Service Portability

Why http redirect is the model for the future

Sumeet Singh Scott Shenker George Varghese

Abstract

The Internet provides tremendous flexibility, in that it can support a wide variety of services, and accessibility, in that these services can be invoked from anywhere. However, the current Internet architecture does not easily support service portability. *If users want their service names to be persistent then they must stick with the same service provider because service names, such as email addresses, are tied to administrative domains.*

In this paper we present a system called Permafind that gives users a persistent name for their services while allowing them to switch among service providers. Permafind applies to a wide range of services, and is immediately deployable. Serendipitously, Permafind also allows dynamic service insertion thus permitting many of the capabilities of more revolutionary proposals such as i3. Permafind embodies no technical innovation, but it does suggest that the notion of redirection, as embodied in HTTP, is a crucial feature for future service protocols.

1 Introduction

The Internet, through the generality of its architecture and the ubiquity of its deployment, provides a flexible, and pervasive communication platform. This fertile electronic soil has given rise to today's thriving ecosystem of Internet services. This evolving ecosystem has gone through at least three distinct developmental phases. In the beginning of the web, most Internet content was provided by individuals and nonprofit organizations. It took a few years for the commercial entities, such as news organizations and banks, to believe in the Internet but once their doubts were allayed, they adopted the web with a vengeance. Their adoption ushered in the second, more commercial, phase of the Internet, which has seen a new generation of corporate giants such as Yahoo!, Amazon, and Ebay, arise *de novo* from the Internet froth.

In recent years it appears that we are entering a third phase, sometimes called Web 2.0, in which an ever-increasing variety of *personal* services and content are being offered, though often hosted on commercial platforms. For instance, Myspace and YouTube have become extremely popular in recent years, and these are in addition to instant messaging, blogging, electronic mail, and web pages. While it isn't clear which category of content, commercial or personal, dominates the Internet, the hallmark of the last few years is the dramatic emergence of the latter.

The rise of these personal services presents a problem, one that already existed with ancient services such as email and web pages, but is now exacerbated by the proliferation of other modes of expression. While some adventurous individuals host their own services, the vast majority of these personal services are hosted by third parties, such as commercial providers, employers, or other organizations whose resources are not under control of the individual. There are a variety of reasons why users would like to move their services to a different hosting platform — such as pricing, change of employment, better service, etc. — but the current environment makes this difficult. Put concretely, users would like to *easily* switch their email from yahoo.com to gmail.com, and their blogs from blogger.com to newblog.com, as needed, without manually contacting all possible email correspondents or blog readers.

There are two main barriers. First, the Internet naming system ties services to administrative domains. Mail sent to an email address of sally@company.com will be delivered to a mail server controlled by company.com, and if Sally moves to a new company there is no way for this email address to follow her there. Second, despite years of effort there is no effective Internet directory that can serve as the "phone book" of the Internet. Search services do an amazing job with web pages, but for other services they perform significantly less well.

Thus, when someone moves their blog from Apcala to zoomshare, there is no way for a dedicated reader to know unless the person leaves a forwarding message. This also applies to email, which must be forwarded, and web pages, which must be redirected. In each case, the person must rely on the kindness of, or payments to, the previous hosting platform to maintain their forwarding address. This has created a significant barrier to service *portability*. Users tend to keep their services where they are, even when better opportunities appear, because of the significant inconvenience of moving to a new platform.

Given the role the Internet has played in disintermediation, which greatly increased the transparency and directness of many services, it is both archaic and ironic that for some of the Internet's most basic services, such

as email and web, individual users face such significant barriers to movement.

There are myriad papers that describe changes to the Internet architecture that would solve this problem (see, for example, [1–3, 5–7, 9–11, 13]). For instance, portability could be provided by a level of indirection, so that the name of a person's service is persistent but always resolves to the current provider. Similarly, an effective directory service would greatly alleviate the problem. However, neither of these are likely to happen any time soon, so our goal in this paper is to describe a solution that, while neither ideal nor elegant, is deployable in the short term. Beyond this short term focus, we are also looking for an approach that will push the architecture in a better direction in the long-term. To that end, we explore the basic principles one would need to follow when designing a protocol that would allow a service to be more portable. It turns out that HTTP, with its ability to redirect, is a model for such protocols [4].

