A Secure Peer-to-Peer Web Framework

Joakim Koskela
Helsinki Institute for Information Technology
PO Box 19800, 00076 Aalto
Email: joakim.koskela@hiit.fi

Andrei Gurtov
Helsinki Institute for Information Technology
PO Box 19800, 00076 Aalto
Email: andrei.gurtov@hiit.fi

***Abstract*—We present the design and evaluation of a secure peer-to-peer HTTP middleware framework that enables a multitude of web applications without relying on service providers. The framework is designed to be deployed in existing network environments, allowing ordinary users to create private services without investing in network infrastructure. Compared to previous work, scalability, NAT/firewall traversal and peer mobility is achieved without the need for maintaining dedicated servers by utilizing new network protocols and re-using existing network resources.**

I. Introduction

Peer-to-peer (P2P) systems have been popular within network research during the past years as they have the potential to offer more reliable, fault-tolerant and cost-efficient networking. P2P systems do not suffer from the same deployment complications and performance bottlenecks that systems relying on dedicated centralized servers often do. The research around P2P systems has scrutinized topics ranging from architectures and routing algorithms to use-cases and security issues. As P2P networks have matured we see more applications leverage the technology, and currently there are a myriad of applications which (at least partially) are based on P2P models.

P2P web applications, or more generally a P2P adaptation of the HyperText Transfer Protocol (HTTP), is an area that has received a fair amount of interest. As HTTP is one of the most widely used protocols, creating a generic P2P HTTP architecture could enable a vast amount of existing services to benefit from the advantages of P2P networking.

However, current initiatives focus only on specific HTTP-based applications, replacing certain functionality with a P2P overlay. For instance, current proposals on using P2P networks for HTTP-based web services are based on migrating the centralized service discovery to a peer overlay while the service requests follow a traditional client-server model. Furthermore, these proposals often require dedicated servers or public peers to bootstrap the system, and do not consider deployment on a wider scale. Security and middlebox traversal are commonly overlooked as well.

In this paper we present the design and implementation of a secure P2P middleware architecture for HTTP-based communication focused on deployability in the current global, and mobile, Internet. The model is evaluated through performance measurements in network environments typical for residential users.

Compared to previous work and commercial solutions, we present a generic model suitable for most HTTP-based application, that can be deployed without investing in dedicated infrastructure while addressing issues such as middlebox traversal, mobility, security and identity management.

II. Peer-to-peer HTTP

From its launch in the early 1990s, the HyperText Transfer Protocol (HTTP) had grown to be one of the most popular protocols on the Internet today. It is used daily for everything from past-time activities, such as recreational browsing, gaming and media downloads, to business- and security-critical applications such as payment systems and on-line banking.

The success of HTTP has clearly grown beyond its original design as a simple, easy to manage protocol for exchanging markup-based content. Today it is being chosen for a wide range of applications, working in different environments and in different models. The slight overhead of the text-based protocol is clearly over weighted by the benefits of its wide acceptance and the availability of tools and ease of development.

HTTP follows a client-server model where the connections are established from a client to a public server, often located using the Domain Name System (DNS). This limits the use of the protocol to service providers which have a public, unblocked, access to the network and a fairly static Internet Protocol (IP) address. This is not the case for most of the devices connected to the Internet today.

The surge of wireless Internet access through mobile devices has created a wide base of potential micro-service providers. However, these hosts usually reside behind network address translators (NATs) or firewalls blocking direct connections. Furthermore, they might change network location at times, and be accessible (on-line) for only short periods at a time. Clearly, there are a number of obstacles to overcome.

Considering the huge amount of HTTP-based applications present today, a generic P2P HTTP architecture could enable a range of new uses for applications on the Internet with minimal or no additional effort. The applications would not be tied to the limitations of dedicated servers, but could tap into the collective resources of the end-users, greatly expanding the diversity of the available information and services. Personal web pages and social sites would become truly personal, reflecting the status of the user more precisely. Content sharing and collaboration would be done on the users' terms, without the privacy concerns, censorship or other interference imposed by a service provider.

