

IBM Research Report

uServ: A Web Hosting and Content Sharing Tool for the Masses

Roberto J. Bayardo, Amit Somani, Daniel Gruhl, Rakesh Agrawal

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120



Research Division

Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

uServ: A Web Hosting and Content Sharing Tool for the Masses

Roberto J. Bayardo Jr., Amit Somani, Daniel Gruhl, Rakesh Agrawal

(bayardo@alum.mit.edu, somani@almaden.ibm.com, dgruhl@almaden.ibm.com, ragrawal@acm.org)

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

Abstract

One reason for the web's success is that it makes it extremely easy to access content. uServ is a system that makes publishing content just as easy, and at a very low cost. uServ harnesses existing desktop computing infrastructure, allowing its users to pool resources to provide *high availability* web hosting and file sharing. By using existing web and internet protocols (e.g. HTTP, DNS), uServ does not require those who access uServ-published content to install special purpose software. In this paper, we describe the design, implementation, and a successful intranet deployment of the uServ system, and compare it with several alternatives.

Keywords: web hosting, peer-to-peer (P2P) systems, edge of network

Wordcount: 7940

1. Introduction

People want to share files over the internet. Whether the files are simple web pages, audio clips such as MP3's, photographs, or other content, the preferred method of sharing files is through the web. While the web makes accessing content simple (almost anyone has access to a browser), publishing content on the web is far more difficult both because of technical hurdles and cost.

This difficulty in publishing content has led, at least in part, to alternative peer-to-peer (P2P) file sharing mediums and protocols such as Napster [[Nap](#)] and Gnutella [[Gnut](#)]. These systems allow you to easily publish as much content as you can store on your system's hard drive, and are for the most part free. Unfortunately, users who wish to access this content must install special purpose software, which limits the audience capable of reaching the content. (While Gnutella uses HTTP to support downloads, most newer clients forbid browser-initiated download connections, even if content were locatable within a Gnutella net using the browser alone).

We propose and describe the uServ system, which exploits peer-to-peer techniques to provide easy to use, low-cost *web* publishing of content. With uServ, ordinary PC owners can use their own hardware to cheaply put content on the web. Unlike simply installing and enabling something like Apache HTTP Server [[Ap](#)] or some other webserver on your own machine, uServ is a complete web hosting solution in that with uServ:

1. By simply running the software, you are immediately assigned a convenient domain name which always

directs to your site content, even if your ISP assigns IP addresses dynamically.

2. By pooling resources of a group of friends, content can remain accessible even after your computer is turned off.
3. Even firewalled users (or users who otherwise cannot accept inbound connections) can publish content from their own machine.

One challenge in building uServ is that content must remain accessible using standard web protocols (e.g. DNS & HTTP) despite impediments such as firewalls, which can be more easily overcome with special purpose protocols. For example, because a firewalled user cannot accept inbound connections, protocols such as Napster and Gnutella allow either the content requester or the content publisher to initiate the transfer connection. This workaround is not possible if content is to be browser accessible, since the HTTP GET specification requires that the browser (content requester) always initiate the connection [Fi+99].

Another challenge, which cannot be underestimated, is keeping the system simple. In our internal deployment of uServ, users who wish to publish a site install the *uServ peer node software* with a standard wizard based installer, enter their corporate intranet id and password, and two clicks later their first file can be accessible on the web at a convenient domain name based on their corporate e-mail address. Deployed as such on the intranet, uServ is a convenient alternative to mailing around e-mail attachments, which consume precious (and expensive) server-side storage. We see an internet-based deployment of uServ serving a class of users whose needs are not well met by existing solutions. Free web hosting systems impose highly restrictive space quotas, and also bombard the site visitor with advertisements, almost certain to discourage repeat visits. Other for-fee web hosting services require high monthly fees (or impose restrictive space quotas), and many in addition require technical expertise, such knowledge of FTP, not held by a typical web user. uServ is cost-effective because the end-user's machines do most of the work. The server-side components to uServ consist of a dynamic DNS server and a component we call the *uServ coordinator*, which for the most part serves as a "thin" broker for the various system functions. We believe the uServ service can therefore be profitably offered for a small yearly fee. Richer "add on" services (e.g. such as a uServ-hosted replicator) could be offered at a higher price.

2. Related Work

The usefulness of "personal" webservers for supporting file sharing and other collaborative tasks has been recognized for several years, motivating projects such as WebDAV [WG99] which aims to extend the HTTP protocol with functions that support web authoring. uServ does not require or use any HTTP protocol extensions. Instead, it strives to expand the audience that can make use of the existing protocol by reducing and often eliminating problems imposed by firewalls, dynamic IP address assignment, network outages, and limited system uptime which are common in household or personal web server deployments. While uServ does not currently support WebDAV, this would be a natural extension of the system.

There are numerous companies providing webserver software that allows your PC to serve your own website. The

BadBlue webserver in particular [W01] aims to make web serving lightweight and "personal", and also supports a peer-to-peer search service for locating content that is similar in principle to the Gnutella network. The MacOS Personal Web sharing function is another system that makes it easy to publish a designated folder on the web, and Microsoft's Personal Web Server function is similar. None of these systems, however, address any of the problems listed above (firewalls, dynamic IP & automatic domain name assignment, limited uptime & network outages) which prevent such software from providing a reasonable alternative to web hosting services for most users.

Many other companies provide software that supports file sharing, ranging from collaborative work tools such as Lotus Notes [Lot] or Groove [Gr01] to instant messenger systems such as ICQ [ICQ] and the aforementioned file sharing networks such as Napster. These tools appeal to a much more limited audience than uServ since they require special software for accessing content. For example, many people would not be willing to download and install special software just to casually view some photos from a friend, no matter how streamlined the installation process. We foresee photo sharing becoming one of the most common uses of uServ when deployed on the open internet.

XDegrees is a company that provides software which assigns "location independent" URLs to files throughout an organization, providing a uniform namespace for the organization's distributed information assets. Components of the system also provide intelligent storage services to improve availability and performance. At the time of this writing, the technical information publically available from XDegrees consists of only a brief whitepaper ([X01]), so a detailed comparison is not possible. The whitepaper does, however, clearly note that XDegrees employs a naming system that resolves the physical location of content based on its entire URL. In contrast, existing Web browsers consult the internet domain name system (DNS) to resolve the machine location using only the domain name portion of the URL (the DNS protocol predates HTTP and has no provisions for resolving IP addresses from a full URL [M87]). This implies that people who want to access XDegrees content must install and run special software that executes this XDegrees-specific URL resolution scheme. While XDegrees content URLs are prefixed with "http://", making them appear as standard WWW links, these URLs cannot be resolved by standard web software and protocols. By extending the URL resolution scheme, XDegrees allows content to be replicated and distributed at the granularity of a document. uServ in contrast replicates content at a site-level granularity, which we will show allows transparent access to replicated content without installing any special software.

The Farsite project [Bol+00] provides a serverless, fault-tolerant distributed filesystem. Like uServ, it replicates content across end user machines. Farsite aims to make your personal files always available through replication and redundancy, while using encryption to ensure that files are available only to those who are authorized to access them. While similar in some respects, Farsite does not solve the primary problem addressed by uServ, which is providing highly available file sharing via standard web protocols with very low cost.

Project JXTA [Sun] is an open source project led by Sun Microsystems which is creating a common platform that simplifies building a range of distributed services composed of addressable and communicating peers running on

arbitrary devices. JXTA technology could potentially simplify building a system such as uServ in a manner similar to how the Vinci distributed services architecture [A+01] simplified our uServ development tasks.

