Device aggregation with data networking

Implementing a Personal Area Network

SHASHA ZHANG

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2008

COS/CCS 2008-23

Device aggregation with data networking

Implementing a Personal Area Network

Shasha Zhang
shasha@**kth.se**

Master's thesis

3 October 2008

Examiner and Supervisor
Prof. Gerald Q. Maguire Jr.

Abstract

Technology is advancing rapidly and intelligent devices are becoming affordable and wireless infrastructure is becoming pervasive. Personal information technology appliances have become part of our life, via cellular phones, PDAs, Bluetooth headsets, handheld keyboards, GPS, and digital cameras. In the future, new intelligent devices will be invented as technology evolves. However, because multiple devices provide similar (but different) functionality it is complex for the average user to choose a single device. Moreover, today it is hard to configure, administer, and use several different appliances together. In order to be utilized in an effective manner and in an accessible way, personal devices should be aggregated, i.e., connected together via a local area network so that they can appear to the user as if they were a single device.

This paper introduces a connection model based on device aggregation to realize shared state, the behavior of a shared appliance, and with the superset of the individual device functionality. Such an aggregated logical device might even exhibit functions which a user would have a very hard time realizing by manually combining devices. This will facilitate the user's control over their appliances (build of different devices), but acting as one device.

The project was a joint effort with David Sabaté Mogica. We developed such a system based on Dynamic Host Client protocol (DHCP) and Service Location Protocol (SLP) for service discovery and Virtual Network Computing (VNC) for remote desktop control. The system builds on a laboratory network environment. This thesis concerns the implementation and evaluation of service discovery. The Remote desktop control was researched and implemented separately and will be reported separately. Service discovery between two computers has been implemented using a custom program developed for a PDA. However, at present the PDA only sends a DA request packet with DHCP. However, service discovery has been successfully tested between two computers. This provided an important base for the programming on PDA and the future development of a similar program for a cellular phone.

Keywords: PDA, SLP, network, remote desktop

Sammanfattning

Den snabba tekniska utvecklingen ger våra apparater mer och mer intelligens, priset på avancerade produkter är överkomligt och trådlösa infrastrukturer binder samman allt fler produkter.

Tekniska produkter har blivit en del av vår vardag: Mobiltelefon, PDA-er, trådlösa hörlurar och tangentbord med blåtandsradio, GPS och digitalkameror. I takt med den tekniska utvecklingen kommer hela tiden nya intelligenta och kommunicerande produkter.

Man kan hitta liknande funktioner i olika produkter, och det är svårt för den vanlige användaren att välja den optimala produkten. Dessutom har de avancerade produkterna många parametrar att ställa in, och att använda olika produkter tillsammans kräver att användaren är djupt insatt i tekniken. Genom att aggregera, koppla ihop, produkterna i ett lokalt nätverk, kan deras funktioner användas effektivt och göras bättre tillgängliga genom att de för användaren ser ut som om de tillhör en enda produkt.

Denna avhandling introducerar en kommunikationsmodell baserad på produktaggregering genom delade gemensamma tillstånd och reaktioner hos de ingående produkterna, med tillägg av respektive produkters särskilda funktioner. En sådant logiskt produktaggregat kan också fås att utföra funktioner som användaren annars skulle ha väldigt svårt att realisera genom att manuellt kombinera de nödvändiga produktfunktionerna. Det underlättar alltså användningen av systemet (byggt av flera olika produkter), som fungerar som om det vore en enda produkt.

Projektet har genomförts tillsammans med David Sabaté Mogica. Vi har utvecklat ett system för produktaggregering baserat på Dynamic Host Client protocol (DHCP) och Service Location Protocol (SLP) för att identifiera tillgängliga fuktioner och Virtual Network Computing (VNC) för ”remote desktop control”. Systemet är byggt i ett laboratorienät.

Avhandlingen fokuserar på hur identifiering av tillgängliga funktioner och tjänster genomförs och utvärderas. ”Remote desktop control” utvecklades och infördes separat och kommer att rapporteras separat. Ömsesidig identifiering av funktioner mellan två datorer har genomförts med ett program utvecklat speciellt för en PDA. Hittils sänder emellertid PDA’n bara ett DA-frågepaket med DHCP. Den ömsesidiga funktionsidentifieringen är dock testad och fungerar mellan två datorer vilket gav den nödvändiga grunden för programmeringen av PDA’n och för framtida utveckling av liknande program för mobiltelefoner.