978-1-4244-7265-9/10/$26.00 ©2010 IEEE

III. Related work

P2P use of HTTP-based protocols is not a novel concept. However, existing proposals often target only a specific use-case or application, or have not been design to be deployed in large networks.

P2P web services have received a fair amount of interest through a number of proposals [1] [2] [3]. These proposals depict architectures where end-users, the peers, act both as the consumers as well as the providers of services, using a distributed lookup mechanism for rendezvous. Although in line with our goals, these proposals concentrate on distributing the lookup, ignoring deployment details such as network obstacles (middleboxes) and security issues.

Nokia's Research Center has developed a personal mobile web server for Symbian Series60-based smartphones [4]. This Apache HTTPD based software can serve both static content and dynamic, context-dependent, pages written in PHP or Python. These mobile web pages are thus able to serve as a personal information center integrated with the core mobile applications. For instance, information from the phone's calendar can be utilized as well as any photographs or short messages present on the device. Compared to the P2P web service proposals, connectivity has been carefully addressed, as it presents one of the greatest challenges in cellular environments. For reachability the system relies on a public server for relaying the data traffic.

Opera Unite [5] offers a similar P2P web experience for desktop computers. Instead of a dedicated server application, it is based on JavaScript applications that are run within the Opera web browser itself. As with the Nokia mobile web server, it uses a centralized model for authentication and rendezvous, and does not provide an interface for external applications.

IV. Solution model

As ordinary end-users or organizations may not have the resources or know-how to deploy supporting infrastructure, our system was designed to re-use existing network services (referred to as *overlay infrastructures*) for lookup when establishing connections. In contrast to many existing P2P protocols, we do not use these overlays for exchanging application data, instead we establish straight connections using the middlebox-traversal capabilities of recent network protocols.

The system is designed to be deployed as a local daemon acting as a proxy for HTTP traffic. An overview of the system is presented in Figure 1.

A. Identity management

The P2P system uses a strong, public key-based identity scheme. Peers are referred to using email-like identifiers (e.g., *alice@example.com*), which are tied to the public half of a key pair. This key pair is used for authentication, and to electronically sign and encrypt the data packets stored in the P2P overlays.

Since users only deal with the identifiers, the *names*, of the peers, we need mechanisms for securely binding the identifiers

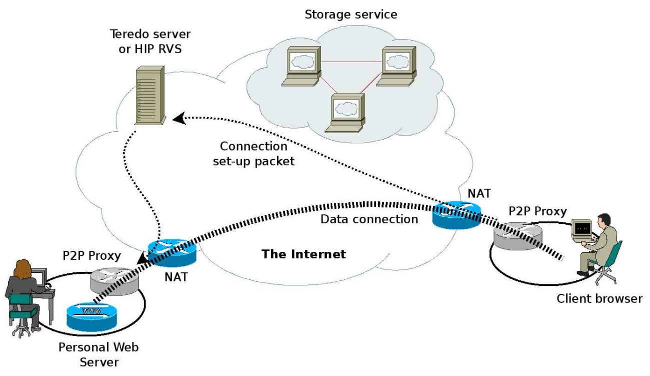

Fig. 1. System overview. Storage services are used for lookup. Teredo or HIP RVS for forwarding the initial connection packet, further data go straight between the hosts.

to the public keys to prevent impersonation. This is done either manually, through leap-of-faith (at the first encounter) or using certificates issued by a trusted identity authority.

The first two mechanisms do not require interaction with a third party, and are suitable even for ad-hoc or closed networks. Using a trusted identity authority might seem to break the P2P model, however, as these certificates are issued beforehand, the authorities do not need to be accessible during authentication.

Each peer maintains a database of the root certificates of trusted identity authorities. Compared to public-key infrastructures, new identity authorities can be created at any time, for any reason, and peers choose which ones to trust. For instance, enterprises can maintain their own authority to issue identities used in company-internal communication. At the same time, a peer could also trust certificates issued by an authority created for a close group of friends. Compared to previous proposals, these identity authorities do not require a dedicated server, but can be created and distributed *offline*.

B. Connectivity

As we do not rely on overlay routing, we need efficient methods for traversing network obstacles that prevent direct connections. Our focus has been on using existing, stable, and widely available solutions; namely Teredo [6] and the Host Identity Protocol (HIP) [7].