We call the resulting system *Permafind*, and have a prototype available for experimental use (we encourage the reader to visit *www.permafind.com* and use the invite code 222 to register; this system is similar to the IKI web site http://http://www.iki.fi/index.html). While there is no ingenuity in the component mechanisms (indirection, redirection, relaying, and proxying), our intent is to turn a combination of these component mechanisms into a more universal mechanism that can be seamlessly invoked for all applications.

A side benefit of this approach is that it naturally enables a limited form of *service insertion*. For instance, one can have one's email sent through a spam filtering service of one's own choice.[1] The ease with which users can access third-party services might give rise to a much richer set of such services, thereby enhancing user functionality.

2 Designing for Portability

In this paper we set ourselves the following goals for our solution to service naming:

- *Persistence*: Each user should have a persistent, and therefore provider-independent, name for his/her personal services, ensuring that others can always reach them.
- *Generality*: To avoid separate *ad hoc* approaches for every application, the solution should work with traditional applications, newer Web 2.0 applications, and also future applications. In particular, future applications may run on top of other protocols than TCP or HTTP, so that one cannot rely solely on techniques embedded into these protocols (such as HTTP redirect).
- *Incremental Deployability*: The solution should be deployable without changing existing structures. In particular, services using this solution should be accessible by users of existing browsers and unmodified hosts.
- *Performance*: The portability solution should not cause significant loss in efficiency or increase in cost, both of which might deter its use by providers and users. Relevant performance measures include user-perceived latency and server throughput.
- *Ease of use*: The service should be usable by naive users, so it can't require users to run their own servers or configure their own DNS records.

These goals impose several constraints. Clean-slate designs, such as those based on flat and self-certifying names (*e.g.*, [13]), are ruled out by the need to be incrementally deployable. Thus we must provide persistent service identifiers based on the current naming system (domain names resolved by DNS). This would suggest an approach where each user has a personal and persistent domain name; *e.g.*, Bob could have a domain name bob.org (or, more likely, bob1753.org). To achieve generality, it must apply across all applications; for this we could use a hierarchical naming scheme such as:.

web pages: http://www.person.domain/path
blog: blog.person.domain
email: name@person.domain

The use of naming conventions such as these does not, by itself, solve the incremental deployability problem. Take the case of Bob Smith's blog, blog.bobsmith.org. If we look it up in DNS, we can get an IP address, but the requesting client (the one trying to reach Bob's blog) doesn't want an IP address; it needs the result to be translated (in the case of blogging) to a URL such as blogger.com/bobthebuilder. The problem is that the name given to Bob at blogger.com (*e.g.*, bobthebuilder) may be specific to the provider blogger.com and is thus not invariant. Unfortunately, DNS as it exists today only translates domain names to IP addresses, not URLs. Application-specific hacks like MX records only translate application-specific requests to an application-specific IP address (*e.g.*, returning the IP address of a mail server rather than the IP address used for a web page at that domain name). Such DNS modifications do not allow translation to additional provider-specific service tags such as usernames.

From a technical standpoint one could easily modify DNS so that it could return this additional information. Unfortunately, this approach not only requires substantial changes to DNS, it would also require all

[1] This feature is already available, such as in acm.org email, but our emphasis here is that Permafind explicitly sets itself up as a broker for these third-party services, allowing user-specified service insertion on a per user-name basis, as opposed to offering a single inserted service as a side-benefit.

clients (such as browsers) be modified in order to understand this new information. For instance, an unmodified browser trying to connect to Bob's blog will try to resolve bob.blogger.bobsmith.org and expect an IP address in return. Even if DNS resolves bob.blogger.bobsmith.org into blogger.com/bobthebuilder, the existing browser will treat this returning data as an IP address and fail.

