



Zero Configuration Networking with Symbian OS™

White Paper - June 2007

Table of Contents

1. Introduction.....	3
1.1.Networking as simple as plugging in a cable.....	3
2. The Zeroconf technology.....	5
2.1.Addressing without DHCP.....	5
2.2.Name resolution without DNS.....	7
2.3.Service discovery.....	8
3. Benefits of deploying Zeroconf.....	10
4. Zeroconf adoption in other operating systems.....	11
5. Zeroconf in Symbian OS.....	12
6. Supporting Zeroconf with Resolv.....	13
6.1.Creating a Zeroconf server session.....	13
6.2.Resolving hosts and services.....	14
6.3.Asynchronous notification of events.....	16
6.4.The hints API.....	17
6.5.Advertising a service.....	18
7. Further reading.....	19

1. Introduction

The march of progress means that today we have more computing power in our pockets than could be fit in to an entire room only three decades ago. Mobile technology has changed the way we think about our lives. If we are lost, we retrieve a map or if we're bored we can play a game with our friends, no matter where they are. Yet, as wonderful as this new technology is, often usability is left behind in the rush to implement the latest and greatest feature.

One such area where the pace of change has left usability languishing is networking and connectivity. The development of IP networking has seen a number of usability improvements, for example friendly machine naming with DNS, but all the codified and widely adopted standards are centred around fixed infrastructure networks where machines are by and large not portable. In the mean time, Internet Protocol (IP) networking has moved on. Increasingly machines talk to each other without wires, and roam from place to place, network to network. The protocols that made networking easy in the past simply don't work in this environment, and the fallout is that one again end users need to know about such esoteric subjects as what an IP address is and how to use a subnet mask.

The fixed Internet has more or less standardised on using IP for all communications between end nodes, but the usability gulf in the mobile space has allowed other communications protocols to flourish. Technologies such as Bluetooth® provide friendly names for devices, and easy service discovery. Yet, as an unintended consequence, the proliferation of communications technologies means that we are once again in the situation where two different devices only metres from each other can't communicate because they do not share a common 'language.'

1.1. *Networking as simple as plugging in a cable*

What is needed for mobile devices is to get to the same level of usability that you might enjoy in any modern office. You put your laptop on the desk, connect an ethernet cable and you are on the network. In an office environment, that is achieved by utilising centralised infrastructure. The company will have a DHCP server to allocate you an IP address, and DNS servers so you can talk to the services you need using names rather than numbers. But how is this to be achieved in mobile devices, where the networks you want to form are often ad hoc with no centralised infrastructure?

One solution is *zero configuration networking*. Using Zero Configuration Networking (or Zeroconf, for short) a device assigns itself an address and human readable name, avoiding clashes by negotiating with its peers. Zeroconf also enables a device to advertise the services it offers to the network, and to browse for services that other devices are advertising.

Finding a printer no longer involves asking someone where the printer is, what its name is and what model it is. The printer itself advertises those details, and any device that wants to print can simply browse for the printing service and offer a list of printers to the user. Advertised services can also be more personal; for example software such as the Pidgin instant messaging client allows users to find people to chat with on their local network and Apple's iTunes® allows users to share libraries of music with one another.

Using this software doesn't require any specialist knowledge. The device connects to the network automatically and specialised applications present lists of network services they can use. Literally, there is zero configuration involved. The end user is insulated from the complexity of the network, all they will see are friendly service names that they can utilise. As we will see, it is also trivial to make your own networking application as easy to use as this using only a few lines of code.

2. The Zeroconf technology

Zeroconf is an umbrella term for a number of underlying technologies. First amongst these is the ability to acquire an IP address without the aid of a central server or static configuration. Obviously, the device must be addressable as a prerequisite for it to be able to use the network.

Layered on top of addressing Zeroconf also provides a protocol for human readable naming of devices and the advertisement and discovery of services, enabling networked applications to present friendly user interfaces to the user.

These protocol layers are not dependent on one another. For example, it is possible to deploy Zeroconf naming and service discovery on a network that does not use the Zeroconf addressing protocol.