Yang and Garcia-Molina [YGM01] have studied various *hybrid* P2P architectures where at least some of the system is centralized. The focus here is on how to best handle queries for distributed, transient shared content. A dynamic search capability is not essential for uServ, since uServ sites are always accessed through location independent URLs. Many uServ sites maintain availability high enough to allow them to be indexed in the standard way by search engine crawlers. In fact, several hundred uServ sites in our deployment are already indexed in our corporate search engine. That said, many uServ sites remain more transient than the typical website. In addition we expect uServ sites to be more dynamic, given that we have made it extremely easy to add new content. We therefore feel that a Napster or Gnutella-like querying capability would be a valuable addition to the system, though an addition that is orthogonal to the features we describe here.

3. A Tour of uServ

In this section we describe our deployment of the uServ system, and its use from the perspective of an end-user. We provide technical details and protocol descriptions in later sections.

3.1 System Usage

We have made uServ available within our corporate intranet at IBM, where it can be freely downloaded and installed by any employee with intranet access. We currently advertise uServ as an experimental system to be used "at your own risk". Despite the warning, the system has already been used by over 1800 people spread across a dozen countries, and the userbase is consistently growing (mostly due to word-of-mouth). Its usage in the presence of our disclaimers demonstrates the need and desire to share files over the web, even in a technology company where there are many high powered and traditional alternatives available for free (e.g. Lotus Notes, existing web server tools, and network mounted disks).

At the time of this writing, anywhere from 800-900 users are active during the week, with 500-600 of their sites available simultaneously during peak hours. During non-peak hours, fewer sites are online, bottoming out around 300 during weeknights and 200 over the weekend. Our site replication feature has been available for only 3 weeks, and has as yet to be adopted by most users. We expect the variance in the number of sites available to reduce as more users upgrade their software to exploit this new feature. (Thus far under 10 percent of available sites are being served through a replica at any given instant.) We believe these usage statistics are extremely modest compared to what might be expected from an internet or "official" intranet deployment.

3.2 Getting Started

uServ adopts the philosophy that any system designed for the masses has to be extremely simple to use in every aspect. We have found it takes less than 10 minutes for an individual who knows how to use a browser to make

their first file accessible on the web using uServ, including the time required to download and install the software.

Every employee in most companies have an e-mail address that uniquely identifies them. At IBM, this address also has a direct mapping to an "Intranet ID" that is used for accessing various web-based applications. We automatically assign a uServ domain name for every employee based on this ID. For example, the e-mail address "bayardo@us.ibm.com" maps to the domain name "bayardo.userv.ibm.com". This means that locating someone's web site is as trivial as looking the person up in the employee directory or remembering his or her e-mail address.

Once a user downloads and runs the uServ installer, uServ starts up and requests this intranet ID and password for login. Unless the user specifies otherwise, uServ remembers the login information and connects automatically every time the system is restarted. After the initial login, uServ creates a brand new empty directory and populates this directory with a default homepage and a "private" subdirectory. The default homepage is populated with information extracted from the IBM corporate directory, and includes things such as job title, phone number, snail mail, and so on (**Fig. 1**). The user can manually change the shared folder at any time.

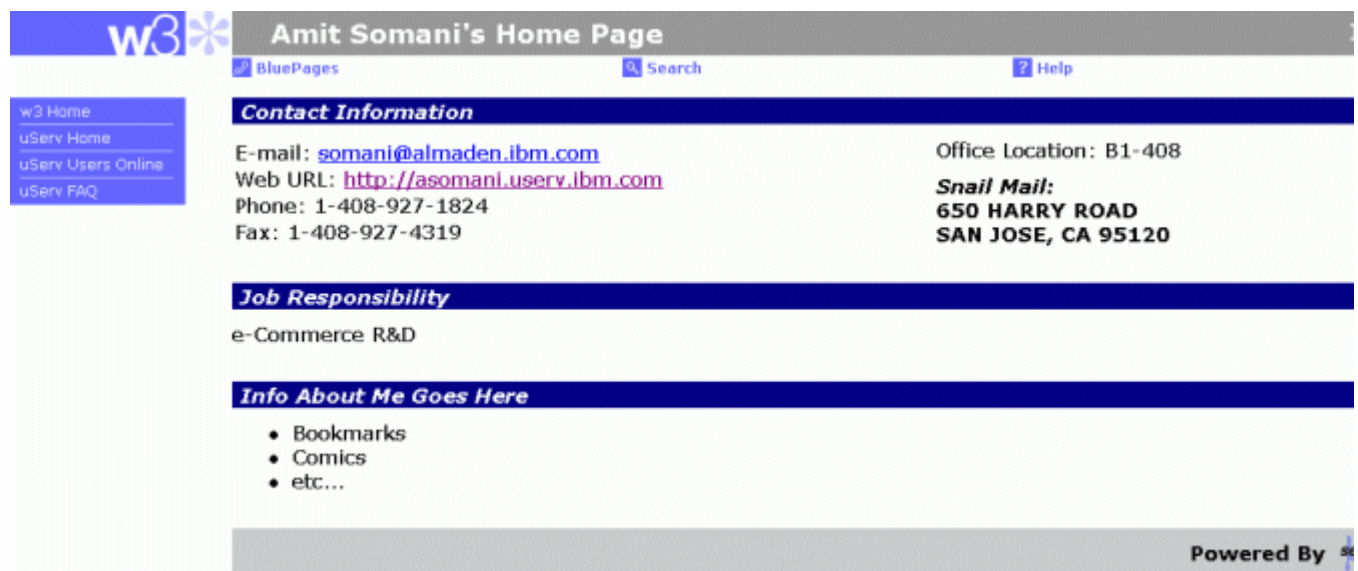


Fig 1. uServ generates a default homepage for each user.

uServ restricts web access to the private directory by requiring the username and password which was entered during login. All other files in our current deployment can be accessed without a password as long as the URL is known. A more sophisticated access control scheme is currently in the design phase. Access control is non-trivial in an environment where peers directly host content. In order to avoid compromised passwords, login information should never be directed to peers, but instead validated by a trusted third party. We elaborate on access control issues in uServ in section 6.

To share a file, a user can either copy it to his or her shared directory, or use a uServ specific feature which makes it even easier. With this feature, the user can simply right-click over the file and select "Publish to uServ" (Fig 2). The user is then prompted to choose a specific sub-directory in which to save the file, after which the file is written to the shared space and a URL pointing to that file provided to the user. This URL can be launched or copied to the clipboard with one click, making it easy to share the URL with others via e-mail or instant messaging. uServ also lets users publish entire folders of files with this feature by right-click-publishing on a folder instead of a file.