Given that we have to stick with current DNS semantics, the next step in our search for a solution is to continue to use service-specific DNS names (as in bob.blogger.bobsmith.org), but instead of resolving them at the DNS level (*e.g.*, providing IP addresses), we generate application-level responses. To illustrate, consider the domain permafind.com which we have adopted for use in our system. A user that wishes to deploy a portable service (*e.g.*, email, blog) is given a persistent name bobsmith.permafind.com. Assuming Bob wishes to deploy a blog and email, he does so using two separate names email@bobsmith.permafind.com and blog.bobsmith.permafind.com.[2]

The trick is to have the Permafind server act as both a regular DNS server (to translate standard service requests to IP addresses) as well as an application server. The Permafind DNS server resolves DNS requests to itself. When the application-level request is then sent to the Permafind server, it attempts to direct the request to the appropriate server name. To do this, Permafind builds on three basic primitives: *redirection*, *relaying*, and *proxying*. In redirection, the Permafind server responds to an application request with a redirect message containing the appropriate service invocation. The key here is that this occurs at the application-level, which understands the appropriate semantics (such as the inclusion of a username in a URL). For applications that don't understand redirection, the Permafind server relays the request to the appropriate location. For applications that support proxying, the user can choose to have the Permafind server act as proxy; this will, as we will discuss later, allow the user to invoke further processing on the returning data. The Permafind server maintains a table, based on user input, of the appropriate mappings between the persistent Permafind service names and the current service locations.

We illustrate this with three concrete examples:

Redirection: Joan accesses Bob's blog by typing in the URL blog.bobsmith.permafind.com into her browser. The browser uses DNS to resolve blog.bobsmith.permafind.com into the IP address of the Permafind server. Joan's browser then sends an HTTP request, to which the Permafind server responds with an HTTP redirect pointing to the current URL, blogger.com/bobthebuilder.

Proxying: This is like relaying except that the Permafind server, rather than returning an HTTP redirect, forwards the request on as a proxy and receives the returning data. Proxying allows a user to specify, in addition to the location of his current service, any additional third-party functions that should be applied to the data. For example, for Instant Messenger services this could involve translating the IM request from one format (say Yahoo) to another (say AOL).

Relaying: Joan sends email to Bob at email@bobsmith.permafind.com. DNS resolves the MX record to the Permafind server. When the email arrives at this address the Permafind mail server relays the mail to the appropriate mail address, say bob@gmail.com.

Almost all current service interfaces support one of these three primitives. Moreover, the series of indirections (whether redirection, proxying, or relaying) may not be expensive in terms of throughput or latency. Thus, this approach is general, incrementally deployable, and reasonably efficient. It is also extensible; if a new breed of application X uses its own protocol rather than HTTP, Permafind will only have to add an application X server to do application X level redirection, proxying, or relaying. Since Permafind need only support a small subset of application X (enough to invoke relaying, proxying, or redirection), this is not very burdensome. We now discuss these mechanisms in more detail.

3 Detailed Design

The Permafind service allows users to sign up for subdomains of the permafind.com domain name, over which they have autonomous control. For example bob may register the sub-domain bobsmith, thus giving bob control over how the FQDN bobsmith.permafind.com is resolved at the permafind.com DNS server.

A Permafind name is processed in two steps.

DNS resolution: Clients use DNS to resolve the FQDN into the IP address of the permafind.com server. A easy and efficient solution to achieve this is to create a wildcard DNS entry in the DNS zone file for the permafind.com server. Using the wildcard DNS entry (allowed by all popular DNS servers), all sub-domains of permafind.com are automatically resolved to the IP address of permafind.com. This ensures that there is no delay for name resolution, so once a user reserves a particular sub-domain on the permafind.com server it is visible instantaneously.

Application-level translation: After the client receives the IP address from the DNS resolution, it then submits its application-level request to the permafind.com server using the user's service-specific FQDN (such as email@bobsmith.permafind.com and

[2] We use the formulation bobsmith.permafind.com as an example; it is easy to extend this to allow bobsmith to have several "accounts" attached to the bobsmith name, for example bobsdog.bobsmith.permafind.com

blog.bobsmith.permafind.com) . The corresponding service running on the Permafind server maps the service request to one appropriate for the current service location using information previously provided by the user. To remain compatible with existing clients, Permafind only maps an input name to an output name of the same type (*e.g.*, from URI to URI, or from email address to email address). The mapping also indicates whether a redirect or relay service is used.

