z2z: Discovering Zeroconf Services Beyond Local Link

Jae Woo Lee*, Henning Schulzrinne*, Wolfgang Kellerer† and Zoran Despotovic†
* *Department of Computer Science, Columbia University, New York, USA*
{jae,hgs}@cs.columbia.edu
† *DoCoMo Communications Laboratories Europe, Munich, Germany*
{kellerer,despotovic}@docomolab-euro.com

***Abstract*—The Zeroconf technology, better known as Apple Bonjour, is one of the most prominent solutions for service discovery in local area networks. Zeroconf uses multicast to attain its goal of eliminating configurations in service discovery. The multicast-based design, however, makes it difficult for Zeroconf services to reach beyond the local link. This makes the technology unsuitable for certain discovery scenarios which would otherwise be good candidates.**

This paper presents the Zeroconf-to-Zeroconf Toolkit (z2z), our first attempt to realize a hybrid architecture that combines the simplicity of Zeroconf with the scalability of DHT-based peer-to-peer networks. Our z2z connects multiple Zeroconf subnets using OpenDHT. By doing so, it extends the reach of existing Zeroconf-enabled applications beyond the local link. Furthermore, it provides a framework on which to build a global service discovery solution based on Zeroconf.

***Index Terms*—Zeroconf, service discovery, Bonjour, DHT, OpenDHT, iTunes, z2z.**

I. Introduction

Zero Configuration Networking (Zeroconf) [1] solves the following problem: when multiple IP-enabled devices are physically connected with one another, one device should be able to use the services provided by another without requiring the user to configure the devices manually. For example, when a user connects two computers either directly using an Ethernet crossover cable or via an Ethernet switch, he should be able to accomplish his file-transfer task by simply starting up the appropriate applications at both ends. The applications should *discover* each other without the user telling them where to find them.

Today, Zeroconf technology is one of the most widespread solutions for service discovery in local area networks. Bonjour is Apple [2]'s Zeroconf implementation, and it is an integral part of Mac OS X operating system. Bonjour is also installed on a large fraction of the personal computers running Windows operating system, thanks to the popularity of iTunes—Apple's music playing application—which installs Bonjour for Windows as part of its installation process. For UNIX-like platforms, there is a mature open-source implementation of Zeroconf called Avahi [3], which comes preinstalled in a number of major Linux distributions such as Debian [4] and Ubuntu [5]. On the hardware side, virtually every printer sold today supports Zeroconf. The number of Zeroconf-enabled applications is rapidly increasing as well, as evidenced by the growing number of Zeroconf service types registered in [6].

The multicast-based design of Zeroconf, however, effectively limits its usage to the local subnet. This presents no problem for the discovery scenarios that are primarily motivated by hardware devices, such as discovering the printers in a network. But as the focus of Zeroconf is shifting towards more sophisticated services provided by software applications, the limited reach of the services often makes the technology unsuitable for many discovery scenarios that would otherwise be perfect candidates for Zeroconf. For example, a number of chat applications (such as Apple's iChat) use Zeroconf to discover other users in the local link and display them in their *ad hoc* buddies window. A straightforward extension of this mechanism is to discover those people who have convened for the same purpose even if their computers are not in the same local network, such as a group of people attending an academic conference scattered in a number of adjacent buildings, or using a mixture of wired and wireless networks which are usually separate subnets. As another example, iTunes lets a group of officemates share their music. It would be nice to include the coworkers working at home or at a remote satellite location.

In this paper, we present an approach to extend the reach of Zeroconf service discovery, inspired by the recent innovation in peer-to-peer network research. Structured peer-to-peer overlay networks based on distributed hash tables (DHT) became popular as the substrates on which global-scale distributed systems are built. A DHT network is characterized by an efficient algorithm to map an arbitrary string to a particular node in the network and to produce a routing path of a bounded number of hops from any node to that node. The mapping is deterministic and results in a uniform distribution (or other desired distributions for some algorithms) of the strings among the participating nodes. This enables efficient implementations of a number of global-scale services such as file sharing and overlay multicast. Our approach is to connect multiple Zeroconf subnets using a DHT network. We have designed and implemented the Zeroconf-to-Zeroconf Toolkit (z2z) that connects Zeroconf subnets using OpenDHT, a publicly accessible DHT service. A z2z process running in a subnet *exports* locally available Zeroconf services into OpenDHT. Another z2z process running in a different subnet can then look up

the services in OpenDHT, and *import* them into its own local network as if they had originated locally. Such imported services are indistinguishable from the real local services in the eyes of the applications, and thus the imported services simply show up along with other locally available services in the existing, unmodified Zeroconf-enabled applications.