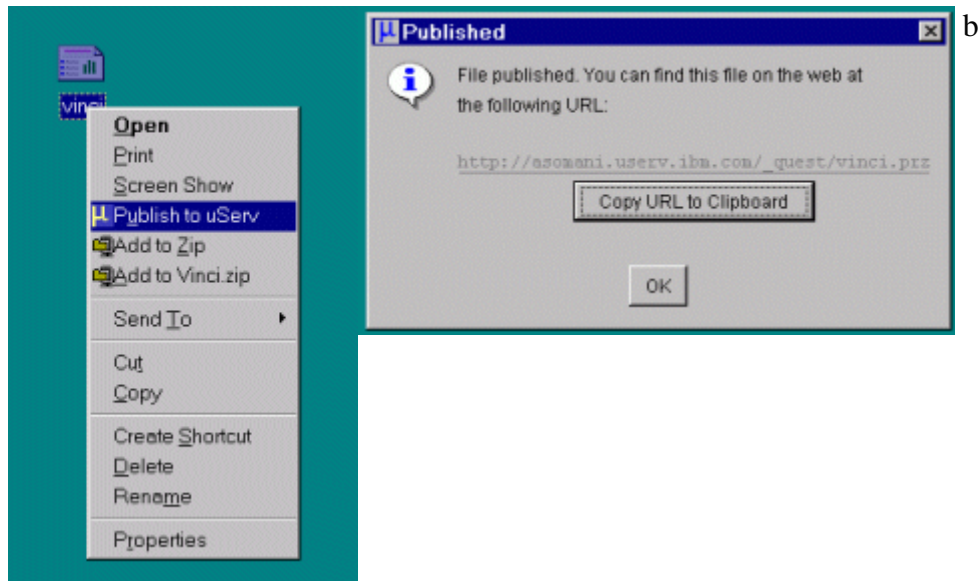


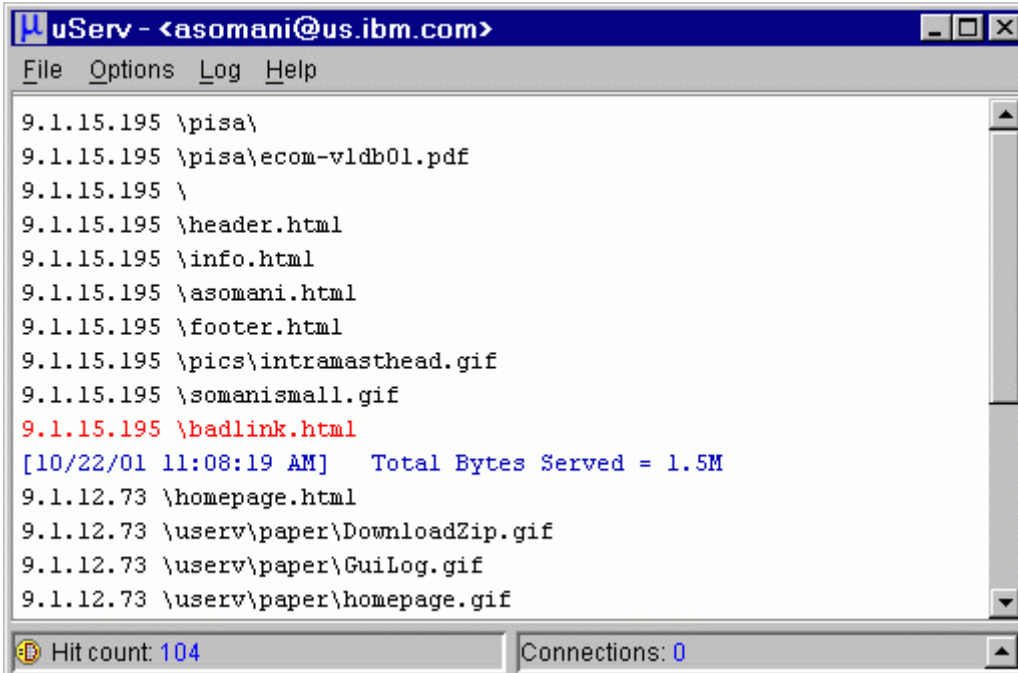
Fig 2. uServ lets a user quickly put a file up on the web.



Fig 3. Directory listing provided by uServ. Note the ability to download all files as a ZIP file in one click.

In our deployment, when a user visits a uServ site, uServ will by default list the contents of the shared folder (**Fig. 3**). We chose this behavior since we've found that most people like to share files without maintaining sophisticated HTML pages linking to them. Directory browsing allows users to find content without having to remember the exact URL. Users who do maintain HTML links to their content can rename their homepage.html file (or some other file) to index.html in order to have that file served in place of a directory listing. This behavior is consistent with that of other webserver software including Apache and Microsoft IIS. One unique feature of uServ is how it allows site visitors to download the entire contents of a shared directory hierarchy with one click in ZIP format (note the link in **Fig. 3**). Most users find creating ZIP files manually to be too cumbersome to use regularly, thus the feature is quite valuable when sharing multi-file content such as photo albums or source code trees.

The uServ GUI displays a log that lists any files that have been accessed, when they were accessed, and from which IP address (**Fig. 4**). This log also flags error requests (such as file not found) in red, which facilitates site debugging. These "voyeuristic" features give uServ a distinct advantage over other file-sharing methods such as e-mail attachments, which may remain unopened without the sender ever knowing. Users who are not interested in monitoring the activity of their site can close the GUI window. A uServ control-tray icon allows the GUI to be restored as desired.



The screenshot shows a window titled "uServ - <asomani@us.ibm.com>". The window has a menu bar with "File", "Options", "Log", and "Help". The main area displays a log of file accesses:

```
9.1.15.195 \pisa\  
9.1.15.195 \pisa\ecom-vldb01.pdf  
9.1.15.195 \  
9.1.15.195 \header.html  
9.1.15.195 \info.html  
9.1.15.195 \asomani.html  
9.1.15.195 \footer.html  
9.1.15.195 \pics\intramasthead.gif  
9.1.15.195 \somanismall.gif  
9.1.15.195 \badlink.html  
[10/22/01 11:08:19 AM] Total Bytes Served = 1.5M  
9.1.12.73 \homepage.html  
9.1.12.73 \userv\paper\DownloadZip.gif  
9.1.12.73 \userv\paper\GuiLog.gif  
9.1.12.73 \userv\paper\homepage.gif
```

At the bottom of the window, there is a status bar showing "Hit count: 104" and "Connections: 0".

Fig 4. uServ reports file accesses through its GUI as well as in a more traditional file-based log.

3.3 Replication

We have noted that with typical personal webserver deployments, once the user's system is turned off or removed from the network, the content is no longer accessible. This greatly hinders asynchronous collaboration, in which it is not known exactly when a shared file will be downloaded. uServ supports site replication and shared hosting in order to overcome this problem. In uServ, any user can list other uServ users ("slaves") who are willing to host their content when they are offline. These other users must also list in uServ the users whom they are willing to host ("masters"), thereby enforcing a two-way agreement. Many groups or teams in our company have at least one member who is willing to leave a desktop machine running continuously. This member is typically used as a slave by the other members of the team. Some people have multiple machines, e.g. a desktop and a laptop machine. We have found that these people tend to use their desktop machine as a uServ slave system and maintain the master copy of their site on their mobile laptop. As we have previously noted, thus far replication is a rather new feature in our deployment. As its use becomes more widespread we intend to perform a more detailed analysis of its usage.

Site replication is performed transparently to the user. Once the masters and slaves are specified, replicas synchronize with the master site automatically, and replicas are activated automatically by the uServ coordinator when the user disconnects, even when the user does not "properly" shut down.

Use of replicas is also transparent to the people who access uServ content. Regardless of who is serving someone's uServ content, it is always accessed through the same location independent URLs.

3.4 Proxying

Approximately 20% of our userbase use machines which cannot accept what are known as "inbound port 80 connections", which must be allowed for standard web server software to function. Several reasons prevent inbound port 80 connections, the most common of which is firewall software. Our corporate security guidelines mandate mobile laptop owners install firewall software. Others install firewall software because of general security concerns. While firewall software can be configured to allow port inbound 80 connections, quite often this configuration step is beyond the capability of the average user. Virtual private networks (VPNs), network address translators (NATs) [Sen01], and even the presence of other webserver software running on the same machine can also forbid or otherwise prevent uServ from accepting inbound port 80 connections.

In order to accommodate this population, uServ provides what we call *peer-to-peer proxying*. Put simply, in uServ, members of the uServ community who are able to accept inbound port 80 connections can be called upon to accept them on behalf of users who are not. A user who accepts connections on the behalf of someone else is referred to as a "proxy". By default, any user who runs uServ is willing to serve as a proxy for at most 4 other users. Users can change this limit or even disable the feature completely. We encourage laptop users and other machines with limited bandwidth connections to avoid serving as a proxy since proxying can be bandwidth intensive.

The uServ peer node software that a user runs on his own machine to publish a uServ site will detect if a proxy is needed when it first starts up. Should a proxy be needed, the uServ coordinator forwards the contact information of another uServ user willing to serve as a proxy. The user's machine connects to the proxy's machine which will then accept connections on his behalf. More technical details on proxying are provided in Section 4.