1) Teredo: Teredo is an IPv6 over IPv4 tunneling protocol designed to traverse NATs and other network middleboxes through the use of UDP encapsulation and *hole punching* techniques [8]. Teredo uses a public server, *Teredo server*, for initializing the connection between the *Teredo clients*, removing the need for each client to have a public IP address.

The Teredo protocol is designed to be simple and light, both in packet overhead and load. The server load is kept minimal by only forwarding the initial packet and encoding state into the Teredo IPv6 addresses. This has contributed to its popularity, and Teredo is currently available for a wide range of operating systems (including built-in support in Microsoft Windows), with a number of organizations maintaining open, free, Teredo servers.

Teredo does have a few drawbacks. The NAT traversal method used is simple, and does not penetrate all types of NATs.

2) The Host Identity Protocol: The Host Identity Protocol (HIP) is a communication architecture that splits the notion of locator and identifier through the use of cryptographically generated identifiers. End-points are addressed using an IPv6 address constructed from the hash of their public key-based Host Identity (HI). These non-routable IPv6 addresses, Host Identity Tags (HITs), are translated into a routable address by the HIP stack, adding a layer between the transport and network layer.

Although not the primary purpose of HIP, an extension has been developed for NAT and firewall traversal. This is based on UDP encapsulation and the use of the Interactive Connection Establishment (ICE) [9] to find an optimal route between the hosts. HIP can also use rendezvous servers (RVS) and relays to assist in connection establishment and NAT traversal.

There are currently a number of HIP implementations available for Microsoft Windows, Mac OS, Linux and FreeBSD. Furthermore, organizations such as HIIT maintain free RVS servers for public use.

C. Lookup

Connections are established using the information found in *registration packets* stored in the overlay infrastructures. These packets are published by the users, under a key matching the hash value of the user name. Depending on the connectivity methods in use, these contain IP addresses, addresses of rendezvous servers and relays or other protocol-specific information.

As these packages are signed, the system is able to use any type of storage that offers a hash table-like indexing. These are referred to as overlay infrastructures, and are ideally (as the name suggests) fault-tolerant overlay networks such as OpenDHT. But our concern is not how the storage is hosted, but that it offers the required service. As we are able to use multiple of these simultaneously, we can use redundancy to create a sufficiently reliable system, even if the individual storage services are unreliable or untrusted

This is one of the differences between our model and many other P2P systems. We do not stress on building an optimal overlay network for sharing the registration packets, but are able to use a number of existing services for it, even simple web-sites. This eliminates the need to deploy custom servers to bootstrap the system, as the required components are already in place.

Prime examples of these open, free, storage infrastructures are different cloud-computing services such as Google's AppEngine. It allows users deploy either Python or Java-based web applications, and provides a simple storage API with database features. Using the Google AppEngine, a suitable storage back-end can be created with a few tens of lines of code. Other possible infrastructures include Amazon's SimpleDB service, which offers similar quota-limited services for free. Also, existing P2P networks, such as Gnutella or various BitTorrent trackers could be used to distribute the registration packets.

D. Application interface

Applications use the framework by *registering* HTTP-based services at specific port numbers for the user's *identity*. For instance, an RSS feed generator run by the user Alice could register itself at port 1000 for her identity, alice@example.com. This feed is accessed by subscribing to the address alice@example.com, port 1000, while using the framework as an HTTP proxy.

The P2P system offers applications two ways for interaction; as a standard HTTP proxy, or through URL-rewriting. The HTTP proxy interface results in shorter, more natural, URLs, but requires that the software supports the use of HTTP proxies.

Using the URL-rewirting interface, applications issue requests to the P2P proxy, indicating the target user and service port as the first two path components. For instance, a request to *http://localhost:5050/alice@example.com/1000/get_rss* would be forwarded as a request for */get_rss* to the service port 1000 of alice@example.com.

The HTTP proxy interface is much simpler. The target identity is specified as the host name, but as the character `@` has special meaning (indicating a login name for authentication), it is escaped with the sequence `.at.`. For instance, the URL *http://alice.at.example.com:1000/get_rss* would result in a request to the RSS service in our example.