Nyckeord: PDA, SLP, nätverk, remote deskto

Table of contents

List of Figures

Glossary

DA	Directory agents
DHCP	Dynamic Host Client protocol
DNS	Domain name server
GPS	Global Position-finding System
ICMP	Internet Control Messages Protocol
IETF	Internet Engineering Task Force
NFS	Network File System
P2P	peer to peer technology
PDA	personal digital assistant
RDC	Remote Desktop connection
RDP	Remote Desktop Protocol
SA	Service agents
SLP	Service Location Protocol
UA	User agents
UPnP	Universal Plug and Play
USB	Universal Serial
VNC	Virtual Network Computing
Wi-Fi	wireless fidelity
WLAN	wireless local area networks

1 Introduction

1.1 Problem Statement

A large variety of smart devices will be available to users as information technology develops in future. To perform a particular task, many small networked computing devices can work together, enabling them to offer to the user the aggregate of their functions. Unfortunately, today users have to care more than one device because each of the devices has generally been designed to operate alone. In addition to a user's personal devices, there are often a number of fixed devices such as speakers, presentation projectors, printers, or keyboards in the local environment. Currently, the large number of devices gives burdens the user rather than benefiting him or her as the difficulty of trying to get the different devices to work with each other is often very significant. Even though some devices are capable of being connected to others, the interaction is very simple to (for example, using Microsoft's ActiveSync to share settings, information, and files with the other connected device) or very complex (for example, connecting to a data projector and adjusting the settings to get the best possible picture for the audience while maintaining the view of the speaker's notes which the speaker wants to see). In some cases the user would like to maintain an interactive session with someone else even though they change their device configuration (for example, changing from a headset to a set of speakers in the room) [20].

1.2 Objectives

Our goal is to propose a means of aggregating multiple devices via a local network infrastructure. This network will utilize protocols operating on different layers, to enable intelligent devices including PCs, cellular phones, PDAs, Bluetooth headsets, GPS receivers, camera, etc. to work together in order to create an intelligent environment (Figure 1) for users. We envision a home or office of the future with this technology, which allows users to switch between devices arbitrarily; just as if using a single device. Obviously, this aggregated device should make it easier to do more tasks than is the current situation when the user has to manually use one device at a time in the correct order to perform the task which they wish to accomplish. This we expect that a user will be able to use the devices at hand to perform the application they want.

Figure 1. Intelligent environment

1.3 Organization of this Thesis

The whole project is doing by David and me. We are developing such a system based on Dynamic Host Client protocol (DHCP) and Service Location Protocol (SLP) for service discovery and Virtual Network Computing (VNC) for remote desktop control. The system builds on a laboratory network environment. This thesis concerns the implementation and evaluation of service discovery. The Remote desktop control was researched and implemented separately and will be reported separately. Service discovery between two computers has been implemented using a custom program developed for a PDA. Following this introduction, Chapter 2 gives a brief description of the concept of device aggregation, and explains the existing technology. Chapter 3 proposes architecture for device aggregation based on data networking and specifies each device and protocol to be used. Chapter 4 tests the implementation and analyzes the test results. The final chapter offers some conclusions and suggests some improvements and future work.

2 Background and Related Work

In this chapter, the background and related work are discussed. These contents provide the necessary background for the reader to understand the later chapters. First we begin by introducing the basic idea of device aggregation. Devices aggregation scenarios show how device aggregation can be exploited in real life. Related work is elaborated considering four different protocol layers. The chapter ends with an examination of remote desktop software and other means of providing an audio or visual interface to an application which is running on another computer. An example of using a cordless headset with a device is shown to illustrate that it is not only visual information and text information, but also other types of media which can be exchanged between devices.

2.1 What is device aggregation?

Before discussing the related work and existing technologies, some crucial terms used in this thesis are defined:

Intelligent or smart device	a device that has its own computing capability. In this paper, intelligent or smart devices will include cellular phone, PDAs, Bluetooth headsets, and GPS receivers.
Device aggregation	means to make multiple devices work together. To understand aggregation architecture, each level is defined in the context of the devices being aggregated.

