhanced searching by integrating privacy tools with existing content discovery and management infrastructure. Moreover, we argue that the introduction of privacy features would represent a small incremental cost to an established search engine.

5 Discussion and Future work

We have implemented a prototype P3P-enabled search engine that allows users to determine which of their search results are on web sites that have privacy policies matching their personal privacy preferences. We have demonstrated the feasibility of adding P3P functionality to a search engine. The next steps are to address performance and scaling issues, experiment with user interfaces, and investigate the types of P3P policies associated with web sites that are found using typical search queries.

Although our prototype system was developed with the intention of evolving into a fully functional and efficient service, we have not yet addressed all of the issues necessary to insure that the system will scale. As discussed in the previous sections, cache ejection, the policy of not caching evaluation results, and related issues may need to be revisited.

A number of user interface issues warrant further investigation. We would like to find an interface design that is easy to use and helps users find sites that are most likely to both match their queries and their privacy preferences. Our prototype Privacy Bird Search returns search results pages annotated with Privacy Bird icons. Users can then scroll through the search results to find those hits that have green bird icons. However, users tend not to look past the first screen or two of search results, and some search queries may not return green bird icons in the first two screens of results. Alternative interfaces might reorder search results so that green bird hits appear on top. However, this raises a number of questions. Should yellow and red birds also be taken into consideration when hits are reordered? If so, should sites with a policy that does not match a user's preferences be ranked higher or lower than sites with no policy at all? Should an attempt be made to order red bird hits according to the number of deviations from a user's preferences? For search results that return a large number of hits, should re-

ordering be performed on only a subset of the hits (for example, the top 10 or top 100 hits)? Privacy Bird Search is designed as a front end to another search engine; however, a similar system built into a search engine could be positioned so that the privacy policy was taken into account in that search engine's ranking system. In that case a variety of options might be available to determine how much influence privacy policies have on ranking as compared with other factors. What kind of interface will best allow users to configure their preferences about privacy policies, ranking or search results, and other customizable features?

Ranking of search results on the basis of privacy policies raises usability issues as well as commercial and political issues. Because so many Internet users view the web primarily through the filter of a search engine, search engine ranking has enormous influence on what web sites users visit. This in turn influences the companies with which people do business and the ideas to which they are exposed [11]. If a popular search engine were to begin using privacy policies as a factor in search ranking it could influence web site operator decision-making about posting privacy policies and P3P policies. Depending on the approach to ranking, sites may have incentives to improve their policies. A search engine that offered only a standard privacy setting or that used a default preference set unless a user went to an advanced interface to configure preferences might influence sites to adapt their policies to match the standard or default setting. On the other hand, a search engine that ranked sites with no P3P policies higher than those with P3P policies that do not match a user's preferences might serve to discourage P3P adoption. Furthermore, because commercial sites are more likely to adopt P3P policies than non-commercial sites, reordering could reduce the chances that users would become aware of non-commercial sites.

In addition to annotating search results with privacy bird icons, we also plan to add a feature that will allow users to click on the bird icons to retrieve summary information about a web site's privacy policy, similar to the information provided by the Privacy Bird browser helper object [9]. This will include a summary of the site's policy, an explanation of why a site received a red bird, a link to any opt-out information provided by the site, and a link to the site's full human-readable privacy policy.

We previously developed software to gather data on P3P enabled web sites automatically. We have reported the results of our initial study of data collected using this software in [4]. However, no studies have yet attempted to use P3P to compare web site privacy policies systematically or determine the degree of variation of P3P policies across similar sites. Some factors that will determine the usefulness of a P3P-enabled search engine include the fraction of P3P-enabled sites among top hits to frequent search queries, and the fraction of those that tend to match users' privacy preferences. If few queries return hits that are both good matches to the query and have policies that match users' preferences, users are likely to find Privacy Bird Search more frustrating than useful. Future work in this area might include observations of Privacy Bird Search in use as well as simulations based on lists of most popular search queries.

Our work on Privacy Bird Search brings us a step closer to being able to provide privacy-related information to Web users at a time when it will be most useful. We were able to leverage our previous work developing the Privacy Bird browser helper object and automated tools for taking a census of P3P policies to develop this prototype. After

further work on user interface, performance, and scalability issues we expect to be able to make Privacy Bird Search available to the public.

References

[1] AT&T Privacy Bird, January 2004. http://privacybird.com/.

[2] Google, January 2004. http://http://www.google.com/.

[3] Google Web APIs Home, January 2004. http://http://www.google.com/apis/.

[4] Simon Byers, Lorrie Faith Cranor, and David Kormann. Automated Analysis of P3P-Enabled Web Sites. In *In Proceedings of the Fifth International Conference on Electronic Commerce (ICEC2003)*, October 2003. Pittsburgh, PA.

[5] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) 1.1*. W3C, 1.1 edition, March 2001. http://www.w3c.org/TR/wsdl.

[6] Lorrie Cranor, Marc Langheinrich, and Massimo Marchiori. *A P3P Preference Exchange Language 1.0 (APPEL 1.0)*. W3C Working Draft, 15 April 2002. http://www.w3.org/TR/P3P-preferences/.

[7] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, and Joseph Reagle. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*. W3C Recommendation, 16 April 2002. http://www.w3.org/TR/P3P/.

[8] Lorrie Faith Cranor. *Web Privacy with P3P*. O'Reilly and Associates, Sebastopol, 2002.

[9] Lorrie Faith Cranor, Manjula Arjula, and Praveen Guduru. Use of a P3P User Agent by Early Adopters. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, 21 November 2002. http://doi.acm.org/10.1145/644527.644528.

[10] Eszter Hargittai. The Changing Online Landscape: From Free-for-All to Commercial Gatekeeping. In Peter Day and Doug Schuler, editors, *Community Practice in the Network Society: Local Actions/Global Interaction*. New York.

[11] L. Introna and H. Nissenbaum. Shaping the Web: Why the Politics of Search Engines Matters. *The Information Society*, 16(3):1–17, 2000.

[12] David R. Musser, Atul Saini, and Alexander Stepanov. *STL Tutorial and Reference Guide: C++ Programming With the Standard Template Library*. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, MA, 1996.

[13] A. Spink, B.J. Jansen, D. Wolfram, and T. Saracevic. From E-Sex to ECommerce:Web Search Changes. *IEEE Computer*, 35(3):107–109, 2002.

[14] Humphrey Taylor. Most people are "privacy pragmatists" who, while concerned about privacy, will sometimes trade it off for other benefits. *The Harris Poll*, (17), March 19 2003. http://www.harrisinteractive.com/harris_poll/index.asp?PID=365.

[15] Joseph Turow. Americans and online privacy: The system is broken. Technical report, Annenberg Public Policy Center, June 2003. http://www.asc.upenn.edu/usr/jturow/internet-privacy-report/36-page-turow-version-9.pdf.

[16] W3C. *Simple Object Access Protocol (SOAP) 1.1*, 2000. http://www.w3c.org/TR/SOAP.