2.1. Addressing without DHCP

With broadband connections so commonplace, there are more and more small private networks in people's homes and small offices. In general, these will be using Dynamic Host Configuration Protocol (DHCP) to assign addresses to clients, identify name servers and identify routes to the rest of the network. This is made possible because these networks are centralised, using either a dedicated router or a PC acting as a router as the hub of their network.

However, not everywhere that you want to connect devices has infrastructure this sort of infrastructure available. Imagine you wanted to share a photo between two devices over lunch in a coffee shop. Our devices must have addresses before they can run any higher level protocol to transfer the file.

Zeroconf solves this problem with IPv4 Link-Local Addressing (IPv4LL) [1] an IETF standard, widely supported protocol. IPv4 reserves a range of addresses (169.254.x.x) for link-local use. A host that wishes to join the network selects an address at random from this range and queries the network to check if it is currently in use or not. To perform these queries, we use the Address Resolution Protocol (ARP) to issue three sequential '*who has?*' requests at two second intervals. If none of these requests is answered we can reasonably assume the address we have selected is not currently in use and may be claimed as our own. If our requests are answered by a host that is currently using the selected address, we simply select another and repeat the process. Provided the selection method is sufficiently random conflicts, are very unlikely considering that there are 2^{16} possible addresses to choose from, and ad hoc networks seldom grow to as many as one hundred machines.

1 RFC 3927, Dynamic Configuration of IPv4 Link-Local Addresses, Cheshire et al.

Typically, networks formed in this manner will be using some form of broadcast wireless technology rather than physical cables to connect them together. What we refer to as an ad hoc network is formed when a number of devices come in to broadcast range of each other. Zeroconf ensures that rather than being isolated, these devices will have an address that they can use to communicate amongst themselves on a single link. Because the network is broadcast, new hosts may discover and join the network at any time. Similarly, hosts may move out of broadcast range and leave the network, or whole groups of hosts may split off and form their own network that is isolated from the parent. Networks of this type can vary in size from our simple two device network in our coffee shop example, to hundreds of hosts in the audience at a trade show presentation forming a collective back channel to discuss the presentation.

Most people encounter these link-local addresses in the context of a failure. Their machine has failed to connect to the network and get an address from the DHCP server, and so it falls back to a 169.254 address. Without DHCP to supply them with a default route and a set of name servers, they come to see this network connection as useless. However, in an environment where machines are prepared to function correctly in an ad hoc network, these addresses are perfectly adequate for communication to take place.

This approach to allocating addresses has some obvious drawbacks though. It offers no route to the larger internet by default, even if one or more of the machines in the network has access to the internet and is willing to share it. In addition, because no network transport is 100% reliable, there is always the chance of an addressing conflict despite the initial probing. This can occur if your probes are lost, or if another host is not obeying the protocol or is even acting maliciously to deny you an address. The protocol specifies a number of steps to attempt to resolve accidental conflicts, but the protocol is not resistant to deliberate denial of service attacks.

The IPv4LL protocol is implemented on most desktop operating systems and other networked devices. Microsoft refers to this protocol as Automatic Private IP Addressing (APIPA) or in older releases of their operating system, IP Automatic Configuration (IPAC.)

2.2. Name resolution without DNS

Much of the success of the world wide web is comes from being able to refer to hosts by a friendly name rather by its IP address. Far easier to tell a friend about www.google.com than it is to tell them to visit 66.102.9.147. With mobile devices, the network we are connected to can change and, with it, the IP address that the device is assigned. With naming, the name always remains the same even as the underlying address is changing. This is especially important where we're using link-local addressing, where conflicts may cause our address to change at any time.

Of course, with the centralised naming system you have to register the name you want to use, set up authoritative DNS servers and wait for any changes to propagate around the internet. For ad hoc networks we need something a little more agile, so for local naming we use the pseudo top level domain name .local to avoid the registration requirement. Link local names present few of the restrictions that DNS has inbuilt, names are formatted in UTF-8 and have no forbidden characters.

Since we have no DHCP to hand our hosts name server addresses, we have no central authority to refer to to resolve names to addresses. So each host must run its own name server to advertise its own name. The protocol used by Zeroconf is called Multicast DNS (mDNS) which, as the name suggests, is derived from the traditional unicast DNS system but run over link-local multicast. Each host wishing to use the Zeroconf naming mechanism will join the IPv4 multicast group 224.0.0.251 and listen on UDP port 5353 for incoming messages.