As with replication, the use of proxies in uServ is for the most part completely transparent to the end user. Whenever a proxy is used, uServ will non-intrusively notify the user via its GUI exactly which uServ user is serving as the proxy, and also encourages the user to check if his or her machine can be reconfigured so that proxying is not necessary. uServ also informs users who serve as proxies when and for whom they are serving. We have found a wide majority of our users (>80%) are willing to serve as proxies for the community. In most cases, a user notices no performance or bandwidth degradation when serving as a proxy. Proxying only consumes bandwidth when someone is downloading files from the proxied user's site. The CPU time consumed by proxying is typically negligible.

4. Technical Details

This section describes the inner workings of uServ. We overview the various distributed components that comprise the uServ system, and describe how they interact and operate to provide the features we described in the previous section.

4.1 System Components and Protocols Overview

The components of the uServ system are as follows:

- Browsers - any machine running a web browser for accessing uServ content.
- uServ peer nodes - these are the machines of individuals who have set up a uServ site by running the uServ peer node software. These components do all of the "heavy lifting" in that all content is served directly from them, not any centralized resource.
- dynDNS - a centralized component that speaks the DNS protocol for resolving uServ domain names to machine IP addresses.
- uServ Coordinator - a centralized component that provides user authentication, proxy and replica matchmaking, IP sniffing and firewall detection, site availability monitoring, and other "administrative" tasks. The coordinator is the first contact point of any uServ peer node, which must authenticate itself before uServ will set up the appropriate domain name to IP mapping with the dynDNS component.

The communication protocols used in uServ are DNS and HTTP (for supporting standard web browsers), and uServ specific protocols that are implemented using Vinci libraries [A+01]. The Vinci system includes high performance libraries supporting XML document exchange via a lightweight XML document encoding. By using

XML protocols and declaratively queried document models, Vinci applications can be easily evolved without breaking existing code. This evolvability proved invaluable during uServ development, allowing us to release numerous upgrades of the software with improved functionality, all the while maintaining backwards compatibility without adding undue complexity to our code-base. Because Vinci libraries are compact and fast, this evolvability comes with no serious drawbacks compared to using a low-level structure oriented (and hence brittle) binary protocol such as those employed by Napster, Gnutella, and most other P2P applications.

4.2 Accessing uServ Content

We have allocated a subdomain specifically for our uServ deployment (userv.ibm.com), and assign each user's site a unique name in this space corresponding to his or her IBM e-mail username. For example, the domain name bayardo.userv.ibm.com is assigned to bayardo@us.ibm.com. uServ takes advantage of dynamic DNS so that a browser can map these domain names to the location (IP address) of an available peer node capable of serving the requested uServ content. In a typical scenario, the uServ DNS maps a domain name to the machine of the user to whom the domain name belongs. However, should this machine be offline, it could instead map to the machine of another uServ peer node who is capable of serving the content from a site replica. In the third case, if the user's machine is firewalled, this could instead map to a machine which is serving as a proxy for the site.

Our uServ implementation uses BIND [\[AL01\]](#) to provide the dynamic DNS service. Recent versions of BIND allow updates to be performed on a running nameserver. This allows the uServ coordinator component to immediately push any updates to the DNS server. These entries have a very short time to live (2 minutes), assuring that changes in the hosting machine are quickly propagated (e.g. if the host goes offline and a replica takes over).

We describe in turn each of the three different methods in which uServ content can be served.

1. Basic: A uServ peer node is online and capable of accepting inbound connections, and serves its own site.
2. Peer-hosted: uServ peer node is offline, and a replica of its site is served by another uServ peer node.
3. Proxied: A uServ peer node is online but unable to accept inbound connections, and serves its site through a proxy which accepts connections on its behalf.

To keep the presentation clear, we only describe cases where a peer node assists exactly one other peer node by either serving as its proxy or serving a replica of its site. Note though that any peer node in uServ is capable of simultaneously serving as a proxy for multiple users at once, while also simultaneously serving replicas of multiple uServ sites.

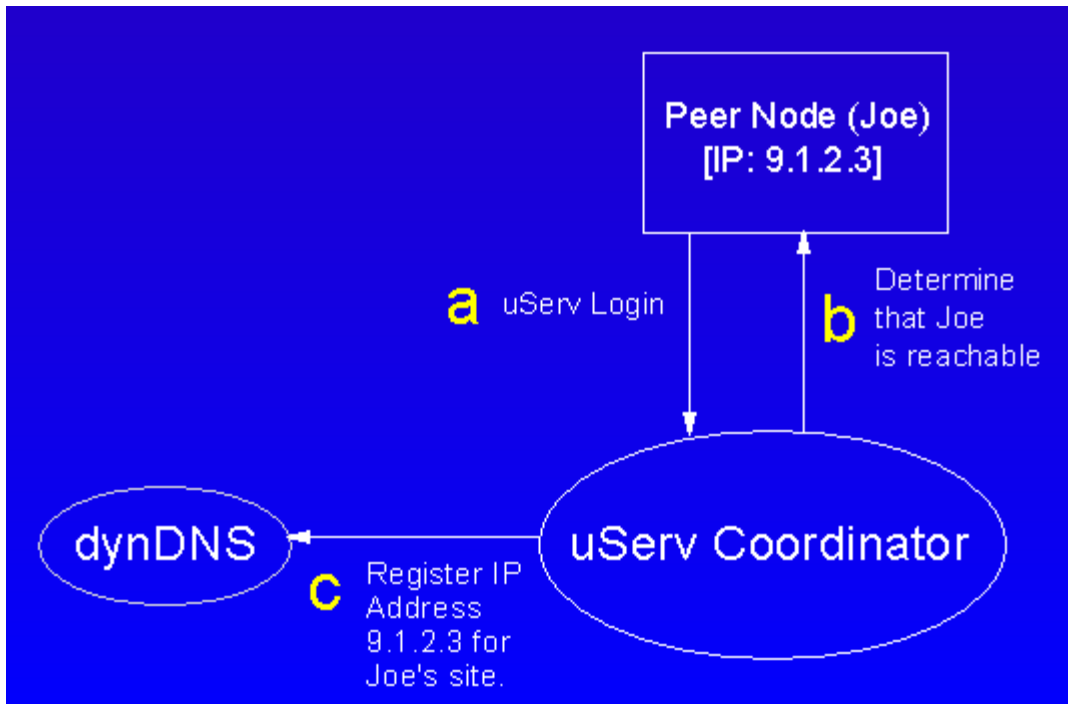


Fig 5a. A uServ peer node that can accept inbound connections comes online.

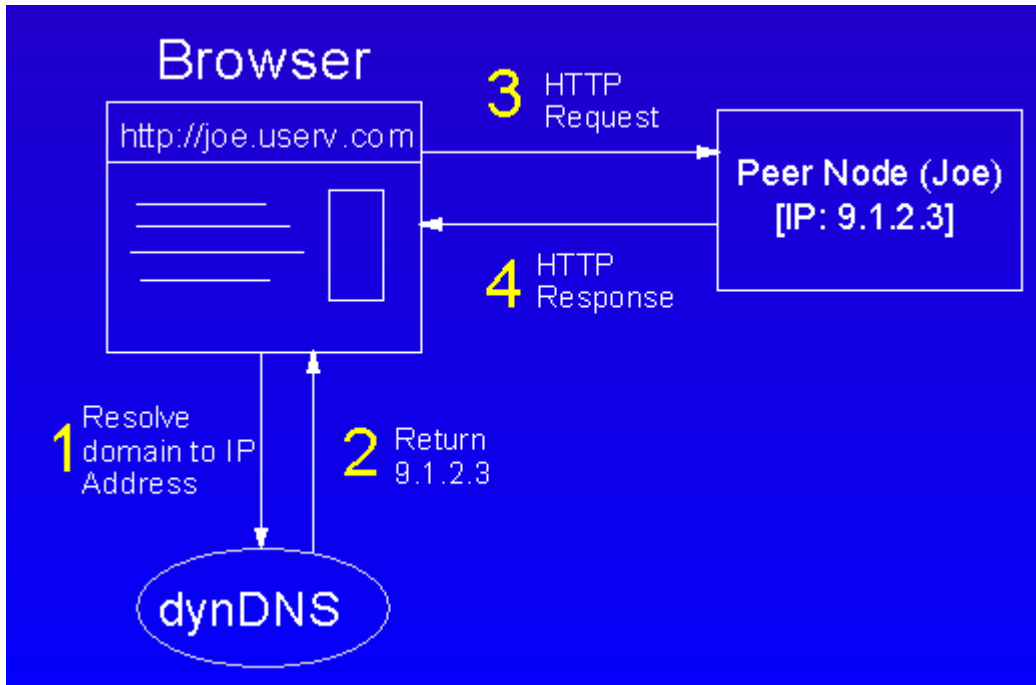


Fig 5b. Accessing content from a uServ peer node that can accept inbound connections.