2.2 Devices aggregation scenarios

To illustrate the advantages of device aggregation from a user's perspective, three scenarios closely related to real life are given.

2.2.1 Scenario 1

John is an employee in a company. He has a PC, a PDA, a cellular phone, camera and a Bluetooth headset. All of these devices are connected to a local network. He is allowed to use any device to authenticate his identity, because all of the networked devices share a common set of configurations and preferences. When John adds some a new device, such as a sensor, stereo headset, or GPS receiver into his existing network, the new device must announce its presence. If this device is accepted as part of John's (personal) local area network, then the device preferences should be shared between these devices, there. John does not need to re-configure each device manually to accommodate the new device.

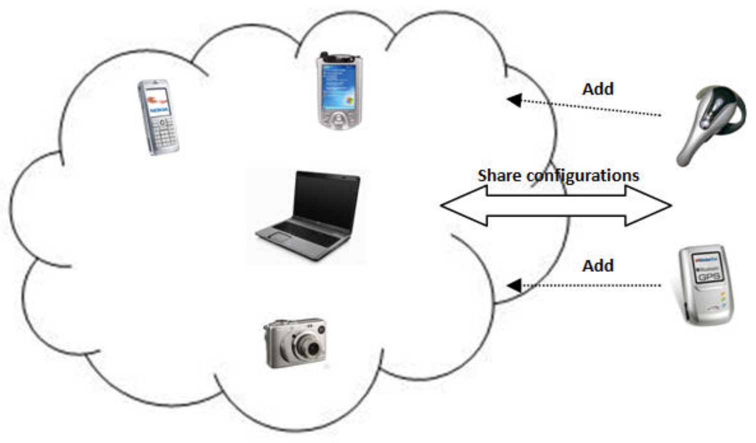


Figure 2. New devices add into local area network

2.2.2 Scenario 2

John uses his cellular phone to connect to the Internet. He is looking at the latest soccer scores with his web browser running on his PDA. As he enters his apartment the PDA detects that he is in the room where his PC is located and asks if John would like to use the large display to view these scores. If John says yes, then the software can use the bigger display of the PC or other display device for display function at hand.

John has just a little work to do in his office, before it is time for his favorite soccer team to start playing. He decides to use a window on his PC’s screen to view the upcoming match - so that he can see just when the game starts. At the same time, he is listening music via his Bluetooth connected headset. He finishes the last of his e-mail, just as the game is about to start. He homes to hear back from Shasha about going to a movie after the game and sends her an e-mail message as he moves into the living room using his PDA. As he moves into the living room the video window with the game follows him into the living room and appears on his large flat panel display. Part way through the match, Shasha’s replay concerning the movie arrives via SMS and because his system knew that he was waiting for it, the message was converted to audio and played via his Bluetooth headset- as well as appearing as a text message on his PDA. From this example, we can see that John’s cellular phone, PDA, and PC shared state and can interact with the smart devices (in this case home appliances such as the flat panel display), therefore John can access data via any of these devices (and appliances). For example, e-mail, instant message, and calls share the same contact lists, call/IM/e-mail…spam filtering can be used on all of these information sources without the user have to configure each one of them,…. Changes made to the black list to block spam are available to the other entire device when John edits and saves changes on one device. He can use any device to edit files, photos, preferences … and continue to work later on any other of his devices.