As when claiming an addresses, each host will choose its own name and send a series of three requests on the multicast DNS address. If an answer is received, the host must choose another name and retry. Multicast DNS uses a number of mechanisms to keep the number of packets transmitted to a minimum. A well behaved host limits itself to one packet transmitted per 250ms, and because packets are transmitted via multicast, hosts can receive and cache responses to queries transmitted by other hosts.

Many of the same security considerations that apply to IPv4LL also apply to mDNS. A malicious host can usurp another's host name, or deny another host a name in the first place. Names assigned to hosts using mDNS should be treated as helpful shorthand, rather than authoritative.

2.3. Service discovery

In addition to addressing and naming services that are commonplace in fixed infrastructure networks, using ad hoc networks presents new requirements to make them usable. In networks that are constantly changing, it's important to know what services are available to use at the present moment. Zeroconf enables this by extending its multicast DNS facility by using DNS service records. [2] A host running a mDNS responder can advertise services, browse for a specific service or find all service types that exist on the network.

A service's name is made up of a friendly, human readable name for the service followed by the service type, the protocol it runs over and finally the .local top level domain name. For example, an HTTP server might be advertised as 'My Webserver._http._tcp.local' meaning that the site 'My Webserver' is an HTTP service running over TCP on the local link. By convention, service and protocol names begin with an underscore to differentiate them from other names. The DNS service record includes the name of the host on which the service is running and port number at which it can be accessed. In addition, an mDNS responder advertising a service will also transmit a text record for the same name which may contain a number of key-value pairs that act as parameters for the service. Our HTTP service, for example, could send 'path=/index.html' to hint the default URL the client should connect to.

Since in our ad hoc network, hosts can join or leave at any time it is important to have a mechanism to keep a host's cache of current services available on the network as fresh as possible. When a host starts advertising a service, it will announce it to the network by sending an unsolicited query response for their service. When the host ceases to offer the service, it will send another unsolicited response designed to flush the service record from the caches of the other hosts on the network. In terms of the user interface, this means applications are notified almost instantly of new services being available, users don't have to click 'refresh' to see new service instances as they appear on the network. Stale entries are usually removed before users attempt to use them, avoiding poor user experiences.

A host advertising a service will have a service and text record under its own unique name as illustrated above, and in addition a pointer record in the name of the service that points to the specific service instance. With our HTTP example, not only would the pointer record 'My Webserver._http._tcp.local' exist, but also a pointer record for “_http._tcp.local” pointing to it. Potentially, a host may know about any number of such pointer records, one for each service that exists in the system. These pointers enable clients to perform queries for all instances of a specific service that exist on the network by asking for all pointer records for the service name. Zeroconf also has one additional level of indirection defined, the name '_services._dns-sd._udp.local' is reserved for pointer records to specific services, enabling clients to browse for all service types that exist on the network.

2 RFC 2782, A DNS RR for specifying the location of services (DNS SRV), Gulbrandsen et al.

As with the Multicast DNS protocol that underlies Zeroconf's service discovery mechanism, steps have been taken to avoid 'chattiness' or excess transmissions. Clients are encouraged to include unsolicited detail in the replies they transmit, so if our host received a request for all pointer records for the name '_http._tcp.local', not only would the host reply that it knows that '_http._tcp.local' points to 'My Webserver._http._tcp.local' but it would also include the service record for that service, and the address record for the host that service record pointed to, avoiding two potential follow up queries. Hosts querying the network are encouraged to use 'known answer suppression', so if a host knew about 'My Webserver._http._tcp.local' but still wished to find more HTTP services, it would list the answer it already knew in its query packet to avoid getting the answer repeated.

3. Benefits of deploying Zeroconf

As previously noted, zero configuration networking is all about providing real plug and play usability for applications and devices that need to talk to one another. Customers purchasing software for use on a mobile handset will expect a 'consumer electronics' experience. It should not be expected that they know about such concepts as an IP address, or necessarily even how to connect their device to a network they already own.