Our contributions are twofold. First, we propose a hybrid architecture that combines the ease of Zeroconf with the scalability of DHT-based peer-to-peer networks. Second, we developed a practical tool that can extend the reach of any existing Zeroconf-enabled application without modification. The modular software design also makes it a suitable framework on which to build a global service discovery system based on Zeroconf.

The remainder of the paper is organized as follows. Section II starts with background information on Zeroconf and OpenDHT, and ends with an architecture overview of z2z. Section III describes the usage of the z2z command line executable, provides a message flow based explanation of how it works, and finally delves into the implementation detail. Section IV lists related work. Lastly, Section V discusses possible future directions of this effort.

II. Background and Approach

A. Zeroconf, mDNS, DNS-SD, and Bonjour

There is some confusion about what exactly the term Zeroconf means. The term came from the IETF Zero Configuration Networking Working Group [7], which was chartered to develop a requirements specification for networking in the absence of configuration and administration. The working group identified three requirements for zero configuration networks:

1) IP address assignment without a DHCP server;
2) Host name resolution without a DNS server;
3) Local service discovery without any rendezvous server.

For the first requirement, the working group produced the self-assigned link-local addressing standard (RFC 3927) [8], which is implemented in major operating systems today. The working group never reached a consensus regarding the second and third requirements, and it became inactive without producing any further specification.

Meanwhile, Apple introduced *Bonjour*. Bonjour is the implementation of Multicast DNS (mDNS) [9] and DNS-based Service Discovery (DNS-SD) [10] protocols, which are Apple's proposals for the second and third requirements of Zeroconf. As Bonjour became widespread, the term Zeroconf became synonymous with the abstraction that Bonjour implements, namely the mDNS and DNS-SD protocols. Our use of the term Zeroconf is in this spirit.

The self-assigned link-local addressing described in RFC 3927 establishes the foundation for Zeroconf by ensuring that IP networking is functional as long as the link layer is present. This aspect of Zeroconf is not relevant in our discussion of z2z, however, since we assume that the subnets are connected to the Internet.

The second requirement of Zeroconf is satisfied by mDNS. An mDNS daemon is essentially a DNS server. It uses the same DNS record types and the same packet layout. In fact, an application querying for a DNS record would not be able to tell whether a response came from mDNS or a conventional unicast DNS server. There are, however, a few important differences:

- mDNS is run by *every* host in a local link whereas a conventional DNS system runs on a single server host.
- Queries are sent via multicast to all hosts in the local link using UDP port 5353 instead of 53, the conventional port for DNS.
- All mDNS record names must end in ".local.". The resolution of such names are routed to mDNS by the operating system.

A mDNS daemon provides local host name resolution using A type records. For example,

```
Toms-Computer.local.  A  160.39.243.99
```

DNS-SD, together with mDNS, satisfies the third requirement of Zeroconf. DNS-SD defines the naming conventions for PTR, SRV, and TXT records carried by mDNS daemons. PTR records are used to enumerate the service instances of a particular type. The service instances are mapped to the host names and port numbers using SRV records. TXT records accompany the SRV records in order to provide additional information about the service instances. The following example illustrates this concept:

```
_daap._tcp.local.  PTR
    Tom's Music._daap._tcp.local.
_daap._tcp.local.  PTR
    Joe's Music._daap._tcp.local.

Tom's Music._daap._tcp.local.  SRV
    0 0 3689  Toms-Computer.local.

Tom's Music._daap._tcp.local.  TXT
    "Version=196613" "Password=false"
    "Media Kinds Shared=3"

Toms-Computer.local.  A  160.39.243.99
```

This is a textual representation (edited for clarity) of a few DNS records produced by Apple's iTunes music player application when its music sharing option is enabled. The PTR records are used to enumerate the two service *instances* (Tom's Music and Joe's Music) that are currently available in the local network for the "_daap._tcp" service *type*. The host name and port number for a specific service instance (Tom's Music in this case) is provided by a SRV record. A TXT record with the same name as the SRV record carries additional information about the service instance. Finally, an A record maps the local host name to an IP address.