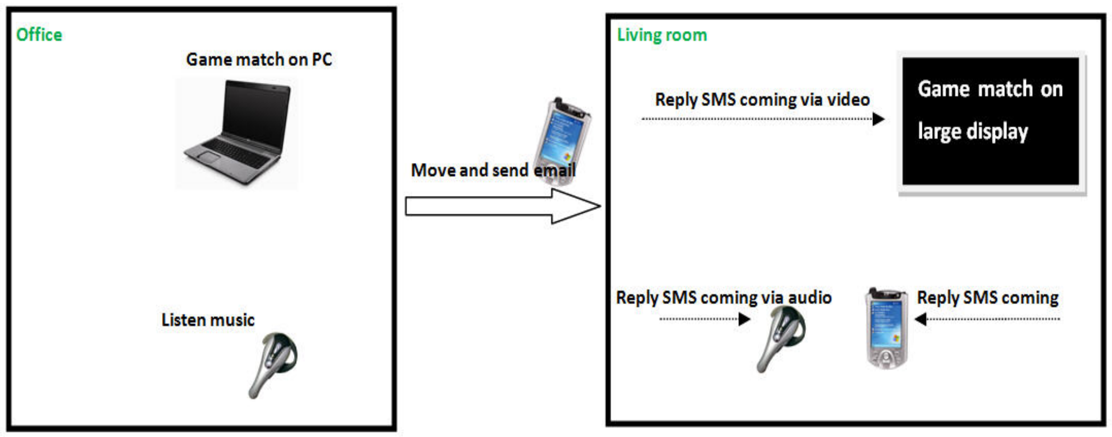


Figure 3. Access appliance from any device

2.2.3 Scenario 3

In addition, when John receives a call from his brother Johan (who is a fan of the opposing team) on his cellular phone, but the phone is not beside him, he can use the keyboard on his PC, a button on his PDA, or interaction with any of the other devices which can accept input from him devices to answer the call; now the call can make use of any of the devices which he has at hand to participate in the call (for example, Johan can use his camera to show Johan the new soccer jack he is wearing for tonight's game – in of course his teams colors).

Mark Wiser described ubiquitous computing as, "invisible, everywhere computing that does not live on a personal device of any sort, but is in the woodwork everywhere" [1].These scenarios show the benefits of device aggregation and how they can indeed simply life (at least viewed from today's perspective).

2.3 Related work and existing technology

Device aggregation can occur at many different system levels. We will explicitly consider some of the forms of aggregation at the link layer, network layer, transport layer, and application layer. This discussion follows the International Organization for Standardization Open Systems Interconnection seven-layer reference model. While it is useful for describing the ideas, we will not be strict in following this model for the actual implementation. Related work is elaborated on each layer as below.

2.3.1 Link layer

Devices aggregation via a local area network or personal area network requires the Link layer to provide low-power, medium to high bandwidth, and always-on connectivity [2]. A number of communication interfaces are commonly found in ubiquitous in many fixed and mobile devices; these include Universal Serial (USB), IEEE 1394a/b (also known as Firewire), Bluetooth, and Wireless Local Area Networks (WLANs).

2.3.1.1 USB

Universal Serial Bus (USB) is a serial bus standard to interface devices and. It was designed to allow many peripherals to be connected to a computer using a single standardized interface socket and to improve the plug-and play capabilities by allowing devices to connect and disconnect without rebooting the computer. The USB 1.0 specification was introduced in November 1995.

Originally USB was intended to replace the multitude of connectors on the back of PCs, as well as to simplify software configuration of communication devices. Other convenient features include providing power to low-power consumption devices without the need for an external power supply and allowing many devices to be used without requiring manufacturer specific, individual device drivers to be installed. USB supports three data rates:

1. Low Speed (1.5Mbit/s) is mostly used for Human Interface Devices (HID) such as keyboards, mice, and joysticks [3].
2. Full Speed (12Mbit/s) was fastest rate before the USB 2.0 specification and many devices fall back to Full Speed. All USB hubs support Full Speed [3].
3. Hi-Speed means 480Mbit/s [3].

2.3.1.2 IEEE 1394a/b

IEEE 1394 (also known as Apple's FireWire, and Sony's i.link) is a high speed serial bus developed by Texas Instruments and Apple computers in the mid-1990s. IEEE 1394 a/b is widely used for multimedia (digital cameras, digital recorders…) and high speed data applications (for example IEEE 1394 external disk drives). IEEE 1394 is a serial bus interface standard that offers isochronous data services and high-speed data communications between digital devices [4].

FireWire is similar to USB, but it supports data transfer rates of up to 400Mbps (in IEEE 1394a) and 800Mbps (IEEE 1394b), compared to USB's 400Mbps. This feature meets high-speed data transfer equipments' need to transfer large quantities of data in real-time (for example, between a DVD player and a PC).

Like USB, it supports both Plug-Play and hot plugging, and provides power to peripheral devices. Additionally, it facilitates peer-to-peer device communications without using PC memory and permits multiple hosts per bus.

When the computer is turned on, it will query all equipment attached and automatically and assign an address to each one. This 64-bit addressing is based on IEEE 1212 standard. Each packet of information sent by a device over the bus consists of three parts:

- 10-bit bus ID: identifies which bus generated the data [4].
- 6-bit physical ID: identifies which device sent the data [4].
- 48-bit storage area: addressing 256 terabytes of information for each node [4].

