Zero Configuration Networking: Implementation, performance, and security

Farhan Siddiqui, Sherali Zeadally, Thabet Kacem and Scott Fowler

Zero Configuration Networking: Implementation, Performance, and Security

Farhan Siddiqui[1], Sherali Zeadally[2], Thabet Kacem[2] and Scott Fowler[3]

[1]School of Information Systems and Technology, Walden University, Minnesota, USA

[2] Department of Computer Science and Information Technology, University of the District of Columbia, Washington DC, USA

[3] Department of Science and Technology, Linkoping University, Sweden

Abstract

The ubiquitous access to wired and wireless networks is making information access possible from anywhere, anytime, and any device. Today, end-users are also highly mobile, often equipped with a range of portable devices, and they expect service availability when they require it. In addition, they do not want to be burdened by complex configurations before they can discover and use services. The Zero Configuration (Zeroconf) networking technology promises to alleviate this configuration burden by allowing users to discover services and devices with little end-user intervention. We compare two popular implementations of Zeroconf namely, Avahi and Mono.Zeroconf running on Linux and Windows XP operating systems respectively. We evaluate their performance using service discovery time as the performance metric. Our empirical results show that Linux Avahi yields almost 99% improvements in service discovery time over Windows Mono.Zeroconf. We also discuss security solutions that can be deployed to enhance the security of Zeroconf networks. We further investigate the performance of the IP Security (IPSec) protocol when used by our Mono.Zeroconf implementation running on the Windows XP platform. With IPSec, service discovery time increases by almost 45% with our prototype implementation.

Keywords: Zeroconf, Avahi, Mono.Zeroconf, IPSec, Service Discovery

1. Introduction

The development of increasingly sophisticated computer devices and networks has increased the need to maintain efficiency as well as the ease of use of these technologies by end-users. In this context, Zero Configuration Networking, well known as Zeroconf [1], was established to enable ease of technology usage in particular situations. In fact, Zeroconf is not a fit-all solution, but is appropriate mostly for small or home networks or networks that do not require a high level of administration. Zeroconf relies on a set of technologies which can be described as follows. First, the address auto-configuration process replaces the traditional Dynamic Host Configuration Protocol (DHCP) server introducing a link-local method of addressing coupled with the mechanism of auto-configuration described in IPv4 (known as IPv4LL) and as well in IPv6 [2]. Second, the name-to-address resolution replaces the need for a Domain Name System (DNS) server. This process resolves its queries using IP multicast (hence the name

Multicast DNS (mDNS)). The latter works in conjunction with link-local addressing but is self-dependent. Third, the service discovery [3] enables the user to find the available services over the network, replacing the directory service and this process is known as DNS-SD which is compatible with mDNS. The last technology used is the multicast address allocation which replaces the need for a multicast server. The Zeroconf Multicast Address Allocation Protocol (ZMAAP) handles this process and solves most of multicast addresses conflict. Zeroconf has been developed by many entities, after being introduced first by Apple and further enhanced by the Zeroconf Working Groups of the IETF. The main implementations of Zeroconf may be found in Apple's Bonjour, Avahi, ZCIP, CUPS, Mono.Zeroconf and J-share. Since Zeroconf protocols are not secure as compared to standard configured network protocols, the development of lightweight security mechanisms remains an area of improvement with the Zeroconf technology.

In this work we present an overview of the Zeroconf technology, its architectures and its applications. We describe two major Zeroconf implementations: Avahi and Mono.Zeroconf running on Linux and Windows XP operating systems respectively. We discuss practical deployment issues for the Zeroconf technology. Furthermore, we present an empirical performance evaluation of these Zeroconf implementations using an experimental testbed consisting of network devices such as workstations, printers, iPods, etc. Finally, we present a secure approach that can improve the security of Mono.Zeroconf on Windows XP.

The rest of this paper is organized as follows. In section 2 we present an overview of the Zeroconf technology. Section 3 provides a review of several currently existing Zeroconf implementations. In section 4, we discuss details of our Zeroconf implementation, followed by sections 5 and 6 which present the performance analysis of the two Zeroconf implementations namely, Avahi and Mono.Zeroconf. In section 7 we discuss the various security aspects of Zeroconf. Section 8 presents the implementation and performance analysis of a security solution for Zeroconf. Finally, in section 9 we make some concluding remarks.

2. Overview of Zeroconf Technology

The Zeroconf Working Group of the Internet Engineering Task Force (IETF) [4] describes the goal of Zero Configuration Networking (Zeroconf) as the capability to enable networking in the absence of configuration and administration. Zero configuration networking is required for environments where administration is impractical or impossible, such as in the home or small office, embedded systems 'plugged together' as in an automobile, or to allow impromptu networks such as "between the devices of strangers on a train". Basically, to reduce network configuration to zero (or near zero) in Internet Protocol (IP) networks, we need to support the following features [1]: a) Distribute IP addresses (without a Dynamic Host Configuration Protocol (DHCP) server), b) Provide name resolution (without a Domain Name System (DNS) server), c) Find and list services (without a directory service), d) Distribute multicast IP addresses, if necessary (without a multicast server). Zeroconf is a link-local technology [5] whose addressing and naming schemes make sense just in a particular network. In fact, link-local addresses and names are not global and are not unique globally. Therefore, Zeroconf is intended to be deployed in small Local Area Networks (LANs) based on wired or wireless technologies especially in circumstances where automatic configuration is indispensable. In networks where traditional techniques such as DHCP or DNS cannot be used, it is highly recommended to use Zeroconf. However, there are situations where Zeroconf is not a good option: medium or large networks, networks that need a high degree of security and control, large public access networks, and networks that have low bandwidth and high latency (such as some wireless networks). When it is not used properly, Zeroconf may generate many problems that affect the usability of the network. Some networks where Zeroconf is expected to be used include: home and small office networks, ad hoc networks at meetings and conferences using wireless technology, instinctive sharing of information between two devices.

The need for a new technology that has abilities such as ease of use, as well as scalability, has been highly desired since late 1990s. In fact, similar software called Apple's AppleTalk existed prior to Zeroconf and it provided automatic configuration. But the main problem with the AppleTalk technology was that it was not scalable especially with the TCP/IP model. To support TCP/IP stacks, the Zeroconf IETF Working group was set up in 1999 to ensure that required features are incorporated in Zeroconf implementations.

Currently, Zeroconf is deployed in various applications such as network printing, collaborative document editing, and instant messaging.

3. A Review of Current Zeroconf Implementations

The current major implementations of Zeroconf are summarized below.

3.1 Apple Bonjour

Bonjour, formerly called Rendezvous, is an implementation of Zeroconf's service discovery protocol. It is used in several operating systems such as Mac OS X and Microsoft Windows. Bonjour uses multicast DNS service records to locate devices such as printers, computers and many other services. Bonjour is utilized to discover services on a LAN and to allow users to set up a network without any configurations. It is often used by operating systems to locate printers and file sharing servers. Software products such as iTunes and iPhoto also use Bonjour for multimedia file sharing. It can also be used for collaborative document editing. In addition, Bonjour is used by Safari to find local web servers and configuration pages for local devices. The Bonjour Browser can be used to explore services announced by these applications. Bonjour only works within a single broadcast domain, which is usually a small area if the DNS server is not appropriately configured. Bonjour does not have the capability to advertise services to the public Internet. Moreover it does not provide any extra access to services, even on the same LAN; it merely advertises their existence. For instance, if Bonjour is used by a client to discover some local computers, it cannot provide access to these computers.