If service insertion is used, the mapping provides the next stage in the chain of inserted services. Once again, redirect or relaying could be used to direct the data to the first stage in the service chain. For email, there is an elegant solution (that requires no changes at intermediate service providers) using a different Permafind name (per user) for each provider in the service chain. Figure 1 shows an example of service insertion for a receiver R (with Permafind address $R@P$) who requires the insertion of a spam filter (with Permafind address $R_1@P$ for user R) and a Virus checker (with Permafind address $R_2@P$ for user R)). $R_1@P$ and $R_2@P$ are internally assigned by Permafind to allow the same mapping database to facilitate both portability and service insertion. Unlike the insertion of Postini spam filtering by acm.org, the design allows *user-specified* service insertion on a granular *per-user* basis.[3]

For HTTP based services, Permafind will need arrangements with each service provider in the chain to relay to the next service. Difficulties with end-to-end semantics caused by mechanisms like cookies make it difficult to do general service insertion for HTTP based services. However, service insertion based on the initial control messages (*e.g.*, URL filtering) is easily possible.

In the actual implementation, the Permafind webserver maintains a mapping table using a MySql server for fast and uniform access. The current prototype has only a URI server and an email server. For all incoming HTTP requests, a local SQL query is made using the FQDN supplied in the URI and the resulting mapping is sent to the client using the HTTP-302 Found directive. HTTP-301 (Permanently Moved) will not work because clients would then bypass Permafind in the future. HTTP redirection ensures reachability for all popular Web 2.0 services such as blog, rss, podcasts etc. as well as traditional web-pages and thus allows incremental deployment.

The Permafind email server uses relaying instead of redirection. While redirect is part of SMTP, not all email servers support redirection correctly. Relaying also hides the final destination email address from the sender. Commercial providers such as acm.org and gmail.com already support email relaying so we will not elaborate further.

[3]Service insertion is not implemented in the current prototype.

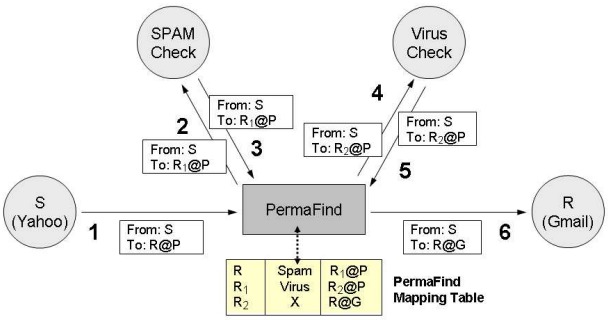

Figure 1: Steps in processing an email message from S to receiver $R@G$ with spam filtering and virus checking as inserted services. $..@P$ denotes a Permafind address. In Step 3 the message comes back from Spam Checking with $R_1@P$ as the To address. Permafind uses the mapping table (indexed by $R_1@P$) to direct the message to the Virus Checker with a To address of $R_2@P$. This in turn causes the virus-checked message to be forwarded to $R@G$ via Permafind.

Besides redirection and relaying, we anticipate a Permafind server being used as an *application level proxy*: a gateway that intercepts and rewrites control and data messages, thus providing additional services. For example, one gateway could convert between Instant Messenger protocols. A second gateway could translate between entirely different protocols and modalities, for example converting voice messages to email messages.

4 Looking Towards the Future

Permafind uses a set of standard mechanisms (relaying, redirection, and proxying) and one level of indirection (a very old idea in computer science). There are already commercial email services, such as acm.org, which offer relaying as a service and the IKI site offers both relaying and redirection. So what was our point in writing this paper?

In our defense, the current situation is far from ideal. These techniques are currently configured, deployed, and invoked on a per-application basis. For example, acm.org does not provide a web redirection service for say blogging or photo swapping. By contrast, Permafind places these methods in a unified framework, requiring no configuration on the client and straightforward account management (to keep mappings up-to-date) by the user.

Beyond this technical unification, the *combination* of indirection with redirection at the indirection point appears to be more powerful than indirection or redirection in isolation. Clearly, the intent of redirection was to allow portability, but redirection at the old service location is more problematic than redirection at an indirection point such as Permafind. Similarly, indirection followed by relaying is less efficient (because all the data passes through the relay) and less general (because many services use mechanisms like HTTP Cookies that may not work through indirect relays) than indirection followed by redirection.