2.3.1.3 Bluetooth

Bluetooth is a radio interface standard operating in the 2.45GHz frequency band that was designed as a wire replacement for portable electronic devices. Thus its primary goal was to provide short range personal area network, but this war later extended to include multihop-enabling larger ad hoc networks [5]. Bluetooth interfaces are available in several classes, ranging from very short distance ~1m to 10m to 100m. Bluetooth was designed as the successor to the earlier infrared standard, IrDA, in order to provide host-to-host communication as well as USB-like wireless connectivity to peripherals.

Bluetooth technology was initially developed by Ericsson, but also gained the support of Nokia, IBM, Toshiba, Intel, and many other manufacturers. It eliminates the need for wires, cables, and connectors between a cellular phone, PDAs, headsets, printers, projectors, local area networks, and so on. Bluetooth has also tried to foster entirely new applications and devices.

2.3.1.4 Wi-Fi

Wi-Fi (short for “wireless fidelity”) is the trade name for another wireless technology (wireless local area networks- WLANs) created by an organization called the Wi-Fi Alliance. This term is used for wireless local area network based upon the specifications of the 802.11family that meet the organizations requirements for certification for interoperability.

The purpose of Wi-Fi is simple: hide complexity by enabling access to information easily, ensuring compatibility and coexistence, getting rid of cabling and wiring, and getting rid of switches, adapters, plugs and connectors.

A Wi-Fi enabled device such as a PC, PDA, cellular phone, or Media player wirelessly connect to the Internet Wi-Fi access ranges from a hotspot and the coverage of one or more hotspot (with coverage of a limited area) can be an area as small as a single room to metropolitan area coverage of many square kilometers. Wi-Fi access via publicly available hotspots, home and business WLANs, and municipal WLANs -provided free of charge, hourly, or subscription based access [6].Peer-to-Peer mode, also called wireless ad-hoc network mode, enables devices to connect directly with each other.

Compared to low-bandwidth standards, especially Zigbee and Bluetooth, power consumption is fairly high for Wi-Fi, but the range is up to 32m indoors and 95m outdoors using a typical Wi-Fi home router using 802.11b or 802.11g with a stock antenna. However, outdoor range with improved antennas can be several kilometers or more with line-of-sight. Additionally, Alisa Devlic and her colleagues have performed measurements of the actual power consumed when performing file and context distribution for several PDAs and have compared Bluetooth and WLAN – finding that in some cases the total power consumption is actually lower when using WLAN .

2.3.2 Network layer

The network layer provides routing and controls information flow between devices connected devices. The requirement for the network layer for device integration includes:

- Appliance advertising and discovery of devices in the “neighborhood”[2]
- Query other devices’ capabilities[2]
- Easy plug-and-play like addition of new devices[2]
- Self-organizing network [2]

2.3.2.1 Jini

Jini technology is a service oriented architecture that defines a programming model which both exploits and extends Java technology to enable the construction of secure and distributed systems consisting of federations of well-behaved network services and clients [7].

Jini technology consists of an infrastructure and a programming model that address the fundamental issue of how clients connect with each other to form an impromptu community. Jini technology uses the methods pioneered by the Java Remote Method Invocation (RMI) protocols to move objects, including their behavior, around the network. Network services run on top of the Jini software architecture [8].

A disadvantage of Jini is that devices are expected to be able to understand and use the Java procedures which they receive as a result of a query for a service. In many cases this means that the device needs to have a Java VM running to interpret and uses this response.

2.3.2.2 UPnP

Universal Plug and Play (UPnP) is a set of computer network protocols promulgated by the UPnP Forum. The goals of UPnP are to allow devices to connect seamlessly and to simplify the implementation of networks in the home and corporate environments by defining and publishing UPnP device control protocols [9]. UPnP is a distributed, open architecture, based on established standards such as TCP/IP, UDP, HTTP, and XML, that provides compatibility for peer-to-peer network connectivity of appliances, wireless devices, and peripherals.