3.2 Avahi

The Avahi system facilitates service discovery on a local network. Avahi allows a new host on the local network to view other hosts and communicate with them (chat, find printers or shared files). Such a technology has already been implemented into the Apple Mac OS X. Avahi is mainly based on Lennart Poettering's flexmdns [6] implementation for Linux and implements DNS service discovery and Multicast DNS specifications for Zeroconf Networking. It is based on DBus to ensure the communication between the user applications and the system daemon. The role of the daemon is to coordinate application efforts in order to minimize the traffic imposed on networks. The Avahi mDNS responder is actually completely implemented and has passed all tests in the Apple Bonjour conformance test suite. Moreover, it supports some unique features such as the correct mDNS reflection across LAN segments. It is implemented as a C library ("avahi-core") which can be embedded into other applications.

3.3 Windows CE 5.0

The Windows CE Wireless Zero Configuration service configures the wireless network adapter to connect to an available wireless network. If there are two networks covering the same area, it is possible to configure a preferred network order and the device will try to contact each network in the order defined until it locates one that is active. It is also possible to limit association to only the configured, preferred networks. If the machine is unable to locate a wireless Access Point (AP) nearby, by default, a Windows CE-based wireless client device configures the network adapter to use ad hoc mode. The wireless network adapter can also be configured to automatically get disabled if no AP can be located. These network adapter enhancements are integrated with security technologies. If authentication with one wireless AP or network fails, the device will attempt to authenticate with the additional detected wireless networks. The Windows CE-based client device attempts to perform an 802.11 shared key authentication if the network adapter has been preconfigured with a WEP shared key. If shared key authentication fails or if the network adapter is not preconfigured with a WEP shared key, the network adapter reverts to open system authentication. [7]

3.4 Other Link Local IPv4 Implementations

3.4.1 ZCIP

ZCIP is an implementation of the ad-hoc link-local IP auto-configuration algorithm described in the IETF Internet Draft "Dynamic auto-configuration of IPv4 link-local addresses" [8]. The characteristic feature of ZCIP is to get an IP address without manual assignment or use of DHCP. It is also integrated in Linux, UNIX and BSD systems and uses some specific library such as libcap and libnet which use regular socket interfaces provided in these operating systems kernels. ZCIP completely operates in the user space.

3.4.2 CUPS

CUP is an implementation of the Internet Printing Protocol (IPP). CUPS implements an effective method of discovering network printers. However, a major drawback of the CUPS technique is that its service discovery protocol relies on broadcast. Therefore, it exhausts most of the network bandwidth. CUPS integrates a Service Location Protocol (SLP) if it finds the appropriate library which is usually OpenSLP. Consequently, it supports a standardized way to locate printers not only on the network but also those across routers without generating excessive network traffic.

3.4.3 J-Share

J-share is a simple LAN file sharing utility based on Zeroconf/Bonjour/UPnP protocol and Java platform. In order to work appropriately, J-share needs the Java Runtime Environment (JRE) installed and the port 6666 well configured to enable traffic if the host uses a firewall. The advantages of J-share are two-fold. First, J-share facilitates finding, sharing and downloading files easily on a LAN without the need of a central server or any special configuration. Second, the fact that J-share is platform independent allows users of different operating systems to share files by avoiding the complex process of configuring Windows Sharing (SMB), Apple File Sharing (AFP), FTP or NFS. J-share is a complete Java project and some of the used libraries are released under the Apache License v2.0. The software supports drag-and drop that eases the file sharing process for users.

3.4.4 Mono.Zeroconf

Mono.Zeroconf provides a unified API for performing the most common Zeroconf operations on several platforms and subsystems. Using Mono.Zeroconf, developers can employ a single API that will work regardless of the underlying implementation that a particular operating system uses. Developers can publish services that will be advertised to other computers on the network and also query for services that can be available. Mono.Zeroconf abstracts away the differences between providers (mDNSResponder/Avahi). Therefore, some of rarely used features of each provider stack are not exposed through Mono.Zeroconf.

Table 1 summarizes various Zeroconf implementations and highlights their main features.

Zeroconf Implementation	Main Features
Apple Bonjour	Implementation of Zeroconf's service discovery protocol. It is used in several operating systems such as Mac OS X and Microsoft Windows. Bonjour works only in short-distance areas since its messages can only reach users located on the same link.
Avahi	Avahi is mainly based on Lennart Poettering's flexmdns [6] implementation for Linux and implements DNS service discovery and Multicast DNS specifications for Zeroconf Networking. It is based on DBus to ensure the communication between the user applications and the system daemon.
Windows CE 5.0	Based on a LLMNR (Link Local Multicast Name Resolution)-based implementation
ZCIP	Implementation of the ad-hoc link-local IP auto-configuration algorithm described in the IETF Internet Draft "Dynamic auto-configuration of IPv4 link-local addresses"
CUPS	Implementation of the Internet Printing Protocol (IPP)
J-Share	A simple LAN file sharing utility based on Zeroconf/Bonjour/UPnP protocol and Java platform. J-share is platform independent allows users of different operating systems to share files by avoiding the complex process of configuring Windows Sharing (SMB), Apple File Sharing (AFP), FTP or NFS.
Mono.Zeroconf	A cross platform Zero Configuration Networking library for Mono and .NET. It provides a unified API for performing the most common Zeroconf operations on a variety of platforms and subsystems.

Table 1: Zeroconf Implementations

4. Our Prototype Zeroconf Implementations: Avahi and Mono.Zeroconf

We deployed two major implementations of Zeroconf: Avahi and Mono.Zeroconf. Since Avahi runs in Linux and Mono.Zeroconf runs in Windows, we configured the machines so that they can be dual-bootable. We used the same machines, Dell XPS M1210, in order to get consistent results that will constitute a solid basis for comparing the performance results of the two Zeroconf implementations. As shown in figure 1, the main goal was to implement a service discovery layer to both Zeroconf implementations. This service discovery layer supports the discovery of both services and devices in the Zeroconf network.

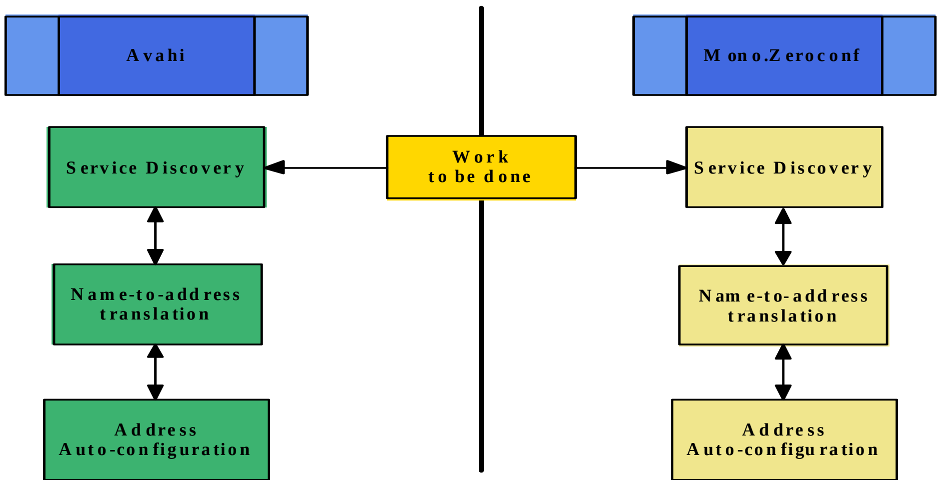
Avahi
Mono.Zeroconf
Service Discovery
Work to be done
Service Discovery
Name-to-address translation
Name-to-address translation
Address Auto-configuration
Address Auto-configuration

Figure 1: General Architectural View

4.1 Avahi

Avahi [9] is a cross-platform software implementing the main functionalities of Zeroconf: multicast Domain Name System (mDNS), Domain Name System- Service Discovery (DNS-SD) and Internet Protocol version 4 Link Local addressing (IPv4LL). Avahi also allows programs to publish and discover services and hosts within a local network without any specific configuration. It is licensed under the GNU Lesser General Public License (LGPL). Moreover, it offers various language bindings such as Python, Mono or QT and is shipped with most Linux or BSD distributions.