We also believe that Permafind can provide a much-needed service. Even if technically boring, Permafind might allow service mobility, which is infrequent and painful today, to become commonplace and convenient. Further, Permafind allows flexible service insertion *today* without architectural changes, allowing users to compose services such as spam filtering and virus checking. We believe there is a great need for short term solutions to mobility and service insertion, even if they are limited in scope.

However, besides meeting *present* user needs, how can Permafind foster movement towards a *future* general architecture for service portability? We believe that the appropriate end point of Internet evolution would, like in [13], have persistent service identifiers and a flexible resolution mechanism that would return metadata that contained information about which application to use (such as HTTP) and what control data (such as the desired URL) should be issued. The Permafind server, already having all the appropriate metadata, can be seen as a forerunner to this resolution service. While now it only reveals this metadata through application-level actions, the Permafind server could easily be modified to support an interface that returns the metadata directly. This interface could be invoked by a new host mechanism that replaces the standard DNS query with one that recognizes the Permafind domain and, when called to resolve such domains, asks for the metadata directly and then issues the appropriate application commands.

These two methods could coexist. Unmodified clients would go through the two-step process described earlier while modified clients would access the broader interface and get the metadata directly. Such an approach, with Permafind servers offering a broader interface to be used by modified clients, could allow the Internet to gradually evolve towards a world with persistent identifiers and flexible service invocation. We view our first deployment of Permafind as an initial step in that direction.

Moreover, such a transition would help remove an internal contradiction. This paper is about service mobility, but our approach requires users to stick with Permafind. This involves committing to a redirection service (Permafind) rather than particular application-level service providers (Gmail, etc.). While this is less noxious, it is still far from ideal. However, if we begin using modified clients that use the broader interface to request metadata, these modified clients could bypass permafind.com altogether and do a direct lookup in another resolution infrastructure. Beyond architectural cleanness, modified clients can surmount two limitations of the current Permafind: first, application names can be bound to arbitrary metadata including different types of names (*e.g.*, binding URLs to phone numbers); second, general service insertion is possible without some of the limitations imposed by unmodified clients.

More specifically, the evolution path could be (a) Permafind encourages user mobility, (b) to bypass the two-step resolution (DNS plus Permafind) and add features, users start deploying modified clients, (c) these modified clients are pre-equipped to use resolution infrastructures other than the Permafind resolver, and (d) such a new resolution infrastructure might come into being, given that there is already a set of hosts ready to use it, and the Permafind resolver is no longer a monopoly service. While this evolution story is a long-shot, we aren't aware of more credible transition paths.

We now discuss various other issues that Permafind must confront.

Security: We can conceive several threats (and possible solutions) to Permafind:

Spam: Malicious users could spam Permafind by creating names that fill the Permafind database. This can be mitigated by mechanisms like Captchas [12].

Phishing: Phishers could use Permafind addresses to hide the final destination from client browsers. As a countermeasure, we could ensure a minimum Hamming distances between Permafind names so that citibank1.permafind.com cannot be registered if citibank.permafind.com is used. Second, Permafind could disallow mappings to names (supplied by Anti-Virus companies) known to be bad. Third, instead of transparent redirection or relaying, Permafind could return a temporary web page with an explicit link to the destination URL.

Hijacking Redirects: While we argue that Redirects allow service portability, the blind following of Redirects in current browsers is also the source of many security holes. One could anticipate browsers (or application level gateways) blocking redirects in the future. As a counter, we argue that redirects are the basis of too many popular applications today to be blocked. Second, observe that the fragility of redirects is caused by browsers believing redirects sent by anyone. If Permafind or a similar resolver is considered a trusted agent then a security association (using say https) can be used to authenticate Permafind redirects.

Data and Meta Data Portability: Much of this paper has been about service *name* portability. However, there is also the issue of service *data* and *metadata* portability. Examples of data include email archives and past blogs; examples of metadata include address books and buddy lists. Porting raw data without additional semantic tags for structure is conceptually easy. Unfortunately, consider email data with an associated date tag. When moving from Gmail to Yahoo mail, a tool can easily read Bobs 2 year old stored email at Gmail and write it to Yahoo. Unfortunately, there is no way for a user tool (with-

out help from Yahoo via some externally visible API) to store old email with a specified 2 year old date.