The mDNS daemons running on each host in a local link collectively store and manage the PTR, SRV, TXT, and A records for the services registered in the local subnet. The

queries and the answers are then exchanged via link-local multicast.

B. OpenDHT

OpenDHT is a publicly accessible DHT service [11], [12]. It consists of 200–300 globally distributed hosts running the Bamboo DHT algorithm [13]. Each host also acts as a client gateway exposing a simple *put* and *get* interface. From a client application's point of view, it is simply a remote storage facility where the client application can *put* or *get* key-value pair data items.

The *put* and *get* operations are performed via XML RPC [14]. This black-box approach greatly simplifies application development because the client applications do not need to integrate DHT access libraries. On the flip side, since OpenDHT does not reveal the nodes in the DHT routing path, it is difficult to implement an application that uses such information, such as an overlay multicast built atop a DHT substrate [15].

We chose OpenDHT for the initial implementation of z2z, mainly because of its ease of use. OpenDHT is sufficient for our current use of DHT, which is limited to storing and retrieving service announcements. Other DHT algorithms and implementations can easily be substituted in the future when OpenDHT no longer satisfies our needs.

Any OpenDHT node can act as a gateway to which a client application sends a put or get request, but for the best performance, a gateway node should be chosen so that it is close to the client host in terms of the network topology. For locating the nearest gateway, OpenDHT uses an overlay anycast service called OASIS [16]. Our z2z uses the OASIS mechanism by default, but it also lets the user specify a particular OpenDHT gateway as a command line option.

C. Architecture Overview of z2z

The basic design of z2z is simple. A z2z process running in a Zeroconf subnet gathers all the service announcements of a particular type (specified by the user) and *exports* them into OpenDHT. Another z2z process running in a different subnet can then *import* those services by *getting* those announcements from OpenDHT and register them in its own subnet as if they had originated locally. Figure 1 depicts such a scenario. Multiple z2z processes can be present in a single subnet as well. Section V discusses this case.

Since each data item in OpenDHT is a key-value pair, z2z associates a key with each service item that it exports into OpenDHT. By default, z2z uses the service name as the key (after prepending it with "z2z.opendht." to avoid name collision in OpenDHT). For example, an iTunes music share might be exported by z2z under the key, "z2z.opendht.Tom's Music", where "Tom's Music" is the name under which Tom is sharing his music library in iTunes. Section III-A explains this in more detail.

III. Design and Implementation

The current version of z2z is a command line program written in Java. This section starts with a few examples of command line usage to explain the basics. Then it shows how z2z works under the hood by following the message flows when exporting and importing service items. Finally we discuss some of the issues we encountered in implementing z2z.

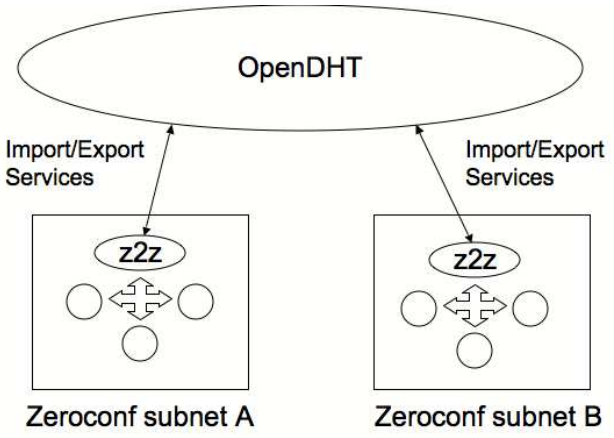

Fig. 1. *Two Zeroconf subnets A and B are exchanging local services with each other. Of course, z2z is not limited to only two subnets. Any number of subnets can export and import services to and from OpenDHT using z2z.*

A. Usage Examples

z2z exports local Zeroconf service announcements to OpenDHT, which then can be imported by other z2z processes anywhere in the world. For example:

```
z2z --export:opendht _daap._tcp
```

will export the iTunes music shares found in the local network to OpenDHT. When exporting to OpenDHT, z2z always stores each service using its service name as the key. For example, if one of the music shares exported by the command above is "Joe's Music", Joe's friend in a different network who wants to listen to Joe's music needs to issue the following command:

```
z2z --import:opendht --key "Joe's Music"
```

indicating that he wants to bring in any service stored under the name "Joe's Music". (If Joe's music share was password-protected, the friend should use as key "Joe's Music_PW" because iTunes adds the postfix to the service name of a protected share.) Also, any character that is neither a letter nor a digit will not be used in matching the key, and the comparison is case-insensitive, so the command above is same as:

```
z2z --import:opendht --key "joesmusic"
```