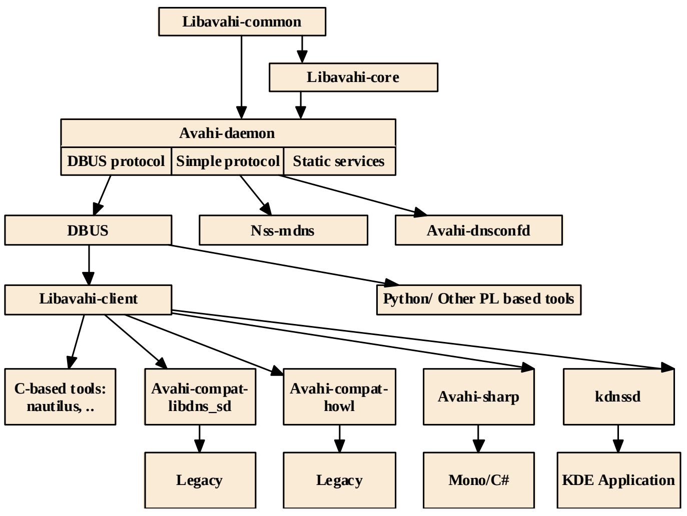

Figure 2: Avahi Architectural View

Figure 2 shows the package diagram of Avahi. The major components of Avahi are as follows:

- *Libavahi-common:* contains some functions used by the Avahi-daemon and the mDNS stack
- *Libavahi-core:* implements a flexible mDNS/DNS-SD stack and is highly used by developers
- *Avahi-daemon:* implements the MDNS stack by using Libavahi-core
- *Libavahi-client:* facilitates the use of DBUS API by hiding DBUS internals

4.1.1 Avahi Deployment

There are two phases in the deployment of Avahi. The first phase involves deploying the Avahi client and second phase involves running the Avahi source code. Each of these phases requires proper configuration of a set of packages. For instance, some of the packages required for Avahi client include Expat (a stream-oriented XML parser library), Libdaemon (a free library facilitating the writing of Linux daemons), and other graphical libraries such as GTK2 and Glade2. The next step involves the configuration of core packages. Avahi allows the discovery of several services in the Zeroconf network. The discovery window presents important information related to each kind of service, such as the service type, transport protocol, etc. The service type is recognized by the regular expression "_applicationprotocol_transportprotocol" (where the application protocol can be http, etc. and the transport protocol can be either TCP or UDP). The Avahi discovery service window displays interactively the available services in a dynamic tree view i.e. it handles the discovery of new services as well as the termination of others. Furthermore, the discovery window offers a command line from which the user can browse, publish and resolve services.

4.2 Mono.Zeroconf

Mono.Zeroconf is a cross-platform of the Zeroconf library for Mono and .NET platforms. Mono provides the required software to run .NET clients and server applications on Linux, Solaris, MAC OS X, Windows, and UNIX. Windows users have to install Bonjour (Apple) before using Mono.Zeroconf. The Mono.Zeroconf website [10] provides the option of downloading either the source code for windows or the windows binaries: while the first option is dedicated to developers who want to use the platform to build their own solution, the second option is useful for simple users aiming to benefit from the functionalities of Mono.Zeroconf in order to run some applications. Developers have the choice to use either Mono or Visual Studio .NET 2005 or a higher version. A source code for Mono.Zeroconf, which consists of a Visual Studio 2008 solution, is also available at the Mono web site at [10].

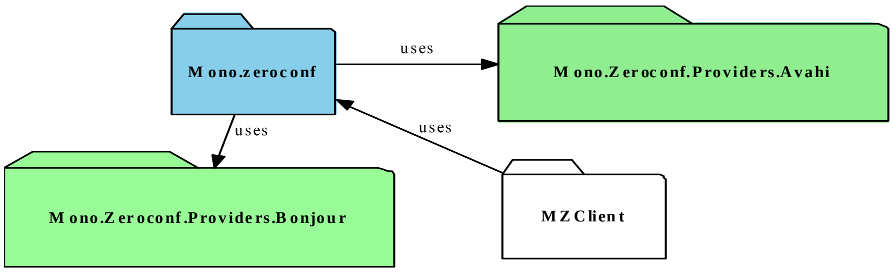
Mono.zeroconf
uses
Mono.Zeroconf.Providers.Avahi
uses
uses
Mono.Zeroconf.Providers.Bonjour
MZClient

Figure 3: Mono.Zeroconf Architectural view

Figure 3 shows the general architectural view of the main packages composing Mono.Zeroconf in our prototype implementations. These packages are as follows:

- Mono.Zeroconf: The Mono.Zeroconf package defines the main interfaces for the major functionalities of Zeroconf such as the interface for registering a service "IRegisterService" or the one for browsing for a service "IBrowseService".
- Mono.Avahi.Providers: The Mono.Avahi.Providers package represents the provider that the Mono.Zeroconf package works with. It can be either Bonjour or Avahi.
- MZClient: The core for the application resides in the MZClient package. Effectively, the main method that launches the process is loaded in this class. We implemented this client application.

4.2.1 Mono.Zeroconf: Creation of Service Discovery Frame

The following is a description of the steps taken to create the Mono.Zeroconf service discovery frame:

- **Customizing the List of Services**

After installing Visual Studio 2008, downloading Mono.Zeroconf source code and compiling it in VS2008, it is possible to begin browsing for services. The initial downloadable version of Mono.Zeroconf includes only limited services. In fact, it offers the command line **MZClient** which displays the history of services (including found, lost, registered and resolved) as well as some additional information for each one of them such as domain name, etc. Therefore, we customized the source code in order to detect the required services. We assume here, that we would like to discover workstations, printers, websites and iTunes.

- **Retrieving Service Details**

Since the command line did not bring that expected ease of use especially for users with no programming skills, it was necessary to transform the data displayed in the command line into a user-friendly format. Therefore, we needed to retrieve the details of the services being discovered. This phase of the implementation was challenging especially from a programming perspective. The **Main** method, the **"OnServiceAdded"** and **"OnServiceResolved"** were static and it was not possible to retrieve the data in a dynamic array list. Hence, the solution was to get the details (name, regular expression type of service and domain) into the static Array lists in the **"OnServiceAdded"** and then transform these array lists into arrays in the **Main** method. Afterwards, these new arrays will serve as arguments to call the form that will display the services and the relative devices in a tree format. The new form is called "frame3" and is located in a separate file **"frame3.cs".**

- **Populating the Service Discovery Window**

To populate the service discovery window, **"frame3"** was called via its constructor (which takes eight arguments that are arrays and contain details about discovered services such as service name, regular expression, domain, status, full name, IP address, port number and interface number. The last four arguments (arrays) are populated the same way as the first four ones but the difference is that the respective array lists are populated in the **"OnServiceResolved"** method. The difficulty at this level lies in the ability to dynamically add the nodes in the tree view especially since the **"InitializeComponents"** method does not really allow adding dynamic variables. Consequently, we put the declarations of the tree nodes as class variables and populated the nodes dynamically from the values of the arrays' elements with some filtering options (regular expression of service's type) to distinguish between the services corresponding to each device. In spite of populating the nodes, one essential procedure was to add an on-click event so that further details are displayed for each service, such as the full name of the relative device, its IP address, its port and the interface on which it is running. Some of the last steps involved adding icons for each type of device and each type of service, in order to make the process more user-friendly and adds the ease of use which is one of the major e goals of Zeroconf.

Finally, the result of the implementation steps above is the Zeroconf service discovery frame, which consists of a dynamically populated tree structure where each node represents a type of service in the network. Every sub-node in each category represents a device implementing the service. The details of these devices (i.e. IP address, name, port number and interface) are displayed interactively once clicked on the corresponding node.