V. Evaluation

The feasibility of the model is evaluated by measuring the performance of the prototype implementation in a network environment common for residential users. The goal was to determine whether the technologies we outlined can deliver a satisfactory user experience in the intended environments.

Three connectivity methods were evaluated. First using HIP for both NAT traversal and security. Secondly using Teredo only for NAT traversal. Finally we evaluated a combination of Teredo and HIP. This configuration uses HIP for security and mobility, but Teredo for NAT traversal. Although the native NAT traversal of HIP is more efficient, it may require RVS servers that are not as widely deployed or lightweight as Teredo.

A. Test set-up

The evaluation was performed using the Linux-based prototype on a 2.2 GHz Intel Core 2 Duo desktop computer with 2 GB of RAM as client, and a 1.4 GHz Intel Core 2 Duo laptop with 4 GB of RAM as the P2P HTTP server. These were installed with the Debian GNU/Linux 4.0 operating system using a standard 2.6.28 version of the Linux kernel. The HIP for Linux (HIPL) version 1.0.4 stack was used for HIP and Miredo 1.1.5 for Teredo connectivity. The implementation was based on the HIP-based P2PSIP system presented in [10]. It was extended to support P2P HTTP connections through

TABLE I
ROUND TRIP TIME (RTT) BETWEEN NETWORK ELEMENTS.

Source host	Target	RTT
Serving	Client	195ms
Serving	Lookup	38ms
Serving	Teredo server	187ms
Serving	HIP RVS	194ms
Client	Lookup	27ms
Client	Teredo server	50ms
Client	HIP RVS	1ms

an HTTP forwarding subsystem (offering both the URL and proxy-based interface) and Teredo support.

We used for lookup a simple Google AppEngine storage application consisting roughly of 20 lines of code. This application offers an HTTP interface for storing and retrieving arbitrary blocks of data using 160 bit keys. Similar to the interface offered by OpenDHT, each item has an expiration date and can be protected from unauthorized removal by a password. The default server for Miredo, teredo.remlab.net was used for Teredo, and HIIT's public RVS, ashenvale.infrahip.net, as the HIP RVS server.

The evaluation was performed with the serving host behind a NAT on a residential digital subscriber line (DSL). The client host was placed 16 network hops away, on a different autonomous system. This provided a realistic scenario for what can be expected in terms of network quality for potential P2P HTTP users. The average round-trip times (RTT) between the network elements is presented in Table I to illustrate the quality of the connections.

B. Connection delay

As a result of the required NAT traversal, the initial connection establishment will be more time consuming than any subsequent requests. Figure 2 show the average times recorded over ten connections for each of the configurations. The variance in the measurements were relatively low, and common tasks (lookup and generation of the HTTP response) took an equivalent amount of time for all configurations.

Teredo performs as expected, adding only the RTTs needed to involve the Teredo server to the connection process which results in the average of 1281 ms total. The HIP-based connections showed worse performance. The HIP connection establishment seems to add an additional 3000 ms to the process, with the total averaging 4026 ms. Only a small fraction (100 ms in *processing*) can be explained by additional chores related to HIP (mapping HITs to IP addresses). As this delay is present both when relaying through the RVS and Teredo server, the HIP NAT traversal cannot be blamed either.

We found the problem to be due to the 2.6.28 version of the Linux kernel dropping the first packet of the IPSec ESP BEET tunnels created by HIPL. HIPL uses by default the kernel-based IPSec, which results in the HIPL daemon being activated to perform the BEX when a packet destined for a new HIT is received. The first packet is subsequently dropped while waiting for the BEX to complete. As the system uses TCP for the P2P proxy connections, the connection is completed only after the TCP retransmission timer expires, resulting in the delay.