It is also possible to tell the exporter to use additional keys in addition to the service's own service name:

```
z2z --export:opendht _daap._tcp
    --key "music from office network"
```

will make z2z export the local iTunes shares not only under their own service names but also under the string "music from office network". This lets an employee working at home issue

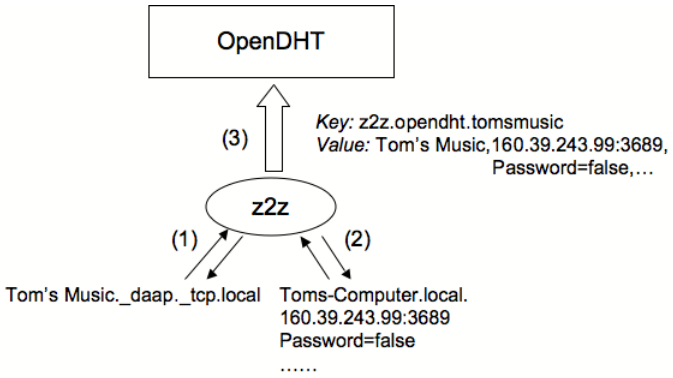

Fig. 2. *(1) z2z discovers a service instance of the type _daap._tcp by issuing a PTR query; (2) The service instance is further resolved to obtain the host name, IP address, and other additional information, using SRV, A, and TXT queries; (3) z2z constructs a key-value pair from the information and sends a put message to OpenDHT.*

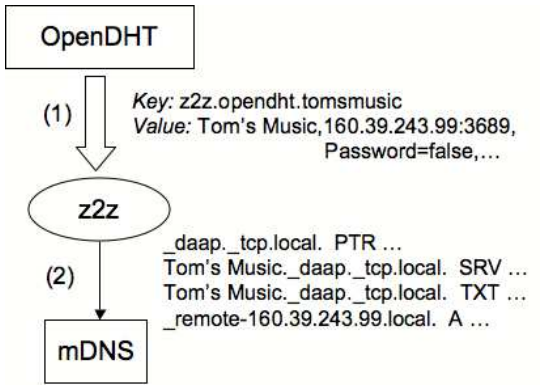

Fig. 3. *(1) z2z retrieves a service item from OpenDHT by sending a get message for the key "tomsmusic". (2) The service item is registered as if it had originated locally. This is done by inserting PTR, SRV, TXT, and A records into the local mDNS daemon.*

the following command to bring in *all* music shares of his office network.

```
z2z --import:opendht
    --key "music from office network"
```

Multiple keys are also allowed in the command line, in which case z2z will store multiple records in OpenDHT for the same service, one for each specified key.

B. Message Flow

1) Exporting: Figure 2 shows how z2z exports a service announcement to OpenDHT. First, z2z sends out a PTR query via multicast to discover service instances of the type _daap._tcp. In Bonjour parlance, this is called *browsing*, and it is performed by calling a Bonjour API function. Tom's iTunes music share is shown here as the example service instance discovered.

The discovered instance is then *resolved* in order to obtain the details of the service. This is also done by calling a Bonjour API function, which makes SRV and TXT queries to obtain the local host name, port number, and any other additional information about the service stored in the TXT record. (Figure 2 has the Password attribute as an example of what is stored in the TXT records.) In a normal Zeroconf service discovery situation, the IP address is not needed since the local host name can identify the host in the local network. However, since the service information that z2z exports to OpenDHT can be used from anywhere on the Internet, the local host name is not sufficient to locate the host. For this reason, z2z resolves the local host name to its IP address and includes it in the service information that it publishes to OpenDHT. Currently z2z does not export the service if the IP address is in the private address space [17]. A future version will address this issue (Section V).

Once z2z obtains all the relevant information about a service instance, it makes a *put* call into OpenDHT in order to store the service item under the specified keys (as explained in Section III-A). Each data item in OpenDHT has a Time-To-Live (TTL) value associated with it. A record is expired in OpenDHT unless it is refreshed with its TTL. Sending the *put* message again refreshes the record. Thus, z2z keeps sending the put request to OpenDHT as long as the service instance is present in the local network. The TTL of the service item and the interval by which z2z resends the put request are by default 5 minutes and 60 seconds, respectively, and they can be changed using the command line parameters.