5. Experimental Testbed and Performance Evaluation

In this section, we present the experimental testbed used to conduct experiments for measuring the performance of Zeroconf implementations. For all tests, we have used Dell XPS M1210 laptops (dual-bootable: Windows XP Pro

SP2 and Linux Debian 4). The performance evaluation metric used in all tests conducted is the discovery time of services, which we have measured in milliseconds (ms). In the following sub-sections, we present details of various tests performed for evaluating Zeroconf, and interpretation of the results obtained. It is worthwhile noting that we repeated the same test several times and recorded the average value obtained in each case.

5.1 Performance Evaluation of Mono.Zeroconf

5.1.1 Test no. 1

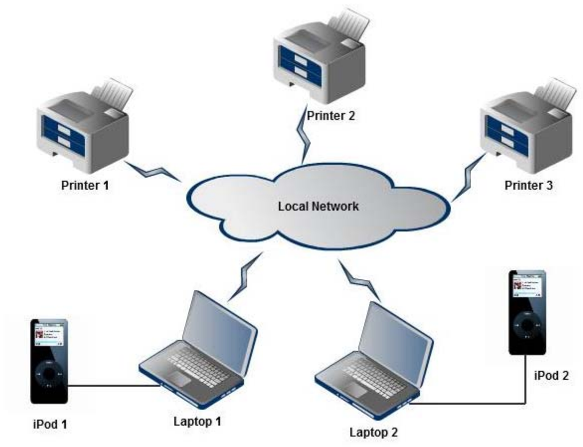

Figure 4: Network testbed for the first set of tests

Device Name	No. of Devices	Average Discovery Time (ms)
Workstations discovery	1 workstation	124.6
	2 workstations	347.25
Printers discovery	1 printer	49.4
	2 printers	128.5
	3 printers	208
Printers' web page discovery	1 printer web page	71.6
	2 printer web pages	234.5
	3 printer web pages	318
iPod discovery	1 iPod	93.4
	2 iPods	411

Table 2: Service discovery time (in milliseconds) for workstation, printers, websites and iTunes

Figure 4 shows the experimental setup used to evaluate the performance of Mono.Zeroconf. The testbed consists of two laptops, Linux Debian (laptop 1) and Windows XP Pro SP2 (laptop 2), three printers and two iPods. The purpose of this first set of tests was to measure the time taken to discover services that already exist in a network. In table 2, we present the service discovery times (in ms) observed for the discovery of various services. Regarding the discovery of workstations, we found that the Linux machine is *faster* to be discovered than the windows one. That may be due to the fact that the Avahi daemon is running on it, which enables listening and advertising services quicker than Mono.Zeroconf does. Regarding the printers, we note that the printers' average service discovery time is faster than the printer web page discovery service. In fact, while the first one concerns the printing process itself, the second takes care of discovery of the websites available on the local network about the configuration pages of these printers (that can be accessed with the IP address and the port number provided by the application). For iTunes, it is faster to discover the local iTunes than the distant ones. This might be explained by the fact that in the case of an overloaded network, services with higher priority (local services) are discovered before others. It was also observed that the service discovery time increased linearly with the increase in the number of discovered devices/ services.

5.1.2 Test no. 2

Table 3 specifies the set of tests that were conducted in order to represent particular situations such as the joining of a new network service (as shown in figure 5). The aim of conducting test no. 2 was to determine the amount of time it takes for a new service to join the network, and the variation of this time with the type of service being added. It is worthwhile noting that in test no.1, we determined the time taken by a Zeroconf client to discover services that were already advertised in a Zeroconf network. To determine these service discovery times, we added two more laptops (Linux and Windows) to our initial testbed. Our tests indicate that the time to discover a newly added service is considerably longer than the already discovered services. Furthermore, since laptops 3 and 4 join the network after a while, the time gap between the discovery of services and the advertisement of new ones further increases the time to the whole discovery process.

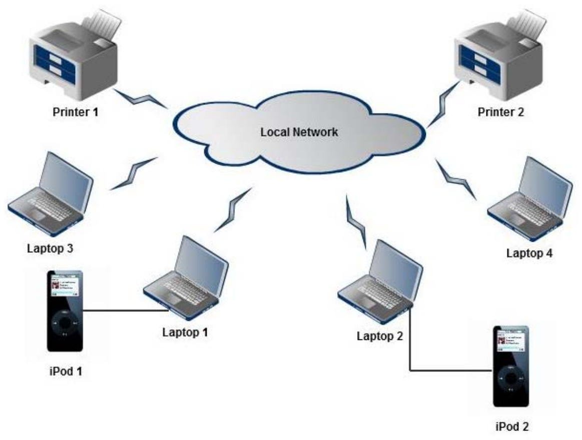

Figure 5: Network testbed for the second set of tests in Mono.Zeroconf

Device Name	No. of Devices	Average Discovery Time (ms)
Workstations discovery	1 workstation	71.75
	2 workstations	239
	3 workstations	103.66
	4 workstations	437.2

Printers discovery	1 Printer	113.25
	2 Printers	132
	3 Printers	156
Printers’ web page discovery	1 printer web page	156.8
	2 printer web pages	146.4
	3 printer web pages	183.25

Table 3: Service discovery time (in milliseconds) for workstations, printers and websites when new devices (laptops 3 and 4) join the network in Zeroconf

5.1.3 Test no. 3

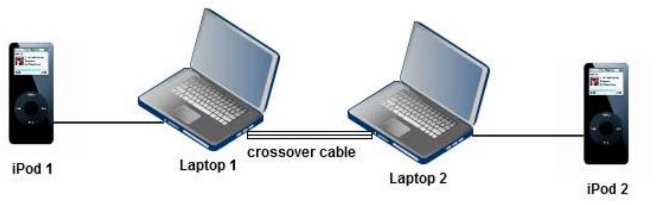

Figure 6: Network testbed for the third set of tests in Mono.Zeroconf

Device	**No. of Devices**	**Average Discovery Time (ms)**
Workstations discovery	1	256
	2	521
iTunes discovery	1	246.6
	2	256

Table 4: Service discovery Time for workstations and iPods in Mono.Zeroconf

Table 4 measures the services discovery time in a minimal network, shown in figure 6, formed by two laptops (Windows) linked by a crossover cable and two iPods connected to each one of them.

5.1.4 API usage

To better understand the time consumed when a new service joins the network, we need to find out the related API calls invoked in discovering a new service. Figure 7 shows the classes that are used when a new service is discovered using Mono.Zeroconf. As shown, the event method “*OnServiceAdded()*” that is called in the main class “*ZeroconfClient*”, located in the package (namespace) “*MZClient*”, is responsible for the discovery of services. It calls an object of the class “*ServiceBrowseEventArgs*” that is located in another package (namespace) “*Mono.Zeroconf”*. This class is an aggregate for the “*IResolvableService*” interface which provides the declaration of some methods that provide some specific information when resolving a service in Mono.Zeroconf. The latter is a child of another interface: “*IService*” which is the parent interface for all the service-related interfaces and provides through the declaration of its methods some basic information about the service. In other words, the package

“Mono.Zeroconf” contains all the classes that maintain Zeroconf functionalities, and these classes are called in the MZClient package (more precisely in the ZeroconfClient class that contains the user interface, as needed).