UPnP works with wired or wireless networks and can be supported on any operating system, any programming languages, and zero-configuration networking. A UPnP device from any vendor can dynamically join a network, obtain an IP address, announce its name, and learn the presence and capabilities of other devices. Devices can also leave a network smoothly and automatically without leaving any unwanted state behind. A Dynamic Host Configuration Protocol client (DHCP) is required for each device since UPnP networking is built on top of IP. When a device first time for the device connects to the network, it will search for a DHCP server and obtains a domain name through a DNS server or via DNS forwarding. The device uses this DNS name or IP address in subsequent network operations.

2.3.2.3 SLP

Service location protocol (SLP) is a service discovery protocol that allows computers and other devices to find services in a local area network without prior configuration [10]. It was originally an *Internet Engineering Task Force* (IETF) standards track protocol that provides a framework to allow networking applications to discover the existence, location, and configuration of networked services in enterprise networks [32]. Traditionally, in order to locate services on the network, users of network applications have been required to supply the host name or network address of the machine that provides a desired service [32].

Protocols that support service location are often taken for granted, mostly because they are already included (without fanfare) in many network operating systems [32]. For example, without Microsoft's SMB service location facilities, "Network Neighborhood" could not discover services available for use on the network and Novell NetWare would be unable to locate eDirectory trees [32]. Nevertheless, an IETF sponsored protocol for service location was not standardized until the advent of SLP [32]. Because it is not tied to a proprietary technology, SLP provides a service location solution that could become extremely important (especially on Unix) platforms. For these details, the reader is referred the following RFCs [32]:

- RFC 2165 - Service Location Protocol, Version 1
- RFC 2608 - Service Location Protocol, Version 2
- RFC 2609 - Service Templates and Service Schemes
- RFC 2614 - An API for Service Location Protocol

SLP can eliminate the need for users to know the names of network hosts. With SLP, the user only needs to know the description of the service he is interested in. Based on this description, SLP is then able to return the URL of the desired service [32].

URL is used to locate the service. SLP is used by service to announce services on a local network [10]. Any network service may be encoded in a Service URL. A service URL example is: “service:printer:lpr://myprinter/myqueue”. The “service:printer: lpr” is called service type. “myqueue” is described one queue on a printer with the host name “myprinter”. Service URL syntax and semantics are defined in RFC 2609.

SLP has three different roles for devices. A device can also have two or all three roles at the same time.

- User agent (UA): A process working on the user’s behalf to establish contact with some service. The UA retrieves service information from the Service Agents or Directory Agents [11].
- Service agents (SA): a process working on the behalf of one or more services to advertise the services [11].
- Directory agents (DA): a process which collects service advertisements. There can only be one DA present per given host.

As figure 4 described, the service Location Protocol supports a framework by which client applications are modeled as “User Agents” and services are advertised by “Service Agents”. “Directory Agent” provides scalability to the protocol [11].

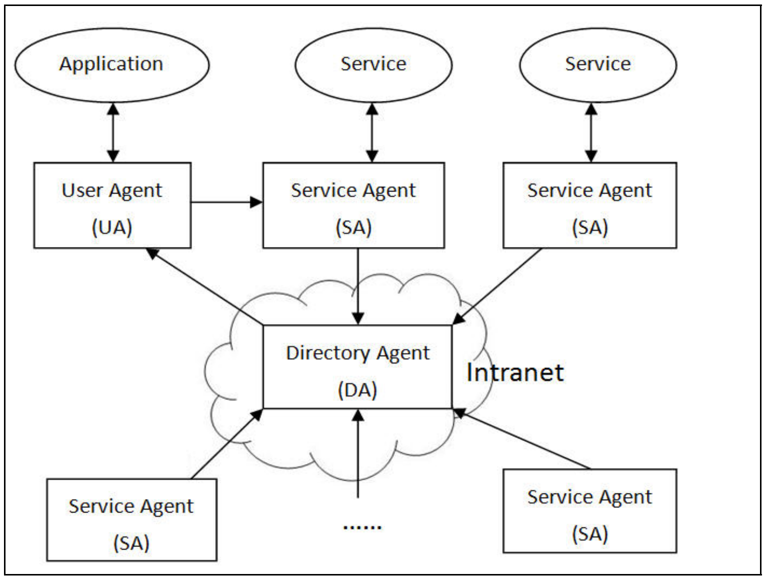


Figure 4. Service location protocol mechanism

As the message schema of Figure 5 shown, the User Agent issues a 'Service Request' (SrvRqst) on behalf of the client application, specifying the characteristics of the service which the client requires. The User Agent will receive a Service Reply (SrvRply) specifying the location of all services in the network which satisfy the request [11].