2) Importing: Figure 3 shows another z2z process in another network importing Tom's music share that had been previously exported. First, z2z makes a *get* call to OpenDHT to retrieve the records stored under the key, "tomsmusic". z2z then *registers* the retrieved service into its local network. All the hosts in the network (including the same host on which the z2z process is running) will see the service as if it had originated from the local network, i.e., the iTunes applications running on this network will show "Tom's Music" as one of the shared music libraries in the network. This is accomplished by the Bonjour API functions that inject PTR, SRV, TXT, and A records into the local mDNS daemon.

Note that an A type record for a *fake* host name is added to mDNS. (We use names such as "_remote-160.39.243.99.local.", but any name can be used as long as it ends with ".local." and does not conflict with other host names in the local network.) This record points to the remote IP address of the machine that is actually providing the service. This trick of registering a remote service masquerading as a local one is called *proxy registering* in Bonjour terminology.

It is tricky to manage the lifetime of an imported service because the only way to learn that the service has been expired from OpenDHT is to try to *get* it. The approach taken by z2z is as follows. z2z keeps making *get* calls cycling through the keys specified by the user. There can be multiple keys and for each key there can be multiple service items. For each service item retrieved, it imports it if it is a new service. If an already imported service is retrieved again, it updates its refresh time-stamp. There is a thread that collects stale services

(those that have not been refreshed for a while) and removes them from the network. The interval between *get* calls and the stale threshold are by default 10 seconds and 5 minutes, respectively, and they can be changed using the command line parameters.

If there is another z2z process in the local subnet and it is exporting, the imported services will be discovered by that z2z exporter. We need a mechanism to prevent the exporter from exporting the imported service again. A short signature is added as a TXT attribute so that the exporter can distinguish the imported services from the native local services.

C. Implementation

1) C++ Prototype: The first prototype of z2z was implemented in C++ using the C version of the Bonjour client API. We developed and tested it in Mac OS X first and subsequently ported it to Windows. For OpenDHT access, we used the open-source `xmlrpc-c` library [18]. Using Cygwin environment [19], we were able to build and use the library in Windows as well.

This approach was problematic because the Bonjour client library in Windows uses the Winsock library, which is incompatible with Cygwin's socket-related functions. In particular, Cygwin's `select()` function fails when called with socket descriptors opened by the Bonjour library. Our workaround was to build two separate executables: one under native Windows environment (Microsoft Visual C++ compiler) and another under Cygwin environment (gcc compiler). The two executables communicated through a socket connection.

2) Open-source Java Implementation: The porting issues of the C++ prototype led us to rewrite z2z from scratch in Java. We used the Java version of the Bonjour client API, and for OpenDHT access, we used Apache XML-RPC [20]. The Zeroconf-to-Zeroconf Toolkit, version 1.0, was released under BSD license and is now available for download from SourceForge.net [21].

It is developed and tested under Mac OS X and Windows. In Windows, it requires Bonjour for Windows available from Apple [22]. (Bonjour for Windows is also automatically installed when iTunes is installed.) The support for Linux or other POSIX-compliant platforms providing Zeroconf through Avahi is planned for a future version.

3) Implementation Issues: The proxy registering mechanism described in Section III-B2 is unfortunately not available in the current Java Bonjour client API. The problem is that the current version of Java Bonjour API does not provide a way to inject a type A record into the local mDNS daemon. (The C API does provide this functionality.) As a workaround, z2z currently does a reverse lookup on the IP address and puts in the real, global host name as the value of the SRV record representing the service instance (as opposed to the fake .local name used when proxy registering is available). This eliminates the need of adding a type A record, but it makes it impossible to import services from those IP addresses that are reachable, but do not have global names associated with them. For example, two private address networks might be connected through a router.

A better workaround might be to use the fake .local host names, but instead of injecting a type A record into mDNS, z2z can listen for multicast and answer the A query itself. We will consider implementing this solution in a future version if the proxy registering API continues to be unavailable in the Java Bonjour client library.