4.2.1 Scenario 1: Basic

Fig. 5a depicts the initialization step for the first scenario where a user's machine is capable of accepting inbound connections. The peer node, in the depicted case run by a user named Joe, comes online and authenticates itself with the uServ coordinator. The uServ coordinator successfully establishes a connection back to Joe's peer node which signals that it can accept inbound connections. The coordinator immediately updates the DNS entry of Joe's site with the IP address Joe's machine.

In **Fig. 5b**, a standard web browser is used to access Joe's site from some arbitrary network connected machine. The user of the web browser does not need to install uServ or other special software on this machine. The browser resolves Joe's domain name to Joe's machine, and executes an HTTP request to retrieve the desired content. Though the figure depicts the browser communicating directly with the uServ DNS, the DNS protocol allows the browser to communicate with a local nameserver or fetch the IP address from a local cache. Ultimately, however, the domain name to IP mapping information arises from this uServ dynamic DNS component.

This scenario is what is provided by dynamic DNS services existing on the internet today [DNS] (minus the inbound connection check). The uServ system is unique in that it can also serve content in the two remaining ways for higher availability.

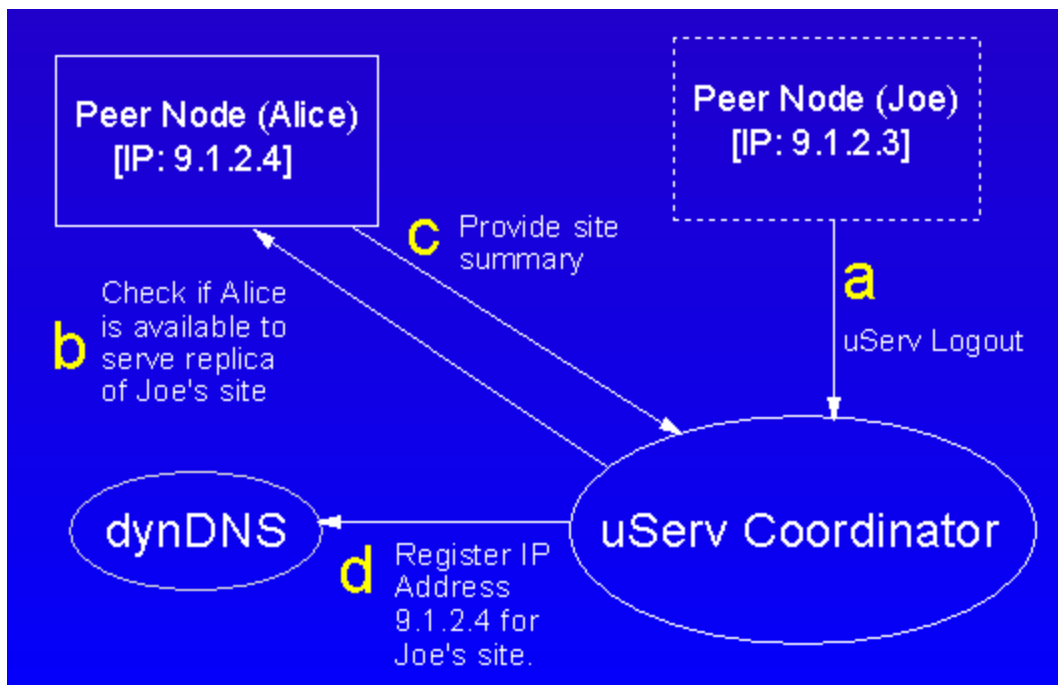


Fig 6a. A uServ peer node goes offline and another node that is replicating his site takes over.

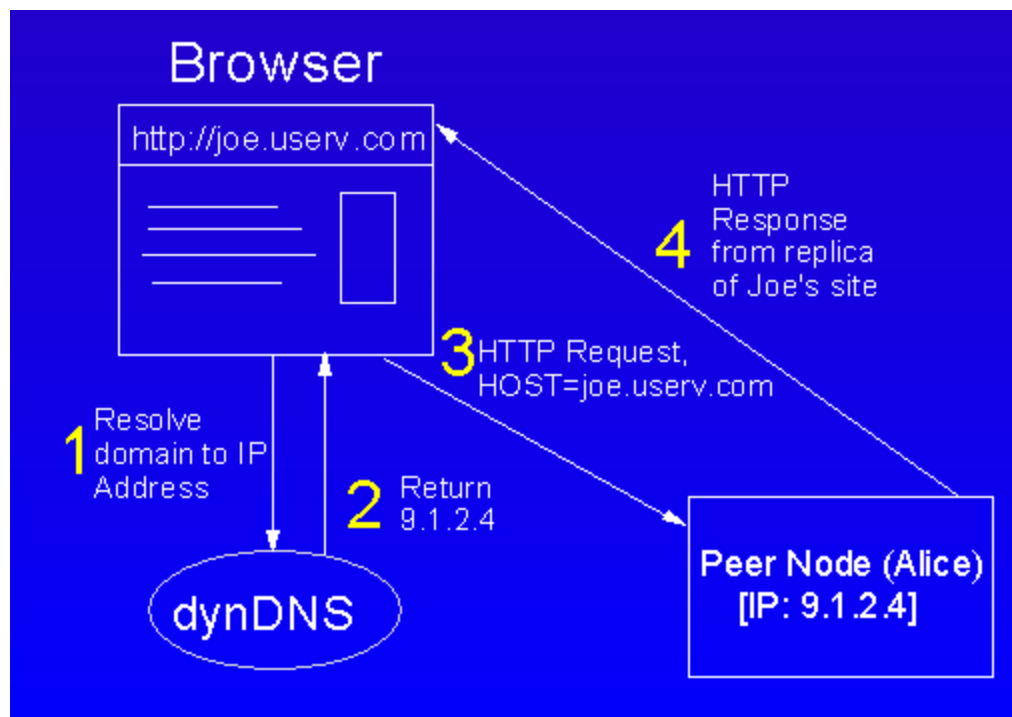


Fig 6b. Accessing content from a site whose peer node is offline.

4.2.2 Scenario 2: Peer Hosted

The next figure (Fig. 6a) depicts a scenario where Joe's machine goes offline (for whatever reason). Sometime before going offline, Joe and Alice agreed to allow Alice's peer node to serve Joe's content while Joe's peer node is unavailable. (The protocol for maintaining replicas is discussed in Sec. 4.3.) When Joe disconnects, the coordinator will check if Alice's peer node is available and willing to serve Joe's content. Alice indicates willingness by returning a site summary (essentially a checksum plus timestamp) of Joe's site. The coordinator may use this summary to determine whether to activate Alice's replica. In our implementation, if Alice is the only replica, the coordinator will activate her replica unconditionally. The summaries are only used to determine which of multiple replicas are the most up to date. Assuming Alice is the only available replica, the coordinator activates the replica by updating the IP address for Joe's site to the address of Alice's machine. If a replica goes offline, the coordinator will determine whether another replica is available and activate it. Should no replica of Joe's site be immediately available, the coordinator will monitor newly active peers in case one should come online.