Using Zeroconf, networked applications can provide a friendly networking experience to the user. If you are designing a networked game, the interface can present a list of people in the local area who are waiting to play for the user to select one, rather than asking them to find out what one another's IP addresses are, and to type them on a limited keypad. Input is kept to minimal use of the direction keys, and the complexity of networking is neatly abstracted away. This should help cut down on support requests from inexperienced users, reducing support costs and increasing customer satisfactions.

From a technical perspective, deploying Zeroconf also means you don't have to constantly reinvent the wheel. Many current generation networked applications employ a degree of service discovery to get the kind of usability described above. If, instead of deploying a proprietary service discovery mechanism, you choose to deploy Zeroconf you get a reliable platform on which to build. This avoids the burden of testing your own proprietary solution, and has the additional benefit of allowing you to share the service discovery code in common between all your applications for a net reduction in disk space taken up, and an increase in device battery life. Development time is also reduced, since the amount of code needed to incorporate Zeroconf in to your application is minimal.

Standardising on Zeroconf also means other people's applications can talk to you. If your application is employing an open protocol, and advertises it using Zeroconf, then applications supplied by other vendors can also detect and utilise the service. While you may not be willing or able to cater for every esoteric use case your customers have, it is entirely possible that a third party will. By using a standardised service discovery protocol, you'll often be surprised to find your users using your application in most unexpected ways. As an example, Apple's iTunes® advertises a music streaming protocol using Zeroconf. One enterprising set of graduate students adapted this protocol to gather statistics about music listening behaviour on their university campus. The more flexible your application is, the more users you can potentially satisfy.

4. Zeroconf adoption in other operating systems

As stated previously, the address assignment portion of the Zeroconf suite has seen widespread adoption and is now included in the vast majority of popular operating systems. Indeed, RFC 3927 that specifies IPv4LL is simply a formalisation of the best practices that were previously seen in operating systems as far back as 1998.

Multicast DNS and DNS Service Discovery, while enjoying widespread support and adoption, are still not as universally available as IPv4LL. These protocols were developed and championed by Apple as a replacement for similar functionality that appeared in their AppleTalk[®] networking protocol suite. Today, the Apple's OS X[®] operating system has the full suite of Zeroconf functionality enabled by default, and enjoys a range of applications utilising the protocol. These applications include iTunes[®] which uses service discovery to find music available on the local network, and SubEthaEdit which allows users to collaborate on authoring documents over the network.

Apple's multicast DNS and DNS service discovery implementation which goes by the trade name Bonjour[®] (previously Rendezvous, but this was changed due to a trademark dispute) is released as open source software under the Apache licence, and has a maintained port to Microsoft Windows available to download from Apple's site, or is installed with the Windows version of iTunes. However, Microsoft does not ship these protocols on their system by default and does not produce their own implementation.

A third party, open source implementation of these protocols has been produced called Avahi. This version, originally for Linux, has been ported to a number of other Unix-like operating systems, including Mac OS X. It offers an interface which is roughly source compatible with Apple's implementation in addition to the preferred interface via d-bus. This implementation is part of the Gnome environment's mobile platform and can be used in a number of portable devices running Linux, such as Nokia's N800 web tablet.

In addition to the general purpose operating systems now running the Zeroconf protocols, many dedicated hardware devices now implement it to advertise the services they provide. Almost every networked printer on the market now uses Zeroconf networking.

5. Zeroconf in Symbian OS

Symbian OS devices ship with inbuilt support for the addressing protocol IPv4LL, but have no built in support for multicast DNS or DNS service discovery. However, Novel Interactions ships the Resolvr Zeroconf framework to correct this deficiency.

Resolvr is designed from the ground up for mobile devices, having a small memory footprint and using very little CPU time. The framework is dual licensed under the GNU GPL open source licence and a commercial, royalty free licence for users for whom the requirements of the GPL are inappropriate.

Resolvr is thoroughly entrenched in the Symbian style of programming, with a simple interface that makes it trivial to add Zeroconf functionality to an application. Using it, adding Zeroconf functionality to a program can take as little as a day, including building an appropriate user interface. It supports all Symbian OS releases from v9 onwards, and is compatible with both S60 and UIQ.