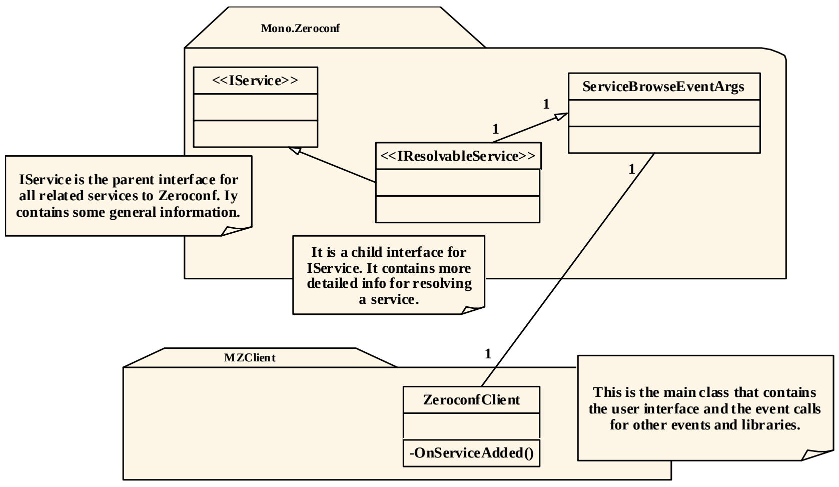
Mono.Zeroconf
<<IService>>
ServiceBrowseEventArgs
<<IResolvableService>>
1
1
1
1
IService is the parent interface for all related services to Zeroconf. Iy contains some general information.
It is a child interface for IService. It contains more detailed info for resolving a service.
MZClient
ZeroconfClient
-OnServiceAdded()
This is the main class that contains the user interface and the event calls for other events and libraries.

Figure 7: Class diagram of discovery time in Mono.Zeroconf

(“1” indicates the number of associations between the objects of each class)

5.2 Performance Evaluation of Avahi

5.2.1Test no. 1

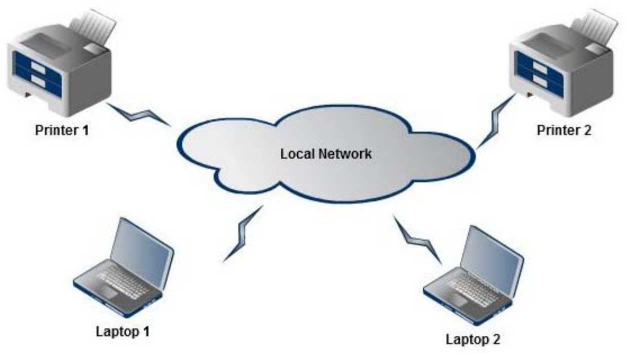

Figure 8: Network testbed for the first test in Avahi

Device Name	No. of Devices	Average Discovery Time (ms)	Average Improvement of Avahi vs. Mono.Zeroconf (%)
Workstation Discovery	1 workstation	0.2065	99.83
	2 workstations	0.0507	99.98
Printers Discovery	1 printer	0.0322	99.93
	2 printer	0.0277	99.97
Printers web page Discovery	1 printer web page	0.0342	99.95
	2 printer web pages	0.0287	99.98

Table 5: Service discovery time for workstation, printers, and websites in Avahi

This set of tests shows the discovery time for several kinds of services in the Zeroconf network: Workstation discovery, Printer discovery and printers' web page discovery that can be technically referenced. The results of the service discovery times are shown in table 5 which also presents the percentage improvement (of almost 99%) of Avahi over Mono.Zeroconf.

5.2.2 Test no. 2

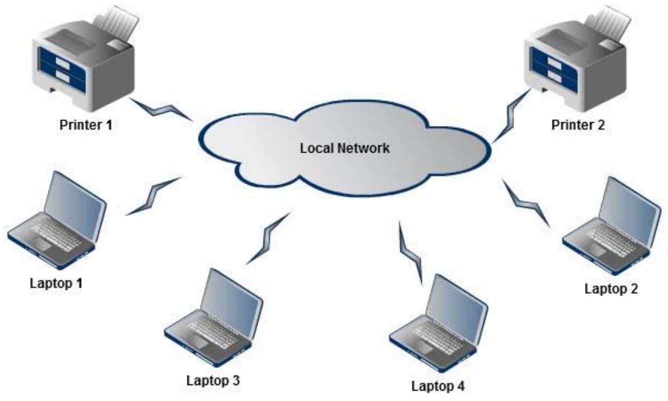

Figure 9: Network testbed for the second set of tests in Avahi

Device Name	No. of Services	Average Discovery Time (ms)	Average Improvement of Avahi over Mono.Zeroconf (%)
Workstation Discovery	1 workstation	0.1602	99.77
	2 workstations	0.0707	99.97
	3 workstations	0.0695	99.93
	4 workstations	0.0855	99.98
Printers Discovery	1 printer	0.0297	99.97
	2 printers	0.03	99.97
Printers web page Discovery	1 printer web page	0.0305	99.96
	2 printer web pages	0.0302	99.97

Table 6: Service discovery time for workstation, printers, and websites when new devices join the network in Avahi

The second set of tests is dedicated to investigate a particular behavior of the system when new devices join an a network. In other words, we added two other Linux Debian machines (figure 9) with the same characteristics (identical Dell machines) to the network in order to measure the discovery time with increasing number of end-systems. Each of the four tests was conducted in each Linux machine. The results (shown in table 6) indicate that the time required to discover machines 3 and 4 is greater than the time for discovering machines 1 and machine 2. The additional time taken in the former case can be attributed to the process of auto-configuration that takes place to set up the new machines in the network. Similarly, machine 1 for example discovers machine 3 after that machine 4 does since machine 3 and machine 4 joined the network late after machine 1 and 2. The printers’ and printers' web page discovery times are practically the same as in the previous test: there is no significant variation in the results in this case.

5.2.3 Test no. 3

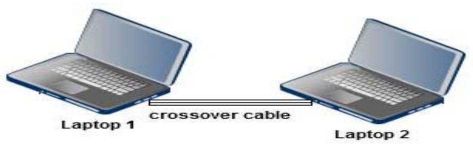

Figure 10: Network Architecture for the third set of tests in Avahi

Device Name	No. of Devices	Average Discovery Time (ms)	Average Improvement of Avahi vs. Mono.Zeroconf (%)
Workstation Discovery	1 workstation	0.1242	99.95
	2 workstations	0.1087	99.97

Table 7: Discovery Time for workstations in Avahi

After measuring the discovery time in a large network (4 workstations and 2 printers), in test no. 3, we reduce the size of the network and observe the change in the discovery time. We used two Linux workstations from the previous network and linked them by a crossover cable (figure 10). We observed that the discovery time in test 3 is greater than the time recorded in test 1. The additional time in test 3 may be attributed to the time needed to perform the Zeroconf initialization steps in a newly formed network, in order to fully operate and be able to discover services in the network.

5.2.4 API usage

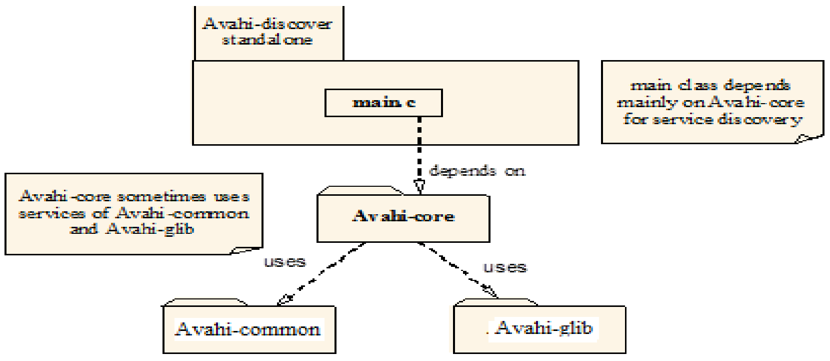
Avahi-discover standalone
main.c
main class depends mainly on Avahi-core for service discovery
depends on
Avahi-core sometimes uses services of Avahi-common and Avahi-glib
Avahi-core
uses
uses
Avahi-common
Avahi-glib

Figure 11: Overview of the package dependency for the service discovery in Avahi

The service discovery time is the result of execution of various API calls as follows. First, it is worthwhile noting that the Avahi program is written in C language; however Avahi presents bindings for other programming languages. So, after examining the list of headers in the main program, located under "Avahi-Discover-Standalone" (figure 11) directory, we found that the service discovery is using basically the functions of Avahi-core which is in fact a library used by many other packages, such as the Avahi-daemon. We found that most of the service discovery time is consumed by the functions of the Avahi-core (Avahi-core itself uses some services of Avahi-common and Avahi-glib).