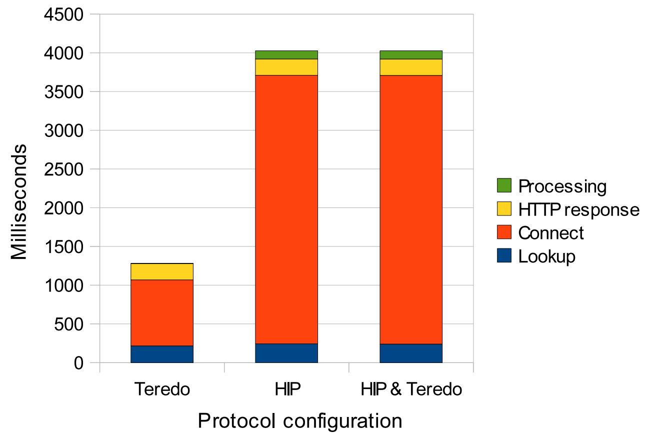

Fig. 2. Connection delays

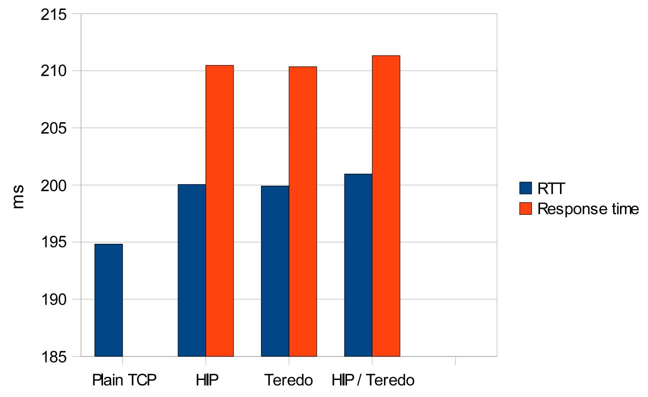

Fig. 3. Average RTT and HTTP response times for each connection type. Plain TCP HTTP connections were not possible as the serving host was behind a NAT.

Although this issue may be addressed in future versions of the HIPL implementation, it should be noted that this affects only the initial connection set-up, not subsequent requests.

C. Latency

The latency was measured using pre-established connections, which provides a view of what the end-user would experience after the initial request. Figure 3 shows the average RTT measured (using ICMP ping packets) for each connection type, and corresponding response times for the actual HTTP content.

As expected, the figures are very similar between the different connection techniques. After the initial set-up, each protocol sends the packets directly between the hosts, adding only different types of encapsulation. The slightly longer times for the HIP-based connections can be explained by the additional encryption.

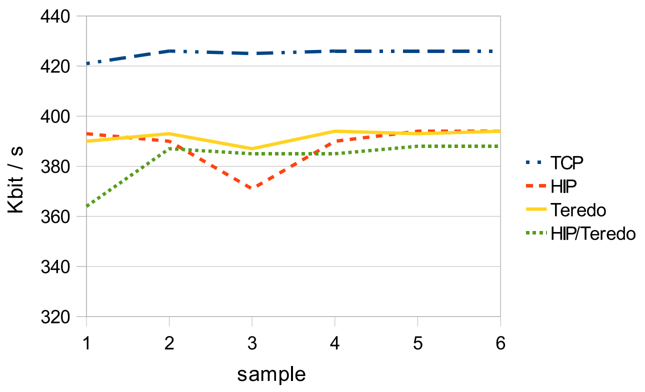

Fig. 4. Measured throughput when using different connection types.

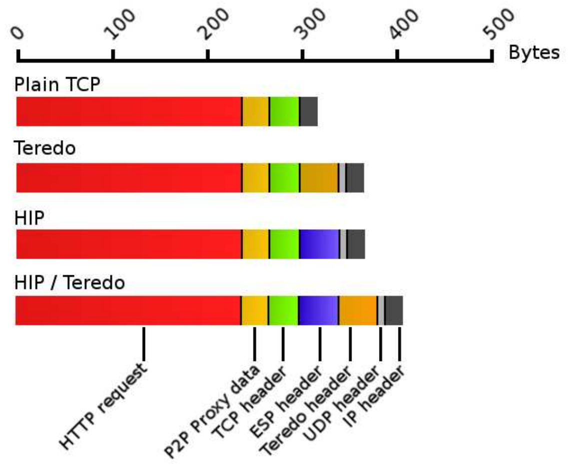

Fig. 5. Comparison and content of a packet containing 238-bytes of application data produced by the different protocols.