6. Supporting Zeroconf with Resolv

To get started using Resolv simply download the SDK from <http://www.novelinteractions.com/resolv> and follow the instructions to unpack it in your SDK directory. Your application will need to link against the client DLL, 'zeroconfclient_20008BAC.lib' and the Network Services capability is required to use the functionality it provides. All the API calls are enclosed in the 'Zeroconf' namespace.

6.1. Creating a Zeroconf server session

The main class needed to use Zeroconf functionality is *RZeroconfSession* defined in zeroconfclient.h. To create a session with the Zeroconf server, you must specify which Internet Access Point (IAP) you wish to run Zeroconf over. Typically, you will get this information from the RConnection class. The code to connect to the Zeroconf server typically looks like this:

```
#include <zeroconf/zeroconfclient.h>
#include <es_sock.h>

RSockServ socketServer;
User::LeaveIfError(socketServer.Connect());
CleanupClosePushL(socketServer);

/* create a connection to an IAP */
RConnection conn;
User::LeaveIfError(conn.Open(socketServer));
CleanupClosePushL(conn);

/* Fetch the IAP number of the connection we just created */
TUint32 iap(0);
_LIT(KCommdbIapSetting, "IAP\\Id");
User::LeaveIfError(conn.GetIntSetting(KCommdbIapSetting, iap));

/* Open a session with the zeroconf server */
Zeroconf::RZeroconfSession zeroconfSession;
zeroconfSession.OpenL(iap);
CleanupClosePushL(zeroconfSession);
```

The Zeroconf server will assign itself a network name based on the device's current Bluetooth name. If Bluetooth is off, or the name can't be retrieved for other reasons then a default name will be assigned to the device.

6.2. Resolving hosts and services

Once you have an open connection to the Zeroconf server, you can use it to browse for services, resolve hostnames and list the types of services available on the network. The following code example shows you how to fetch the list of currently known service types:

```
#include <zeroconf/zeroconfclient.h>
#include <zeroconf/servicetype.h>

RPointerArray<Zeroconf::CServiceType> serviceTypes;
cleanupResetAndDestroyPushL(serviceTypes);
zeroconfSession.ServiceTypesL(serviceTypes);
```

On return, the array *serviceTypes* is populated with a list of *CServiceType* objects, one for every service type the system is currently aware of. The public interface of the *CServiceType* object enables you to retrieve the name and protocol of the service (e.g., '_http' and TCP respectively) in addition to the fully qualified name of the service.

Service resolution is accomplished as follows:

```
#include <zeroconf/zeroconfclient.h>
#include <zeroconf/zeroconfservice.h>

/* The service type we wish to browse for, including leading underscore */
_LIT(KHTTPServiceType, "_http");

/* The name of the service we want to resolve */
_LIT(KHTTPServiceName, "My webserver");

CService* service = iSession.ResolveServiceL( KHTTPServiceName, KHTTPServiceType,
                                             Zeroconf::EServiceProtocolTcp);
```

The *CService* object interface allows access to the human readable service name, the name of the service it's an instance of, the hostname and port it's running on and the protocol it uses. It also provides the list of parameters as key-value pairs. The IP address of the host the service is running on can be retrieved by using the host name resolution functionality.

Host name resolution is accomplished as shown in the following example:

```
#include <zeroconf/zeroconfclient.h>
#include <insock.h>

/* The host we wish to resolve */
_LIT(KHostName, "My Hostname");

TInetAddr address;
TInt err = zeroconfSession.Resolve(KHostName, address);
if (KErrNone == err)
    {
        /*.... */
    }
```

Unlike the service type browsing function, service and host resolution may take a short period of time to complete. Due to this fact a second, asynchronous overloads of the *Resolve* and *ResolveService* functions is provided so applications can maintain their UI responsiveness. If the host is not found, the *Resolve* function will return *KErrNotFound*.

6.3. Asynchronous notification of events

We have already seen how it is possible to get a list of all the services and service types known to the system at a particular point in time. However, many applications require the ability to keep a constantly updated list of services available, either to present the list to the user, or to utilise those services in the background.