After the replica of Joe's site is activated on Alice's machine, web requests for Joe's content are directed to Alice's peer node. Alice's peer node checks the value of the HTTP HOST header within the incoming request. Browsers will set the HOST header value to the domain name used to resolve the IP address of the requested site. In this case the web request will contain Joe's domain name, which causes Alice's peer node to return the requested content from the replica of Joe's site (Fig. 6b).

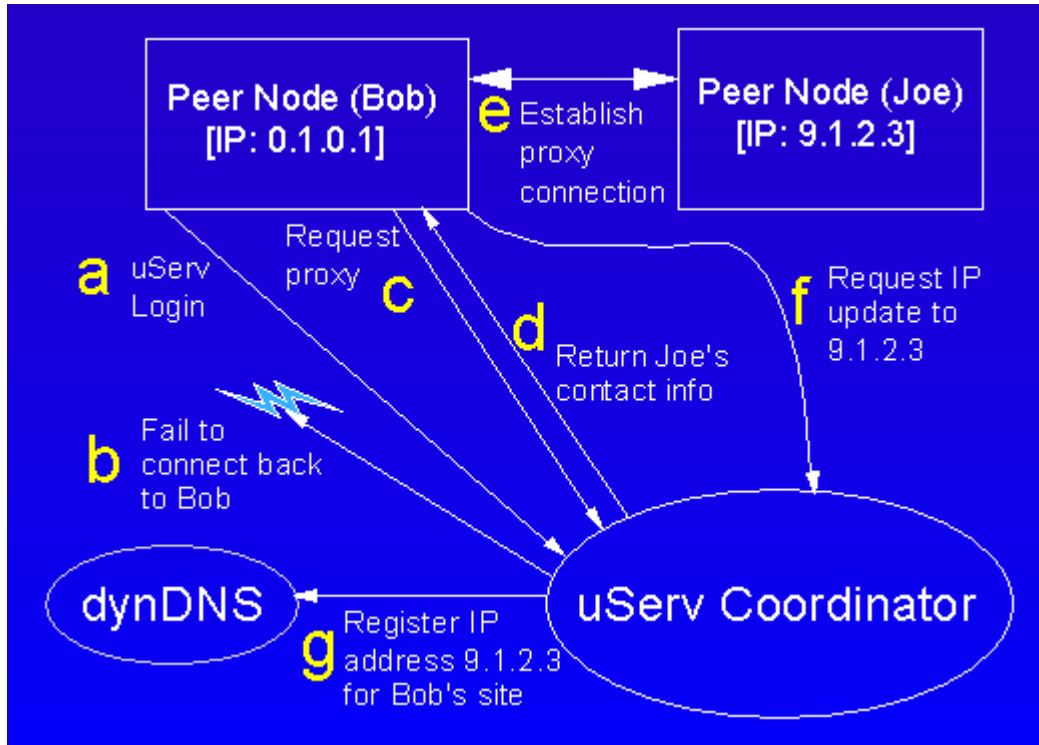


Fig 7a. A uServ peer node that cannot accept inbound connections (Bob) comes online.

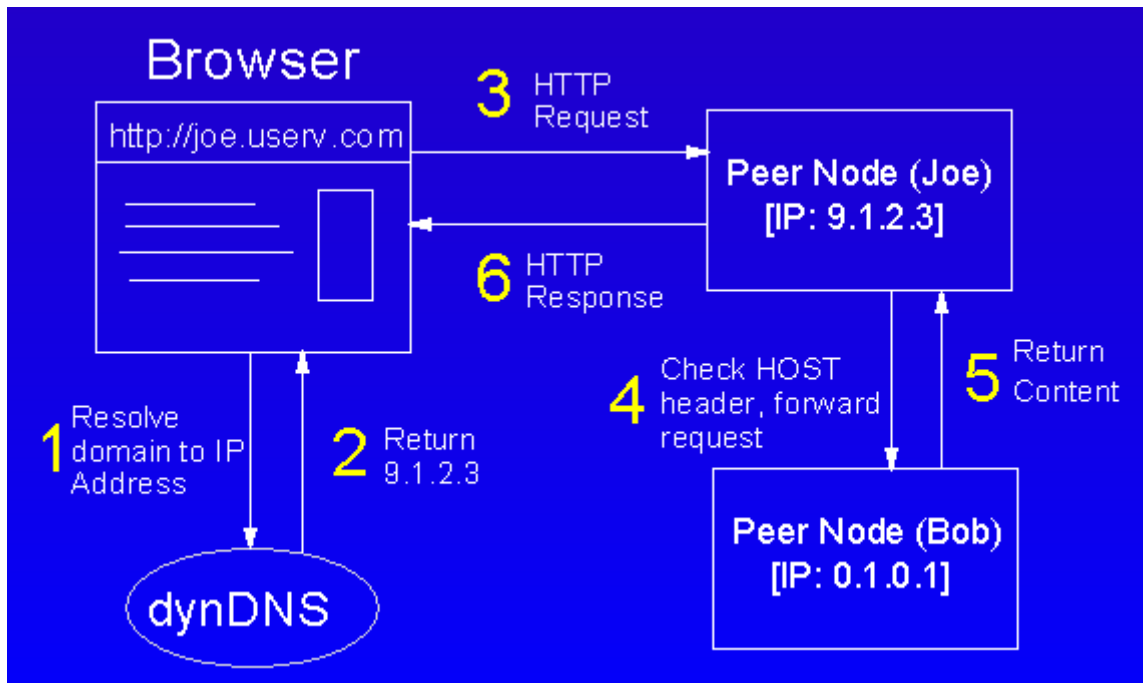


Fig 7b. Accessing content from a peer node unable to accept inbound connections.

4.2.3 Scenario 3: Proxied

In the next and final scenario (**Fig. 7a**), imagine again that Joe is online and capable of accepting inbound connections. Another user, Bob, comes online and registers with the coordinator, which is unable to open a new connection back to him. Bob's peer node recognizes that it didn't receive the expected connection from the coordinator, indicating it is incapable of accepting the necessary inbound connections to serve its own content. Bob's peer node therefore requests that it be directed to an available proxy. The coordinator responds with contact information of an available proxy, in this case Joe. The coordinator returns its response through the connection established by Bob, so an inbound connection is not needed to get this information to him.

Contact information consists for the most part of an IP address and an authenticating token. Bob's peer node uses this contact information to establish an outgoing, persistent connection with Joe's peer node, and reports back to the coordinator that a proxy connection was successfully established. The coordinator updates the DNS entry of Bob's site with the IP address of Joe's peer node. This persistent connection will be used to forward content requests from Joe's peer node to Bob's without having to establish an inbound connection to Bob. Should the persistent connection become broken, Bob's peer node will immediately attempt to re-establish it. If this fails, then the scenario restarts with Bob asking the coordinator for contact information of another available proxy.

The protocol spoken across the persistent proxy connection is not HTTP, but instead a uServ-specific protocol allowing multiple requests to be served in parallel on a single connection. A special protocol is used here because the HTTP protocol requires no more than one request be active at a time on a single connection. Browsers will often open multiple concurrent connections to a site at once to, for example, allow multiple images to load concurrently. By using a special protocol, uServ peers can parallelize proxied content requests while maintaining only a single persistent connection. This proxied protocol has the added benefit of not suffering from the high connection establishment overhead of multiple concurrent HTTP requests, thereby providing improved performance. Note that this special protocol need only be spoken between uServ peer nodes, and not by machines requesting the content.

The next figure (**Fig. 7b**) displays what happens when a browser attempts to access Bob's content. In this case, the browser directs the HTTP request to Joe's peer node, which performs the HTTP HOST header check and determines the request is intended for Bob's content. Joe forwards the request to Bob's machine through the previously established persistent connection (thereby not requiring it establish any inbound connections with Bob). Bob returns the requested content to Joe who returns it back to the browser through the HTTP response.