The Bonjour API function for registering a service takes as a parameter the network interface index for which the service is registered. Usually it is set to a special value indicating all available interfaces. An interesting value one can pass here is one that indicates that the service should be registered for the local machine only. This option is supposed to register a service in such a way that it is only visible on the machine that registered the service, not any other host in the same local network. This is useful in z2z because it is sometimes undesirable to pollute the network with the imported services that are intended only for a single user. It is an issue especially in a large bridged wireless network where mDNS traffic can have a significant impact on the network performance. (See [23] for an example of such networks.)

Unfortunately, we were not able to incorporate this feature into z2z successfully. Under certain conditions, registering services for local machine only caused internal errors on the mDNS daemon in Mac OS X. Another problem with this option is that certain applications (iTunes being one of them) ignore the services registered in such a way, severely limiting the usefulness of the option.

IV. Related Work

Apple's solution for Zeroconf beyond local link is Wide-area Bonjour [24]. Wide-area Bonjour replaces the Multicast DNS in Bonjour with the conventional unicast DNS, thereby removing the link-local confinement of Bonjour services. This comes at a cost of setting up and maintaining a real DNS server, which makes Wide-area Bonjour unsuitable for a discovery solution for transient or ad hoc services. Moreover, the client hosts need to know the DNS servers to which they can send queries and publish services. In short, Wide-area Bonjour requires *configuration*.

We believe that z2z is the first attempt at interconnecting Zeroconf subnets using a DHT-based peer-to-peer network. But there have been a number of attempts at making Zeroconf services available beyond the local subnet.

Rendezvous Proxy [25] offers a simple GUI interface for a user to enter the information about a remote Zeroconf service, such as the IP address and port number where the service can be found. It makes the service available locally by performing the proxy registration, the same technique described in Section III-B2. It is intended as a way to establish a simple point-to-point connection when the user knows the exact nature and location of the service that he wishes to bring into his local network.

LogMeIn Hamachi [26] is a peer-to-peer virtual private network (VPN) solution that provides a virtual LAN connectivity over the Internet. Service discovery is not the main focus of this solution, but Zeroconf is claimed to work in the virtual LAN environment. The fact that it operates on top of a virtual LAN imposes a practical limit on the number of networks it can connect.

Simplify Media [27] applies the idea of social network to iTunes music sharing. Instead of the open peer-to-peer network used by z2z, it uses a private social network to enable iTunes music sharing among friends. Currently Simplify Media is an iTunes-only solution, whereas z2z is a generic solution for all Zeroconf services.

V. Discussion and Future Work

The current implementation of z2z allows multiple z2z processes running in a network to export or import the same set of services. Normally this is not a problem. When a service is exported to OpenDHT by multiple z2z processes, the effect is simply that the service gets refreshed more frequently. When a service is imported into a network by multiple z2z processes, Bonjour recognizes that the DNS resource records being registered are identical, and it treats them as the redundant announcements for a single service. In fact, Apple suggests this as a possible fault-tolerance mechanism [28].

The effect of such redundant registrations on a large local area network, however, needs to be investigated. The multicast traffic from mDNS can have a significant impact on the performance of a large network. This has led some network operators to employ filtering of mDNS traffic [23]. We plan to investigate if, and to what extent, the presence of z2z processes exacerbate the problem. If the redundant registration turns out to be a significant factor, it is straightforward to ensure that only one z2z process is responsible for importing a given remote service.

On the export side, we can eliminate the redundant OpenDHT refreshes by ensuring that only one z2z process is exporting a service type under a given key. This can be implemented using Bonjour. A z2z process can advertise a name constructed from the service type and the key that it intends to export, and then use Bonjour's built-in name conflict resolution mechanism to see if another z2z process is already exporting the type under the same key. It is unclear, however, that the reduction of OpenDHT calls outweighs the additional multicast traffic.

Privacy is another important consideration when z2z is used in a large network, especially when there are a large number of users, such as in a University campus network. When a user publishes a service using a Bonjour-enabled application (when a user shares his music library in iTunes for example), he expects his service to be available in the local network, but he may not be aware that the service can be carried outside the local network by z2z. Therefore we emphasize that z2z should be used in a way that respects the privacy of the users in the local network. It should be noted, however, that z2z does not introduce any new technology that facilitates the invasion of privacy. One can easily browse and resolve the local services using many other readily available tools, and then post the information on a web page, for example.

Currently z2z does not work in a network behind a Network Address Translation (NAT) gateway. We plan to implement the NAT traversal techniques [29], [30] in z2z in order to make it useful in the network settings that employ NAT, such as most home networks.