To do this, we can request the server to notify us whenever an interesting event happens. The events the server notifies clients of are host and service records being added to or deleted from the system, or records being updated. The following code demonstrates the use of the notification API:

```
#include <zeroconf/zeroconfclient.h>
#include <zeroconf/zeroconfmessage.h>

class CEventListener : public CActive
{
public:
    /* Call this method to begin receiving messages */
    void Start();

private:
    void DoAwaitMessage();
    void RunL();
    void DoCancel();

private:
    zeroconf::RZeroconfSession iZeroconfSession;
};

void CEventListener::Start()
{
    /* before we can receive any messages, we must let the framework know we want them */
    iZeroconfSession.ReceiveMessagesL(ETrue);
    DoAwaitMessage();
}

void CEventListener::DoAwaitMessage()
{
    /* This call completes when the server has a new message for us */
    iZeroconfSession.MessageNotify(iStatus);
    SetActive();
}
```

```

void CEventListener::RunL()
{
    User::LeaveIfError(iStatus.Int());

    Zeroconf::TZeroconfMessage message;
    iZeroconfSession.NextMessageL(message);

    /* example: watching for new HTTP services */
    _LIT8(KHTTPServiceFQN, "_http._tcp.local");

    if (EZeroconfMessageStatusNew == message.Status() &&
        0 ==message.Name().CompareF(KHTTPServiceFQN))
    {
        /* .... */
    }

    DoAwaitMessage();
}

void CEventListener::DoCancel()
{
    iZeroconfSession.MessageNotifyCancel();
}

```

For optimal performance, be sure to turn off the reception of messages if your application is no longer handling them.

6.4. The hints API

If your application is especially interested in a single host or service type, you can use the hints API to tell the Zeroconf server to give special treatment to those particular types. You can use the hints API to mean that you want to be notified as quickly as possible when a particular host or service enters or leaves the network, or that you suspect a particular service is no longer on the network but is still cached by the Zeroconf server.

The following code demonstrates using server hints:

```

#include <zeroconf/zeroconfclient.h>

/* Hint that our application is interested in the HTTP service */
_LIT(KHTTPService, "_http");
zeroconfSession.HintServiceL(KHTTPService, Zeroconf::EServiceProtocolTcp);

```

6.5. Advertising a service

The service advertisement API is implemented as a subsession of the main Zeroconf client API. A client wishing to advertise a service should create an instance of the subsession *RZeroconfAdvertisement*, set its name, service type, protocol and port and any parameters you wish to provide. Once the application has finished setting up the advertisement, the asynchronous *Advertise* method should be called. This method completes when the advertisement is up and running, which could potentially take several seconds.

The following example code demonstrates advertising a service:

```
#include <zeroconf/zeroconfclient.h>
#include <zeroconf/zeroconfadvertisement.h>

/* Open a new, blank advertisement */
Zeroconf::RZeroconfAdvertisement advertisement;
advertisement.OpenL(zeroconfSession);
CleanupClosePushL(advertisement);

_LIT(KServiceName, "My Webservice");
_LIT(KServiceType, "_http");
const TUint16 KHttpPort = 80;

/* create an advertisement for an HTTP service */
advertisement.CreateServiceL(KServiceName, KServiceType, Zeroconf::EServiceProtocolTcp, KHttpPort);

/* set a default path attribute for this service */
_LIT(KPathAttribute, "path");
_LIT(KDefaultPath, "/mydir/index.html");
advertisement.SetServiceParameterL(KPathAttribute, KDefaultPath);

/* publish this service advertisement */
advertisement.Advertise(iStatus);
SetActive();
```

It should be noted that while you can change the port the service is running on and the list of parameters that are currently set, it is impossible to change the name, type or protocol of the service after the initial call to *CreateServiceL*. The service continues to be advertised until the *Close* method is called on the subsession object.

7. Further reading

<http://www.novelinteractions.com/resolvr>

The Resolvr product website, here you can learn more about the Resolvr Zeroconf framework and download the Resolvr SDK.

Zero Configuration Networking: The Definitive Guide by Stuart Cheshire.

Published by O'Reilly 2005, ISBN 0-596-10100-7.

This book, written by the inventor of the mDNS and DNS-SD protocols, is both a reference to the protocol itself and the philosophy behind it. It manages to provide a comprehensive technical description of the Zeroconf protocol suite, while at the same time also being entertaining to read. It also contains appendices with details of how use Apple's Zeroconf APIs. A must for any desktop developer wishing to employ Zeroconf.