6. Implementation Experiences with Mono.Zeroconf and Avahi

In this section, we briefly discuss some of the challenges that we faced while deploying and running the Zeroconf implementations on Linux and Windows platforms.

6.1 Mono.Zeroconf Deployment

We faced difficulties while retrieving the details of each service. Every attempt to retrieve service details resulted in getting stuck for a while since the method that contains this information is static and we needed to access it in a dynamic fashion. The dynamic access was required to enable passing the required parameters to the final frame. The solution was to use static ArrayLists and convert them later into Arrays that will be passed as parameters of the constructor of the new frame class. The final challenge was to manage the dynamic updates of the tree structure so that it supports new devices that join the Zeroconf network. That was solved by using the information provided by the Arrays (received from the constructor). Furthermore, we also applied some sorting and graphical techniques that allowed us to overcome this problem.

6.2 Avahi Deployment

The difficulties with Avahi deployment were primarily related to the time consuming configuration work of packages required by Avahi for running the source code. We overcame this problem by using the Synaptic package manager, a powerful tool in Linux that offers better features than the command line tool. Another challenge in Avahi deployment was incurred while attempting to insert a timer function in the Avahi source code. Since the Avahi code is huge and distributed over a large number of packages, extra effort and time were needed to figure out the exact point on timer insertion in the source code. This is crucial in order to accurately measure the performance.

7. Security in Zeroconf Networks

The Zeroconf protocols enable IP-based devices to communicate without any prior configuration or infrastructure services such as Domain Name Service (DNS) or Dynamic Host Configuration Protocol (DHCP). Ensuring that these protocols work securely is a challenging task. In this section, we focus on the security aspects of the Zeroconf technology. We discuss some potential security threats and present an analysis of two possible solutions that can be used to secure Zeroconf networks. Finally, we discuss the results obtained by implementing security in a real Zeroconf network, and its impact on the performance of Mono.Zeroconf, in the Windows XP environment.

7.1 Security Threats in Zeroconf Networks

Devices operating within a Zeroconf network face several security threats such as hijacking, man in the middle attacks, etc. Zeroconf security covers two major aspects:

- ***Confidentiality:*** Confidentiality refers to the security of information, such that, the information transmitted is available only to authorized users and not available to non-authorized users. For example, when a user sends a document to a printer in the Zeroconf network, this document should not be read by any eavesdropper.
- ***Authentication:*** Authentication is the process of proving each other's identity. An example in Zeroconf networks where authentication is required is when a user's machine finds a new printer in the network, but the user needs to verify whether it is really a printer or somebody performing a man in the middle attack.

The vulnerabilities that should be addressed in Zeroconf can be classified into four groups:

- ***Vulnerabilities in Automatic Configuration (IPv4 Link Local Addresses)***: It is possible to locate the DHCP sever before operating in Zeroconf mode so that an unauthorized user may disrupt or disable the normal operation of the Zeroconf protocol. This can make the network susceptible to Address Resolution Protocol (ARP) poisoning. For example, a user may interact with a malicious service because of an ARP poisoning similar to a man in the middle attack. There could be a scenario when an attacker intercepts a confidential document that an authorized user tries to print by impersonating the printer.

- ***Vulnerabilities in Name-to-Address Translation***: Since Multicast DNS (mDNS) uses the DNS services it is vulnerable to DNS poisoning where an attacker can alter the Name/Address table in order to make the user's machine resolve a malicious service.
- ***Vulnerabilities in Service Discovery***: This is an attempt to locate directory agents insecurely allowing an authorized user to disrupt or disable the Zeroconf protocol operation.
- ***Vulnerabilities in Automatic Allocation of Multicast Addresses***: This is an attempt to locate the Multicast Address Dynamic Client Allocation Protocol (MADCAP) server insecurely before operating in Zeroconf mode allowing an attacker to disable or disrupt the normal function of the Zeroconf protocol. Furthermore, transmission of false messages can also occur. A malicious peer can send false Address Space Announce (ASA) message indicating that no addresses are available. This can lead to serious problems including: a. Denial of service after the Multicast Address Allocation Server (MAAS) cannot allocate addresses to authorized users; b. hijacking of addresses since the MAAS, after receiving the false information (ASA message indicating that no addresses are available) will allocate addresses that are already in use by other entities in the domain;

7.2 Security Solutions for Zeroconf Networks

Security solutions for Zeroconf networks can be classified into two broad categories: security via Internet Protocol Security (IPSec), and securing of individual protocols comprising Zeroconf. The first option is to secure all network traffic using IPSec [11]. This solution appears very simple to implement. The second option requires securing the individual protocols used by Zeroconf. This provides a lightweight solution; however, it may require varied security approaches, depending on the protocol involved. We discuss each of these security approaches in detail below.

7.2.1 IPSec for Zeroconf Networks

IPSec [12] provides a set of security services that include access control, data origin authentication and confidentiality. IPSec operates in two modes: a) *Transport Mode* where only the payload (the actual data to transfer) of an IP packet is authenticated and encrypted. This mode is appropriate for host to host communication, b) *Tunnel Mode* where the entire packet (data and header) is encrypted and authenticated. It is suitable for internetwork communication, remote user access and for inter-host communication. IPSec provides traffic security by using three major protocols:

7.2.1 The Internet Key Exchange (IKE and IKEv2) that takes care of negotiation between protocols and algorithms in order to establish the Security Associations (SA) between two sides (host to host or end to end). The IKE also manages the encryption of the data to be transferred and the generation of keys to be used in the authentication phase that is performed by IPSec.

7.2.1.1 The IP Authentication Header (AH) [13] provides connectionless integrity, and preserves information related to the origin of an IP packet. The authentication of IP packets is done by protecting all headers and payloads except for fields that might be altered during transit. It also provides anti-replay services by using the sliding window technique and discarding old packets.

7.2.1.2 The Encapsulating Security Payload (ESP) [14] may guarantee encryption and limited traffic flow confidentiality. It operates at the IP layer using the IP port 50. ESP does not protect the IP packet header unlike AH, even though this can be accomplished in the tunnel mode where the entire IP packet is encapsulated in a new packet and thus is protected by ESP. These protocols can be used in combination, or alone to provide a certain set of security services in IPv4/IPv6. Another important concept is the Security Association (SA) which is the basis for security functions in IP. A SA is a unidirectional connection that works with either AH or ESP to provide authentication and encryption of that particular flow in the connection. It is identified by a Security Parameter Index (SPI), the IP destination address and the security protocol identifier (AH or ESP). If both AH and ESP are used to secure a traffic flow then two or more SAs are needed to reach this goal. Since these security services use the concept of shared secret values characterized by the use of cryptographic keys, IPSec relies on another set of mechanisms to manage these keys. The most famous approach in this context is the use of the public key based approach Internet Key Exchange (IKE) which provides an automatic key management mechanism. As far as the Zeroconf is concerned, the shared group key can be put in place by bootstrapping the IKE protocol which uses the Diffie-Hellman [15] that provides some kind of external authentication to prevent mainly man in the middle attacks. This can be an expensive operation for some resource-constrained end devices that can be found in Zeroconf networks. The problem that arises with the bootstrapping solution is that IKE provides SA negotiation between two entities having unicast addresses. This means that even though IPSec can secure broadcast and multicast traffic there is no way to automatically negotiate the SA. To address this issue, two solutions can be used:

7.2.1.3 Use of Multicast Security (MSEC) [16], a protocol developed by MSEC Working Group, an IETF working group, that provides secure group communication and multicast.