Proxying is bandwidth intensive. We delegate the task to peer nodes instead of performing it centrally in order to spread the load across the entire network. A proxied request roughly doubles the bandwidth and latency required, and will consume the proxy's bandwidth in addition to the bandwidth of the node hosting the requested site. It is possible to have a proxy node cache often requested content from the proxied user in order to lessen the consumed

bandwidth and latency. We are currently extending our proxying protocol to allow for such caching. This involves simply adding something similar to the HTTP HEAD request to the multiplexed download protocol to allow a proxy to determine if cached content needs to be refreshed.

Another potential optimization might be to have Bob directly forward the HTTP response back to the browser instead of routing it through Joe. The problem with this idea is that the HTTP protocol requires that the HTTP response travel down the same incoming connection as the request. It is possible in some situations to have Bob spoof the IP packets to make them appear as a response from Joe (e.g. see the Triangle Boy system for anonymous and secure web access [\[Safe\]](#)). Unfortunately, any such hack involving IP spoofing would be foiled by software or hardware that performs IP rewriting, including SOCKS proxies [\[Per\]](#) (which are commonly used for outbound firewall traversal) and network address translators [\[Sen01\]](#).

4.3 Maintaining Replicas

The uServ peer nodes are themselves entirely responsible for the bulk of replica maintenance. The uServ coordinator's job with respect to this task is simply to provide the contact information and authenticating tokens necessary for sites to directly (or via a proxying peer node) communicate with one another. Because of the obvious security implications, uServ requires permissions to be granted in both directions before the coordinator will activate a site replica. That is, a uServ user must designate other uServ users who are allowed to host her content, and also those users whose content she is willing to serve.

The site synchronization scheme we have implemented is designed with the assumption that the typical site change involves the addition or removal of files from a site, with file modifications taking place less frequently. In most cases, this scheme requires very little data to be exchanged between sites in order to keep a replica up to date. Some users in our deployment are maintaining replicas of several gigabytes and tens of thousands of files.

In our synchronization scheme, *slave* sites (sites which host replicated content) initiate contact with their *master* sites, and also initiate content synchronization when necessary. A slave determines when its replicated content is out of date by periodically comparing a short summary of its replicated content with the master's summary. If these summaries fail to match, the slave site will proceed by providing a more detailed summary to the master which allows it to determine precisely which directories need to be updated or deleted. For each directory that needs to be updated, the slave summarizes the directory contents in order to determine precisely which files need to be updated or deleted. For each file that needs to be updated, the slave site will download the file completely from the master site using a standard HTTP GET request.

While checking for site synchronization, slaves also effectively monitor the availability of their masters. Should any of its masters go offline, a slave will immediately notify the uServ coordinator. The uServ coordinator also monitors site availability, but it must do so on a much larger scale. The slave's assistance in this task reduces site unavailability due to situations such as improper shutdown of a uServ site or network problems, and is consistent

with our attempt to reduce the centralized roles of the uServ system in order to minimize the cost of providing the service.

4.4 One Caveat: DNS caching

Some DNS servers and most browsers do not properly abide by the time-to-live (TTL) contract for caching DNS mappings. The result is that sometimes a uServ site can become inaccessible for several minutes when a replica of the site is just activated, or the IP address of the site changes. This problem is for the most part a minor nuisance that affects a very small percentage of all accesses to uServ sites. An individual uServ site that is not heavily accessed is unlikely to have its IP address cached within a browser or a local nameserver when it is accessed. Further, users aware of the problem can typically cure it by launching a new browser instance, since indiscriminate caching of DNS entries by the browser is usually the culprit.

Recall that uServ uses a 2 minute TTL, which means that uServ DNS entries should be cached for at most 2 minutes, allowing a replica to become accessible by users very shortly after it is activated by the coordinator. In a perfect world, site inaccessibility can be eliminated completely by implementing a delayed shutdown where a uServ peer node will remain running for 2 minutes after activating a replica. Some DNS server software unfortunately allows configurations that override low TTL values with a global minimum. Most popular browsers ignore TTL values completely and use their own fixed cache timeout settings.

Within our company we have identified a handful of nameservers that appear to be configured to use no less than a 5 minute TTL. Even worse, the Netscape browser caches DNS entries for 15 minutes by default [[Moz](#)]. Internet Explorer appears to use a similar caching policy. Rumor has it these unfortunate caching schemes were implemented within browser software because one cannot get TTL information from the DNS libraries on Windows-based machines. This problem is not one unique to uServ, but also affects systems such as dynamic DNS services. As dynamic IP address assignment and services impacted by dynamic IP address assignment become more common, we expect that operating system libraries, DNS servers and their configuration, and browser implementations will adapt by properly abiding by the DNS protocol.

5. Scalability

The uServ peer nodes perform the most resource intensive tasks in uServ: proxying, storing and replicating content, and serving web content. Since the load of these tasks is distributed across a large number of peer nodes, the system bottlenecks are limited to the centralized dynamic DNS and uServ coordinator components. In our deployment, we have one machine running the DNS component and another running the coordinator. Both machines are Pentium III-class desktop systems running at 400 MHz, with modest amounts of memory (256MBytes) and disk (16GBytes). This configuration easily supports the current user base of over 1800 with little CPU, memory, and secondary storage utilization.

Because of their nearly identical roles, the uServ dynamic DNS component and existing dynamic DNS services on the internet have similar operating costs and scalability characteristics. Dynamic DNS services available today on the internet handle hundreds of thousands of users and charge minimal fees. For example, Dynamic DNS Network Services [DNS] has over 150,000 domain names registered and gets by on donations and advertising revenue alone. Its ISP costs are a mere \$3500 a month, which as of today comes out to 2 cents a month per domain name. This service can be offered cheaply because DNS is a lightweight, low bandwidth protocol for which freely available implementations (including BIND) are highly optimized. DNS also supports redundant servers if needed. To reduce DNS traffic in uServ, the coordinator could in addition be reprogrammed to recognize sites that use static IP addresses and rarely if ever fail over to replicas, and increase their TTL values accordingly.

The uServ coordinator component spends most of its time handling user authentication and site availability monitoring. As we have noted, however, the uServ peer nodes assist in availability monitoring, and the protocol could be extended to further push roles other than authentication to the peer nodes should scalability becomes a problem. Authentication thus becomes the primary bottleneck for the coordinator component. Each authentication requires the exchange of only a small amount of data (the encrypted user ID and password) and a single database lookup. Assuming very conservatively that our system can handle 100 authentications per second and that each uServ site authenticates on average twice daily, the capacity of the coordinator would be over 4 million uServ sites.

6. Security Issues

Security is one of the primary concerns of uServ users, and this concern has been heightened by recent worm attacks on Microsoft's IIS web server software (e.g. Code Red and its variants). In addition to worms, our users are worried about hackers who might exploit holes to install unauthorized programs on their machine, or access files that were not designated for sharing. Other specific areas of concern include denial of service attacks and restricting access of certain content to designated users.

6.1 Malicious Attacks

uServ is written in Java which makes it robust (if not immune) to buffer overflow attacks such as those used by Code Red and other hacking tools to install unauthorized programs. In addition, because of its content sharing focus, the uServ webserver does not allow executing scripts (e.g. CGIs) -- another common source of security holes.

Because the webserver within each uServ peer node is quite simple, there are only a few code paths that need to be thoroughly scrutinized in order to improve security. Our implementation provides only one code path through which all served content, for whatever purpose, is delivered to the network. This code path always explicitly verifies that any delivered content resides within the designated shared folder hierarchy.