The HTTP response time is close to the theoretical RTT. This is approximately half of the time required by a traditional HTTP request, which needs two RTTs to complete the three-way handshake of TCP.

D. Throughput

The throughput was measured as the maximum rate at which the serving host can deliver data over a TCP connection. The measurements were done using the iperf network performance measurement tool. Figure 4 shows the results when using the three different connection techniques as well as without any encapsulation (plain TCP) as reference.

It should be noted, as seen on the graphs, that these are highly susceptible to external influences (other users, link quality and ISP policies), but provide still a glimpse of the affect of these protocols on the performance.

As the figure shows, the encapsulation does have a noticeable affect on the overall bandwidth. Both HIP and Teredo decreased the maximum bandwidth from 426 Kbit/second to 394 Kbit/second, a decrease of 8%. Perhaps surprisingly, the HIP / Teredo combination resulted only in an additional 1.5% decrease from that value, with a maximum bandwidth of 388 Kbit / second.

E. Packet overhead

Figure 5 compares the total IP packet size generated by a 238-Byte HTTP request using the different connection types. As the figure shows, Teredo and HIP add approximately an equal amount of overhead to the data packets (40 and 42 bytes in this example, although the ESP header length may vary due to padding). This corresponds quite well to the measured throughput. It should be noted that although the Teredo / HIP combination adds the most overhead, it is only 42 bytes more per packet. Although this is roughly a 33% increase in the total amount of overhead compared to plain Teredo, it is below three percent of a typical MTU of 1500 bytes.

F. NAT traversal

Estimating the success rate of NAT traversal is difficult, as there exists dozens, if not hundreds, of different manufacturers using different NAT policies even within a single product line.

The results presented in [11] provide one estimate of the types of NATs deployed in the Internet in 2005. Although these figures are based on observations of limited set, the fact that 82% of those did support the UDP “hole punching” technique used by both Teredo and HIP provides an indication that most NATs would be traversable using these protocols. Also, HIP does specify an extension for using triangular routing through HIP Relays [9], in case both end-points are behind untraversable NATs.

G. Load on storage

During our evaluation, we observed that storing a registration packet generated roughly 2 KB of upstream and under 200 bytes of downstream data in total. A single lookup resulted in approximately the same amount of data, although in reverse directions. Storing a packet in the AppEngine database was estimated to consume roughly 2 KB of storage space.

Considering a system of 2000 users updating their status once per ten minutes (as was the default in our prototype), maintaining the system would generate approximately 550 MB of upstream and 55 MB of downstream traffic in 24 hours, with 4 MB stored in the database at any time (assuming old packets are discarded). Assuming that each user would contact, on average, five new peers per hour it would add another 46 MB of upstream and 460 MB of downstream traffic. This interaction would total in slightly under 600 MB upstream and 510 MB downstream traffic per 24 hours, with 4 MB of data stored at any given time.

Currently the Google AppEngine limits its free service per application to 1 GB of data transfers in either direction per 24 hours, and 1 GB of storage. This is well within our theoretical limits for a 2000 user population. Considering that most users are not constantly active, we could support much larger groups as well with only a single application. For instance, assuming that 70% of the users log in each day, remaining active on average for 4 hours per day, we could support 17000 users with the same network load.

To validate our assumptions, we experimented with a small-scale simulation of this storage use. We created an application mimicking the requests made by the client to the storage system, and deployed it on 450 nodes on the PlanetLab network. We ran the application for five days using as storage a single Google AppEngine instance. Each one of the PlanetLab nodes made a registration request using actual registration packages (and unique identifiers) once per 10 minutes, and requested a package five times per hour.

Before each reset of the Google AppEngine quota counters we recorded the amount used during that 24 hour period. The results were in line with our assumptions. The storage quota never exceeded noticeable amounts, and bandwidth use was only 8% of the downstream and 14% of the upstream traffic quota.

H. Usability and overall impression

As the work we have done has been to combine and explore technologies that enable ordinary users new ways to communicate, the *ease* of use of any solution is paramount. We need to consider issues such as ease of installment, the complexity of configuring it and what would be the driving force to adapt the system.