User Agent is allowed to issue requests directly to Service Agents. In larger networks, one or more Directory Agents are used [11]. Service Agents send register messages (SrvReg) containing all the services they advertise to Directory Agents and receive acknowledgements in reply (SrvAck). These advertisements must be refreshed with the Directory Agent or they expire. User Agents unicast requests to Directory Agent instead of Service Agents, if any Directory Agents are known [11].

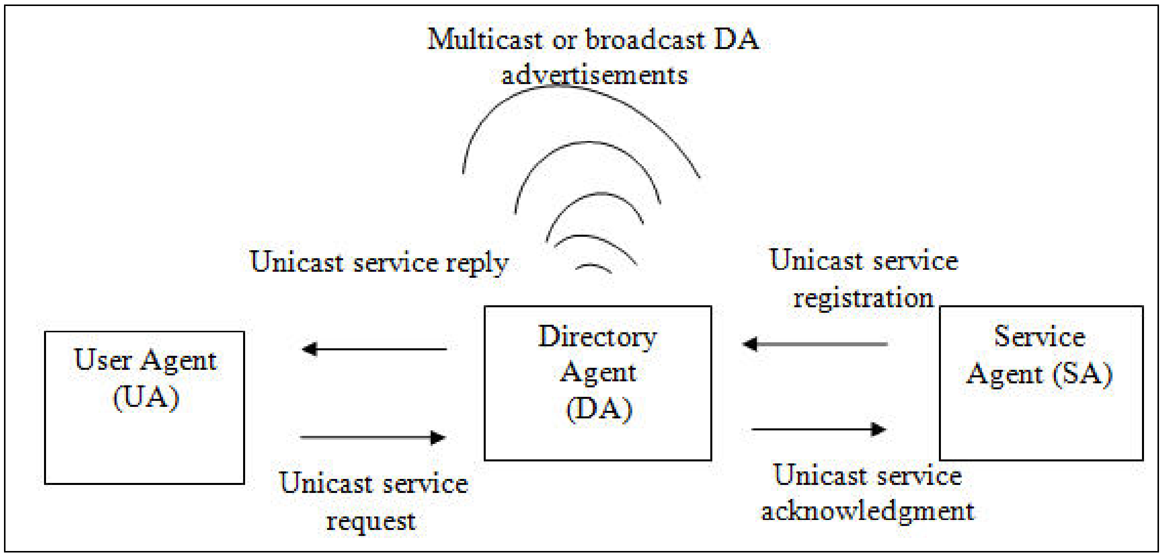

Multicast or broadcast DA advertisements
Unicast service reply
Unicast service registration
User Agent (UA)
Directory Agent (DA)
Service Agent (SA)
Unicast service request
Unicast service acknowledgment

Figure 5. Service location protocol message schema

Because SLP was selected as the server discovery protocol in my thesis, reasons and more SLP details from implementation view such as agent entity and messages will be explained in Section 3.2.2.

2.3.2.4 Comparisons between Jini, UPnP and SLP

To select the most appropriate discovery protocol, three different protocols were compared. All of them provide ways of locating and describing services provided by devices on a network. Each protocol has positive and negative aspects, see Table 2-1. The reason for selecting the Service Location Protocol for this thesis, will explained in Chapter 3.

Table 2-1. Comparison between Jini, UPnP and SLP

Aspect	Jini	UPnP	SLP
Adoption	It was first developed and lunched for small embedded devices.	UPnP website states that there are 648 member vendors of which many are influential actors in the hardware and software field.	SLP seems to have gained acceptance in the Linux domain, mainly as a protocol used by various service location daemons for printers. Axis and Sun Microsystems provide implementations. There is also an Open SLP project.
Openness	Specifications are open and available on Jini website.	Specifications are available on the UPnP web site.	RFCs are accessible from IETF.org and other websites.
Implementation	1. Jini requires a Java run-time environment. 2. CMatos is a C implementation of the Jini framework, in addition to SUN’s implementation. There are several open-source implementations such as Kaffe and Jikes. None of these provide complete implementations of the Java API and cannot be guaranteed to work with Jini libraries. 3. There are no open-source implementations of Jini since SUN prohibits creating non-commercial implementation from scratch.	Libupnp and CyberLink are both major open-source UPnP implementation projects. Microsoft provides an SDK for UPnP for Windows platforms.	Full open-source implementations (Open SLP) exists for C under Linux and for Java.