This could be solved by introducing new APIs for each application that allow data and metadata portability. For example, if email providers decouple storage from presentation, such that users choose their storage server (*e.g.*., Amazon S3 [8]) and the email provider presents email by reading from storage using standard, externally visible standard APIs, then the data mobility problem becomes easier. To switch email presentation services, the user can make the new email presentation point to the appropriate storage server. If the user switches storage servers, a simple tool can migrate the data using standard APIs. In general, we believe that a service like Permafind must address data and metadata portability issues to make service migration easier. There is a rich set of problems in this space with both short-term and general solutions.

Provider Countermeasures: So far we have assumed that application-level service providers will stand idly by while a redirection service such as Permafind allows customers more mobility. However, there are countermeasures service providers could employ, especially to discourage relaying.

For example, suppose a provider (say Gmail) adds a one way (and secret) hash of the destination service name to the message, and this secret hash is specific to the provider Gmail. Then if the email arives back at a Gmail receiver, the gmail receiver drops the mail if the hash is incorrect. Permafind cannot compute the hash when it changes the destination email address (it is a secret hash), but leaving it unchanged condemns the packet to be dropped. A provider of a service could certainly cite security concerns for such a check while using this measure as a deterrent to services such as Permafind. Fortunately, this type of "attack" is only possible for relaying, and relaying is only needed today for email, for which relaying services such as acm.org already abound. Thus adding such a countermeasure would likely result in users crying foul. In general any countermeasure that punishes email relaying via Permafind, should also punish email relaying via Gmail and acm.org which should be too unpopular to contemplate.

For redirection, it appears that it is difficult for service providers to take counter-measures because the redirection step is not visible to the provider: the behavior seen by the provider is the same as if the user contacted the provider directly.

5 Conclusions

By allowing users to switch to best of breed services at will, service portability encourages competition amongst service providers to provide better services. Service insertion creates the further incentive of changing service intermediaries at will. While there are clearly disincentives for existing service providers (incumbents), there are incentives for new aspirants to support a service like Permafind to allow easy adoption.

In this paper, we have described the Permafind design as well as an initial prototype. The Permafind design combines two well-known mechanisms: redirection and indirection (with relaying and proxying for compatibility). Redirection at the indirection point has advantages over redirection at the old destination, or indirection (and relaying). Besides portability, the design offers a general and flexible form of service insertion for email, and a limited form of service insertion for HTTP services. While the current system has limitations, it requires no changes to existing software or infrastructure while still providing service migration and service-insertion functionality. Thus Permafind is *immediately* deployable and not just *incrementally* deployable. Further, it appears possible to gradually migrate to cleaner approaches such as [13] via this approach.

References

[1] Adjie-Winoto, et al. The design and implementation of an intentional naming system. In *ACM SOSP*, Kiawah Island, SC, Dec. 1999.

[2] H. Balakrishnan, et al. A Layered Naming Architecture for the Internet. In *ACM SIGCOMM 2004*, Portland, OR, September 2004.

[3] D. Connolly. Naming and addressing: URIs, URLs, ... W3C Architecture Document.

[4] Fielding, et al. RFC 2616 : Hypertext Transfer Protocol – HTTP/1.1.

[5] B. Frankston. DNS: A safe haven. `http://www.frankston.com/public/ESSAYS/DNSSafeHaven.asp`.

[6] M. O'Donnell. Open network handles implemented in DNS, Sep. 2002. Internet Draft, draft-odonnell-onhs-imp-dns-00.txt.

[7] M. O'Donnell. A proposal to separate Internet handles from names, Feb 2003. submitted for publication.

[8] Simple Storage Service - Amazon S3. `http://aws.amazon.com/s3`.

[9] K. Sollins. Architectural principles of uniform resource name resolution, Jan 1998. RFC 2276.

[10] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[11] M. van Steen, F. J. Hauck, P. Homburg, and A. S. Tanenbaum. Locating objects in wide-area systems. *IEEE Communications Magazine*, 36(1):104–109, Jan. 1998.

[12] L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard AI problems for security. In *Eurocrypt*, 2003.

[13] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the Web from DNS. In *NSDI*, San Francisco, CA, March 2004.