Another area of improvement is searching for services. Currently in order to import a service from OpenDHT, the key under which the service was exported must be given exactly. Various extensions are possible. A hierarchical index can be built and stored in OpenDHT from a list of keywords—possibly extracted automatically from the service name and the TXT record—in a manner similar to [31]. We can also replace OpenDHT with other DHT systems that support more complex queries.

References

[1] Zero Configuration Networking. [Online]. Available: http://www.zeroconf.org/

[2] Apple Inc. [Online]. Available: http://www.apple.com/

[3] Avahi. [Online]. Available: http://avahi.org/

[4] Debian GNU/Linux. [Online]. Available: http://www.debian.org/

[5] Ubuntu. [Online]. Available: http://www.ubuntu.com/

[6] DNS-SD service types. [Online]. Available: http://www.dns-sd.org/ServiceTypes.html

[7] Zero Configuration Networking (zeroconf) Working Group charter. [Online]. Available: http://www.ietf.org/html.charters/OLD/zeroconf-charter.html

[8] S. Cheshire, B. Aboba, and E. Guttman, "Dynamic configuration of IPv4 link-local addresses," RFC 3927, May 2005. [Online]. Available: http://www.ietf.org/rfc/rfc3927.txt

[9] S. Cheshire and M. Krochmal. (2006) Multicast DNS. Internet draft. [Online]. Available: http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt

[10] ——. (2006) DNS-based service discovery. Internet draft. [Online]. Available: http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt

[11] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "Opendht: A public dht service and its uses," 2005. [Online]. Available: citeseer.ist.psu.edu/rhea05opendht.html

[12] OpenDHT home page. [Online]. Available: http://opendht.org/

[13] The Bamboo distributed hash table. [Online]. Available: http://bamboo-dht.org/

[14] XML-RPC home page. [Online]. Available: http://www.xmlrpc.com/

[15] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)*, vol. 20, no. 8, pp. 1489–1499, 2002. [Online]. Available: citeseer.ist.psu.edu/castro02scribe.html

[16] M. Freedman, K. Lakshminarayanan, and D. Mazieres, "Oasis: Anycast for any service," 2006. [Online]. Available: citeseer.ist.psu.edu/757191.html

[17] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address allocation for private internets," RFC 1918, Feb. 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1918.txt

[18] XML-RPC for C and C++. [Online]. Available: http://xmlrpc-c.sourceforge.net/

[19] Cygwin home page. [Online]. Available: http://www.cygwin.com/

[20] Apache XML-RPC. [Online]. Available: http://ws.apache.org/xmlrpc/

[21] Zeroconf-to-Zeroconf Toolkit (z2z). [Online]. Available: http://sourceforge.net/projects/z2z/

[22] Bonjour for Windows. [Online]. Available: http://www.apple.com/support/downloads/bonjourforwindows.html

[23] OIT filters mDNS. [Online]. Available: http://www.net.princeton.edu/filters/mdns.html
[24] S. Cheshire and D. H. Steinberg, *Zero Configuration Networking: The Definitive Guide*. Sebastopol, CA: O'Reilly Media, 2005, ch. 5.
[25] Rendezvousproxy: Tutorial. [Online]. Available: http://ileech.sourceforge.net/index.php?content=RendezvousProxy-Tutorial
[26] LogMeIn Hamachi. [Online]. Available: https://secure.logmein.com/products/hamachi/vpn.asp
[27] Simplify Media. [Online]. Available: http://www.simplifymedia.com/
[28] Technical Q&A QA1311: Registering a Bonjour service multiple times. [Online]. Available: http://developer.apple.com/qa/qa2001/qa1311.html
[29] J. Rosenberg. (2007) Interactive connectivity establishment (ICE): A protocol for network address translator (NAT) traversal for offer/answer protocols. Internet draft. [Online]. Available: http://tools.ietf.org/html/draft-ietf-mmusic-ice-17
[30] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN - simple traversal of user datagram protocol (UDP) through network address translators (NATs)," RFC 3489, Mar. 2003. [Online]. Available: http://tools.ietf.org/html/rfc3489
[31] L. Garcés-Erice, P. A. Felber, E. W. Biersack, G. Urvoy-Keller, and K. W. Ross, "Data indexing in peer-to-peer dht networks," in *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 200–208.