Through our experiments with the prototype, we have recognized that these are problems not easy to overcome. The installation of the software has, even though currently a mere prototype, been made easy through an graphical user interface. Most modern Linux distributions today support the needed protocols (such as IPv6 and IPSec) out of the box, and have well-developed software package management systems. The situation is different on other platforms (Microsoft Windows and Mac OS), but as noted, current Windows versions have native support for Teredo, and with the help of software such as OpenHIP, HIP support is possible.

The ease of use requires work. During our evaluation, we tested two use-cases; file sharing and personal photocasting. The KDE personal file server (Kpf), a small web server of the K desktop environment, was configured to serve content over the P2P system, accessed by desktop browsers. For the second use-case we implemented a simple utility that creates a pod- or photocast RSS stream from the contents of a media folder, which was accessed by a standard podcast client.

The client applications were in general quite easy to configure. Desktop browsers can easily be set to use the HTTP proxy interface of the prototype, resulting in intuitive and memorable URLs for the user (e.g., *http://alice.at. p2pship.org/media*). The podcast client lacked support for HTTP proxies, but could use the URL-based interface (although this resulted in a longer URL).

Configuring the software of the service provider was harder. According to the design, these need to be *registered* with the P2P proxy in order to receive requests. As the podcasting application was made specifically for the evaluation, it was easy to integrate this registration. But for existing applications, such as Kpf, this needs to be done either manually or by a separate application.

Making this integration easy is something still to be solved. We imagine plug-ins could be constructed for applications that support such extensions. The P2P proxy itself could also feature an extensions scheme allowing users to construct and easily distribute add-ons which link specific applications to the system. Following Opera Unite's model, we could consider integrating a scripting engine and support applications specifically built to run in this P2P environment.

VI. Conclusion

Although our prototype displayed good performance and provided the type of services we were looking for, finding applications that will attract users is a challenge. As seen with Opera Unite and the Nokia Mobile Web Server, the concept has been recognized by others as well, but it is still lacking a clear direction.

The main message of this paper is that the technology needed for creating private P2P overlays is available today for ordinary end-users, it is only lacking the proper packaging. Although a killer application for P2P web applications would surely result in a wide roll-out of similar systems from commercial vendors, we argue that these service providers are not needed as the necessary technology is already in place. The existing infrastructure of today enables us to create new networks without being dependent on a single vendor. In this paper, we have concentrated on networks built on the HTTP protocol, but the model can be extended to other protocols as well.

References

[1] F. Banaei-kashani, C. chien Chen, and C. Shahabi, "Wspds: Web services peer-to-peer discovery service," in *In Proceedings of the International Conference on Internet Computing*, 2004, pp. 733–743.

[2] G. Gehlen and L. Pham, "Mobile web services for peer-to-peer applications," in *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, 2005, pp. 427 – 433.

[3] C. Schmidt and M. Parashar, "A peer-to-peer approach to web service discovery," *World Wide Web*, vol. 7, no. 2, pp. 211–229, 2004.

[4] Mobile Web Server, http://betalabs.nokia.com/betas/view/mobile-web-server.

[5] Opera Unite, http://unite.opera.com/.

[6] C. Huitema, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)," RFC 4380 (Proposed Standard), Internet Engineering Task Force, Feb. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4380.txt

[7] R. Moskowitz and P. Nikander, "Host Identity Protocol (HIP) Architecture," RFC 4423 (Informational), May 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4423.txt

[8] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2005, pp. 13–13.

[9] M. Komu, T. Henderson, P. Matthews, H. Tschofenig, and A. Kernen, "Basic HIP Extensions for Traversal of Network Address Translators," Oct. 2008, work in progress. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-hip-nat-traversal-05.txt

[10] J. Koskela, "A HIP-based peer-to-peer communication system." in *ICT2008: Proceedings of the 15th International Conference on Telecommunications*, June 2008, pp. 1–7.

[11] S. Guha and P. Francis, "Characterization and measurement of tcp traversal through nats and firewalls," in *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. Berkeley, CA, USA: USENIX Association, 2005, pp. 18–18.