7.2.1.4 Manually configure a shared SA for all devices simultaneously when the group key is configured. Consequently, all senders using multicast will use the same Security Parameters Index field for a Security Association.

7.2.2 Securing Individual Protocols

This approach consists of securing individual protocols composing Zeroconf.

- **Dynamic configuration of IPv4 Link-Local addresses**

The address allocation mechanism in IPv4 Link Local (IPv4 LL), as defined in the Cheshire Internet Draft [8], uses the ARP protocol. IPv4 LL is vulnerable to the ARP weaknesses and also the possibility of disrupting Zeroconf operation if DHCP is discovered before operating in Zeroconf mode due to the flaws of DHCP [17]. This can be overcome as follows:

  - *Use of Secure ARP and Secure DHCP:* as described in [18], Secure ARP uses digital signatures of messages from senders to prevent the injection of spoofed information. Each host has a set of public/private key pair certified by a trusted party on the LAN acting as a Certification Authority. Secure DHCP aims to prevent MAC spoofing by not assigning IP addresses without proper credentials put in evidence by Secure DHCP servers.
  - *ArpOn*: It is open source software that prevents the ARP spoofing attacks by using the kernel space protocols.

- **Name to Address Translation (Multicast DNS)** mDNS has been the standard used to perform Name to Address translation in Zeroconf networks. However, DNS alone cannot guarantee data integrity or authentication. To address this deficiency, DNS has been extended to DNS Secure (DNSSEC) [19]. DNSSEC provides security by using digital signatures that are included in secure zones known as Resource Records (RR). It also allows the storage of authenticated public keys in the DNS.

In Zeroconf networks, digital signatures for RRs can be created by using the public key pair related to a group of devices. However, it is not necessary to include KEY RR in responses because this public key would have already been shared in a particular secure zone. Meanwhile, the disadvantage of using DNSSEC is that it is not really designed to support confidentiality and access control and consequently it cannot authenticate the sender of a DNS request or to prevent passive discovery of device information.

- **Service Discovery**

Service discovery uses the services of DNS (i.e. DNS queries and updates). Then no additional security requirements are needed since DNSSEC, proposed in the previous paragraph, should be able to satisfy most of the requirements.

- **Automatic Allocation of Multicast Addresses**

To mitigate security attacks in a Zeroconf network, the best approach is to perform an authentication test on Address Allocation Protocol (AAP) messages. Since multicast routing protocols cannot prevent an unauthorized entity from sending fraudulent messages or joining multicast groups, therefore the highly recommended solution is to use IPSec, which is capable of providing confidentiality as well as the integrity of the AAP messages.

8. Securing Zeroconf Network: Our Implementation and Analysis

After a careful review and analysis of the possible Zeroconf's security solutions (as discussed above), we conclude that, as Aidan William also recommended in his Internet Draft paper [20], that bootstrapping IPSec appears to be a secure approach to enable security in Zeroconf networks. Most of the techniques that attempt to secure the candidate protocols of Zeroconf provide authentication but not confidentiality which is also an important criteria that has to be taken into consideration. Moreover, IPSec not only provides both confidentiality and authentication but also reduces configuration overhead. To illustrate the choice of the solution, figure 12 shows the protocol stack of Zeroconf with added security with the TCP/IP reference model.

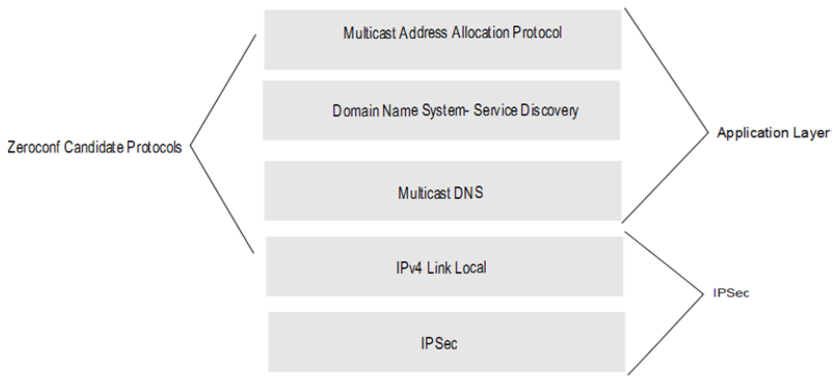
Multicast Address Allocation Protocol
Domain Name System- Service Discovery
Multicast DNS
IPv4 Link Local
IPSec
Zeroconf Candidate Protocols
Application Layer
IPSec

Figure 12: Protocol Stack of Zeroconf with Security with the TCP/IP Reference Model

8.1. Implementation

In this section we discuss the implementation details of securing a Zeroconf network using IPSec. We present our experiences related to deploying IPSec in a Windows XP Pro SP2 environment. IPSec is part of the Windows operating system and was developed by a Cisco-Microsoft collaboration effort. It is integrated in Windows and can be set up through the Microsoft Management Console (MMC) console.

It is worthwhile mentioning that Windows provides some default policies that can enable IP security, but it is always possible to create a customized policy for a complex network scheme that requires several advanced features and special needs. Therefore, it is necessary to activate on one side of the network the Server Policy and on the other side the Client policy. The Server's policy can be based on authentication methods (including a. Active directory via Kerberos v5 protocol, b. Certificates where a third trusted party holds a certificate and serves as authenticator, and c. pre-shared key, which enables two peers holding the same key to establish a secured connection. In our implementation we have used option c (pre-shared key) for simplicity, tunnel setting (we have not used tunnel in our implementation). There are several options for setting the "connection type". Some of the possibilities include "all network connections", or "LAN connections" or "a remote computer". We have employed the LAN connections option in our Zeroconf deployment. The filter list defines the filter that can be applied to the traffic. The most common choices are the Internet Protocol (IP) filter and the Internet Control Message Protocol (ICMP) filter. The filters can also be customized to allow or to block specific traffic types. The Filter Action option corresponds to the list of filters defined under filter lists. In this implementation, we employed the IP filter.

On the client side, the only rule that is defined is the default response policy. This is done to allow the client to respond to the request of the server, and to establish the Security Associations (SAs). The client should also be configured to match the pre-shared key defined in the server. After setting up the server and client policies we verified the security configurations with the IP Security Monitor tool (a traffic sniffer such as Microsoft Network Monitor can also be used to reveal exchanged of packets) which enables the user to verify the settings of the security policy, and also displays the two-phase operations of the IKE protocol.

8.2. Impact of IPSec on the Performance of Mono.zeroconf

We evaluated the impact of IPSec on the Mono.zeroconf implementation. We used the network testbed shown in figure 13 composed of two XP machines connected with a crossover cable and having one iTunes.

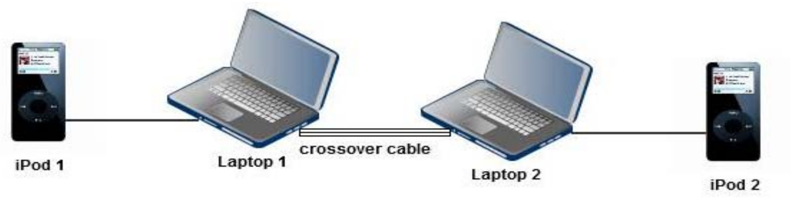

Figure 13: Zeroconf network testbed with IPSec Enabled

Device	No. of Devices	Average Discovery Time (ms)
Workstations discovery	1	256
	2	521
iTunes discovery	1	246.6
	2	256

Devices	No. of Services	Average Service Discovery time (ms) (With IPSec)	Average Service Discovery time (ms) (Without IPSec)	Overhead
Workstations	1 workstation service	463.5	256	81%
	2 workstations services	579.25	521	11.2%
iPod	1 iPod service (iTunes)	452	246.6	83%
	2 iPod services (iTunes)	411.5	256	60.7%

Table 8: Performance impact of IPSec on Service Discovery time

Table 8 shows the service discovery times measured (in ms) to discover various devices (workstations and iPods), along with the percentage increase in service discovery times when IPSec is used. From the results shown in table 8, we conclude that the deployment of IPSec on the Zeroconf network has a significant effect on the discovery time of both, Workstations and iTunes. When IPSec is used with Mono.Zeroconf (on Windows XP), the performance (in terms of discovery service times) is affected and introduces an overhead of about 81% (in the case of single workstation discovery). This is in contrast to Linux-Avahi implementations which use built-in security support to thwart threats (for example, a recent version 0.6.24 has addressed the buffer overflow vulnerability) and is therefore expected to deliver better performance.