The uServ system as a whole is more robust to denial of service attacks than a typical web hosting service. Because

of its distributed nature, a denial of service attack must target multiple machines in order to take out a significant fraction of the system's content. While it is conceivable that uServ's DNS and coordinator components could be targeted, DNS is somewhat resilient to such attacks since IP addresses are cached in local nameservers, and the coordinator being unavailable does not affect existing sites, only sites which need to become activated. An individual uServ site with no replica is likely to be more prone than a hosted site to denial of service attacks since end-user machines typically have rather limited bandwidth and compute power compared to those used by hosting services. Given a replica, though, the uServ coordinator or one of the peer node's slaves will likely lose contact with the site being attacked and trigger the replica to become active. The attack would thus have to keep track of DNS updates and target multiple machines in order to succeed.

6.2 Access Control

The only access control function currently offered by uServ is a (non replicated) private folder that is protected by the ID and password used by the peer node during login. Implementing more sophisticated access control in a peer-to-peer web hosted model such as uServ is non-trivial and remains an area of future work. Below we describe the difficulties and some potential solutions to the problem.

A secure access mechanism must deliver only encrypted data, and also authenticate users to sites and vice versa. Web protocols seamlessly allow browsers to authenticate websites to users and encrypt transmitted data through secure HTTP extensions and third-party issued security certificates [R99]. Unfortunately, web protocols provide no uniform method for websites to authenticate their users [R99]. HTTP does offer a simple mechanism for having the browser prompt the user for an id and password when requesting secured content [Fr+99] Passwords can also be requested through HTTP forms which then set an authenticating cookie. Most websites thus implement their own user authentication scheme which typically requires the user to register a user ID and password with the site.

It would be unwieldy and unreasonable for a user to register with each uServ site requiring secure access. The alternative is a single sign-on scheme, in which case the peer nodes cannot be responsible for authenticating users through passwords; if the sites themselves are responsible for authenticating users via any "uServ global" ID and password, then a malicious peer node could record the passwords presented to it, allowing it to impersonate any user that accesses its secured content.

One interesting solution avenue may be Microsoft Passport [M01] -- a single-sign-on scheme for the web in which a central site accepts passwords in order to authenticate users on behalf of its member sites. In this system, content on a member site requiring secure access forces a redirect to the Passport site, where the user must provide his or her login ID and password. The Passport system then redirects the user back to the member site with the user's authenticated identity encrypted in the redirected request. The member site never receives the user's password. Instead it decrypts the authenticating information provided by Passport in order to reveal the user's identity. Encrypting and decrypting of this user information within the redirect is performed via a symmetric key that has been previously established between Passport and the member site.

uServ makes publishing content on the web as easy and universal as accessing it. Because uServ exploits existing web protocols, uServ content can be accessed with any standard web browser without installing special software. Additionally, by relying primarily on existing desktop infrastructure, the uServ service can be provided at an extremely low cost. Our internal deployment utilizes two low-end Intel Pentium based systems. Currently handling over 1800 users, the system is projected to scale to at least tens of thousands more. The use and growing user base of our deployment lends credence to our thesis that its low cost, wide accessibility, and high availability make uServ a superior alternative to paid hosting services and other content sharing networks for a wide class of users.

Acknowledgements

We thank Andreas Dieberger and Christopher Campbell for many helpful discussions, Yirong Xu and Daniel Meredith for their code contributions, Glenn Deen for advise on corporate guidelines, Eytayo Akins and Jim Spohrer for help with formulating a business model, and the users of our internal uServ deployment for their valuable feedback.

References

- [A+01] Agrawal, R.; Bayardo, R.; Gruhl, D.; and Papadimitriou, S. Vinci: A Service-Oriented Architecture for Rapid Development of Web Applications. In *Proc. of the Tenth International World Wide Web Conference (WWW10)*, <http://www10.org/cdrom/papers/506/index.html> , 2001.
- [AL01] Albitz, P. and Liu, C., *DNS and BIND*, 4th edition. O'Reilly & Associates, 2001.
- [Ap] Apache HTTP Server Project home page. <http://httpd.apache.org/> .
- [A00] Apache Project, Xerces Java Parser. <http://xml.apache.org/xerces-j/> ,2000.
- [Bol+00] Bolosky, W. J.; Douceur, J. R.; Ely, D.; and Theimer, M., Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. In *Proc. of the Int'l Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS-2000)*, pp. 34-43, 2000.
- [Box+00] Box, D.; Ehnebuske, D.; Kakivaya, G.; Layman, A.; Mendelsohn, N.; Nielsen, H. F.; Thatte, S.; and Winder, D., *Simple Object Access Protocol*. <http://www.w3.org/TR/SOAP/> , May 2000.
- [Br+00] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; and Maler, E. (eds.), *Extensible Markup Language (XML) 1.0 (Second Edition)*, <http://www.w3.org/TR/REC-xml> , Oct. 2000.
- [DNS] Dynamic DNS Network Services home page, <http://www.dyndns.org/> .
- [Fi+99] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Mastiner, L.; Leach, P.; and Berners-Lee, T., *Hypertext Transfer Protocol -- HTTP/1.1 - Draft Standard RFC 2616*. <http://www.ietf.org/rfc/rfc2616.txt> , June 1999.
- [Fr+99] Franks, J.; Hallam-Baker, P.; Hostetler, J.; Lawrence, S.; Leach, P.; Luotonen, A.; and Stewart, L., *HTTP Authentication: Basic and Digest Access Authentication - Draft Standard RFC 2617*. <http://www1.ics.uci.edu/pub/ietf/http/rfc2617.txt> , June 1999.
- [Gnut] Gnutella Development home page. <http://gnutella.wego.com/> .

- [Gr01] Groove Networks, Inc., *Groove Product Backgrounder*. Corporate whitepaper, 2001.
- [ICQ] ICQ home page. <http://web.icq.com> .
- [Lot] Lotus Notes. <http://www.notes.net/> .
- [Moz] Mozilla.org, netlib documentation. <http://www.mozilla.org/docs/netlib/> (netlib preferences section).
- [M01] Microsoft Corp. *Microsoft Passport Technical Whitepaper*.
<http://www.passport.com/downloads/TechnicalWhitePaper.doc> , March 2001.
- [M87] Mockapetris, P., *Domain Names - Implementation and Specification -- STD 13, RCF 1035*.
<http://www.ietf.org/rfc/rfc1035.txt> , November 1987.
- [Nap] Napster home page. <http://www.napster.com/> .
- [Per] Permio Technologies, Inc., SOCKS home page. <http://www.socks.nec.com/> .
- [R99] Rescorla, E., *HTTP Over TLS*, IETF Internet Draft,
<http://www1.ics.uci.edu/pub/ietf/http/draft-ietf-tls-https-03.txt> , Sept. 1999.
- [Safe] SafeWeb, Inc. *Triangle Boy Network* (Technical Whitepaper).
http://fugu.safeweb.com/sjws/solutions/white_papers_triangle_boy.html , 2001.
- [Sen01] Senie, D., *NAT Friendly Application Design Guidelines*. IETF NAT Working Group Internet Draft, <http://www.ietf.org/internet-drafts/draft-ietf-nat-app-guide-06.txt> , 2001.
- [Sun] Sun Microsystems, Inc., *Project JXTA: An Open, Innovative Collaboration*.
<http://www.jxta.org/project/www/docs/OpenInnovative.pdf> , April 25, 2001.
- [WG99] Whitehead, J. and Goland, Y. Y., *WebDAV: A Network Protocol for Remote Collaborative Authoring on the Web*. In *Proc. of the European Computer Supported Cooperative Work Conference (ECSCW'99)*, 1999. Available at <http://www.webdav.org> .
- [W01] Working Resources, Inc., *BadBlue website and product description*. <http://www.badblue.com> , 2001.
- [X01] XDegrees, Inc., *Introducing XDegrees Technology*. Corporate whitepaper, 2001.
- [YGM01] Yang, B. and Garcia-Molina, H., *Comparing Hybrid Peer-to-Peer Systems*. In *Proc. of the 27th Int'l Conf. on Very Large Data Bases (VLDB-2001)*, 2001.