9. Conclusion

In this work, we have presented an empirical evaluation of two Zeroconf implementations via performance tests carried out on a network testbed consisting of various devices such as laptops, workstations, printers, iPods, etc. We used service discovery time as the performance metric in all our measurement tests. Our performance results have revealed that the Avahi implementation of Zeroconf on Linux allows much faster discovery of services on a network as compared to the Mono.Zeroconf implementation running on the Windows platform. In fact, the discovery time recorded using Avahi is approximately half of the discovery time obtained with Mono.Zeroconf. We extended our Mono.Zeroconf implementation to support security and compared the performance of security- enabled and non-security enabled Mono.Zeroconf implementations. Two approaches for securing Zeroconf networks were discussed. Such approaches included either securing network level communications or securing each of the Zeroconf candidate protocols. The first choice was more efficient since it requires the implementation of a single protocol (such as IPSec) to obtain end-to-end security. This is easier to deploy compared to using many protocols that can be lightweight but difficult to manage and synchronize. Our performance results show that the service discovery time when IPSec is used with Mono.Zeroconf increases by almost 45% compared to when security is not used.

References

[1] Aidan Williams, "Requirements for Automatic Configuration of IP Hosts", Internet Draft, September 2002 available at http://files.zeroconf.org/draft-ietf-zeroconf-reqts-12.txt

[2] Ping Dong, Hongke Zhang, Hongbin Luo, Ting-Yun Chi, Sy-Yen Kuo, "A network-based mobility management scheme for future Internet", Computers & Electrical Engineering, Volume 36, Issue 2, March 2010, Pages 291-302.

[3] Dittrich, A.; Salfner, F., "Experimental responsiveness evaluation of decentralized service discovery", IEEE International Symposium on Parallel and Distributed Processing, 2010, Pages 1 – 7.

[4] E. Guttman, “Autoconfiguration for IP Networking,” June 2001, available at http://www.Zeroconf.org/w3onwire-Zeroconf.pdf

[5] Edgar Danielyan, “The Internet Journal”, December 2002, volume 5, p. 20-26.

[6] “Avahi Package”, http://doc.pfsense.org/index.php/Avahi_package

[7] “Windows CE Wireless Zero Configuration Service”, http://msdn.microsoft.com

[8] Stuart Cheshire, "Dynamic Configuration of IPv4 link-local addresses", draft-ietf-zeroconf-ipv4-linklocal-00.txt, 2005.

[9] “Package: avahi-daemon (0.6.29-1 and others)”, http://packages.debian.org/sid/avahi-daemon

[10] Mono.Zeroconf Home Page available at http://www.mono-project.com/Mono.Zeroconf

[11] Drago Žagar, Krešimir Grgić, Snježana Rimac-Drlje, “Security aspects in IPv6 networks – implementation and testing”, Computers & Electrical Engineering, Volume 33, Issues 5-6, September-November 2007, Pages 425-437.

[12] Kent and Atkinson, “Security Architecture for the Internet Protocol” RFC2401, November 1998, available at http://rfc.sunsite.dk/rfc/rfc2401.html

[13] Kent, S., and R. Atkinson, "IP Authentication Header", RFC2402, November 1998, http://www.ietf.org/rfc/rfc2402.txt

[14] Kent, S., and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[15] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999

MSEC Working Group Home Page, available at http://www.securemulticast.org/msec-index.htm

[16] Hao Yin, Feng Qiu, Chuang Lin, Geyong Min, Xiaowen Chu, “A novel key-embedded scheme for secure video multicast systems”, Computers & Electrical Engineering, Volume 32, Issue 5, September 2006, Pages 376-393.

[17] “DHCP Flaws”, http://www.cert.org/advisories/

[18] Biju Isaac, “Secure ARP and Secure DHCP protocols to mitigate security attacks”, International Journal of Network Security, Vol.8, No.2, pages: 107- 118, Mar. 2009, available at http://ijns.femto.com.tw/contents/ijns-v8-n2/ijns-2009-v8-n2-p107-118.pdf

[19] Eastlake and Kaufman, “Domain Name Security Extension”, RFC2535, January 1999 available at http://www.faqs.org/rfcs/rfc2065.html

[20] Aidan William, "Securing Zeroconf Networks", IETF Internet Draft, November 2000, http://www.ietf.org/proceedings/49/slides/ZEROCONF-3/index.htm

APPENDIX- DEFINITION OF TERMS

AFP: Apple File Sharing

AH: Authentication Header

ASA: Address Space Announcement

DHCP: Dynamic Host Configuration Protocol

DNS: Domain Name System

EAP: Authentication Protocol

HomePNA: Home Phonewire Network Alliance

IETF: Internet Engineering Task Force.

IP: Internet Protocol

IPP: Internet Printing Protocol

ISP: Internet Service Provider.

JRE: Java Run time Environment

LAN: Local Area Network

MADCAP: Multicast Address Dynamic Client Allocation Protocol

mDNS: Multicast Domain Name Service

MMC: Microsoft Management Console

NIM: Network Installation Manager

RFC: Request for Comments

SA: Security Association

SLP: Service Location Protocol

WEP: Wired Equivalent Privacy

ZMAAP: Zeroconf Multicast Address Allocation Protocol

Farhan Siddiqui received a B.E. (2000) degree in Computer Science from Osmania University, India, and M.S. (2003) and Ph.D. (2007) degrees in Computer Science from Wayne State University, MI, USA. She was a full-time

faculty member at Bradley University, IL, USA during Fall 2008. Since March 2009, she is a faculty member in the School of Information Systems and Technology at Walden University, MN, USA. Her research interests are in Mobile and Ubiquitous Computing, Home Networking, Voice over IP, Security, Quality of Service, and Cloud Computing.

Sherali Zeadally received his bachelor’s degree in Computer Science from the University of Cambridge, England and his doctoral degree in Computer Science from the University of Buckingham, England. He is an Associate Professor at the University of the District of Columbia, Washington DC. He has served as a Guest Editor for over 20 special issues of various peer-reviewed scholarly journals. His research interests include computer networks including wired/wireless networks, network/system/cyber security, mobile computing, ubiquitous computing, multimedia, performance evaluation of systems and networks.

Thabet Kacem is currently a PhD student in Information Technology, with concentration in Software Engineering, at George Mason University. He obtained his Master degree in Computer Science at the University of the District of Columbia in 2010 after receiving his Bachelor degree in Computer Science at the National School of Computer Science at the University of Manouba in Tunisia in 2007. His research interests are software architecture and design, ubiquitous computing, mobile and pervasive software systems, and security.

Scott Fowler received a B.Sc. (1998) from Minot State University, ND, U.S., M.Sc. (2001) from the University of North Dakota, N.D., U.S. and Ph.D. (2006) from Wayne State University, MI, U.S., all degrees are in Computer Science. He was a Research Fellow in the Adaptive Communications Networks Research Group at Aston University, UK from 2006 to 2010. Since 2010 he has been an Associate Professor at Linköping University, Sweden. His research has been funded and supported by EU Framework 7, ELLIIT, Ericsson and Ascom. Scott Fowler was a host for a Fulbright Specialist from the USA in 2011.