Portability	Jini is tied to the Java language.	UPnP is not tied to any language or platform. Libupnp library likely to be reusable on Windows platform. CyberLink is portable to any platform which has a JRE.	The Java implementation is portable to any platform having a Java run-time environment.
License	Sun Java run-time environments can be bundled and distributed without license fees. Commercial use of Jini requires special license from Sun. CMatos also has a licensing cost for commercial usage.	Both Libupnp and CyberLink have a BSD-style license.	SLP is provided with a Berkeley Software Distribution (BSD) style license
Other	Java development typically involves fewer memory-related bugs. This is thought to reduce implementation time.	UPnP is targeted towards home networking. Each UPnP device has a built-in HTTP server.	

2.3.3 Transport layer

Interoperability across the transport layer is essential for many user-facing capabilities including data synchronization, transfer, and formats [2]. Although it is appealing to envision of being able to share digital content between intelligent devices in a local wireless network or personal area network; data is still fragmented across devices thus today users cannot always enjoy their music using the device at the hand, photos and videos cannot be shared between these devices, and the cell phone's message cannot move from one device to the other device – despite our desire that our content follow us (or at least that it be available using any device capable of dealing with this type of media).

2.3.4 Application layer

In the Application layer, applications are created to provide ensemble services. For example, the laptop's keyboard might turn into a voip-entry device for a cell phone, or devices at hand are used as on window into one application. Java platform, Micro Edition (Java Me) provides a robust, flexible environment for applications running on mobile and other embedded devices such as cell phone, PDA, and printers. In addition, Anoop Sinha's CrossWeaver system designs applications with multimodal and multi-devices capabilities. Some of devices do not have keyboard and mouse, and thus CrossWeaver embodies the informal prototyping paradigm, leaving design representations in an informal, sketched form and creating a working prototype from these sketches [13].

It is more difficult, more time-consuming to design multiple devices' applications rather than a single device and multiple devices interface should remain a rich field of study for researchers to explore as well [2].

2.3.5 P2P technologies

A Peer to peer (P2P) computer network uses diverse connectivity between participants in a network and the cumulative bandwidth of network participants rather than conventional centralized resources where a relatively low number of servers provide the core value to a service or application [12].

There are many sorts of applications that use Peer to peer technology such as: file sharing, telephony, media streaming (audio, video), and discussion forums. Besides, peer-to-peer networks are classified according to their degree of centralization including pure peer-to-peer networks and hybrid peer-to-peer systems. Gnutella and Freenet are examples of pure peer-to-peer application layer networks designed for file sharing while Napster, KaZaA, CAN, Gnutella, and JXTA are examples of hybrid peer-to-peer systems.

The goal of peer-to-peer network is to provide resources, including bandwidth, storage space, and computing power to all clients, increase robustness by replicating data over multiple peers, and enable peers to find the data without relying on a centralized index server.

2.4 Remote desktop software

In computing, Remote desktop software is remote access and remote administration software that allows GUI applications to be run remotely on a server, while being displayed locally [14]. The quality, speed and functions of any remote desktop protocol are based on the system layer where the graphical desktop is redirected [14].

The current popular remote desktop protocols are virtual network computing (VNC), Remote desktop Protocol (RDP), and X window System (X11).

2.3.2 VNC

VNC was created at the Olivetti & Oracle Research Lab (ORL). VNC is a graphical desktop sharing system which uses a remote frame-buffer protocol to remotely control another computer [15].

VNC is an independent platform. This a VNC system consists of a client, a server, and a communication protocol. As shown in Figure 6. VNC work schema, the VNC viewer runs on the VNC client. The viewer is a program that connects to a VNC server on another system. It is important to note that the viewer and server can run on any operating system to which they have been ported; and that both have been ported to a very large number of operating systems and hardware platforms. Multiple VNC viewers can connect to a VNC server at the same time, thus multiple views of the desktop can be seen on different devices. The VNC server is installed on the computer to be controlled. Connecting to the VNC server requires a password in order to provide some security. When a VNC viewer tries to connect to the VNC server it must provide the correct user name and password.

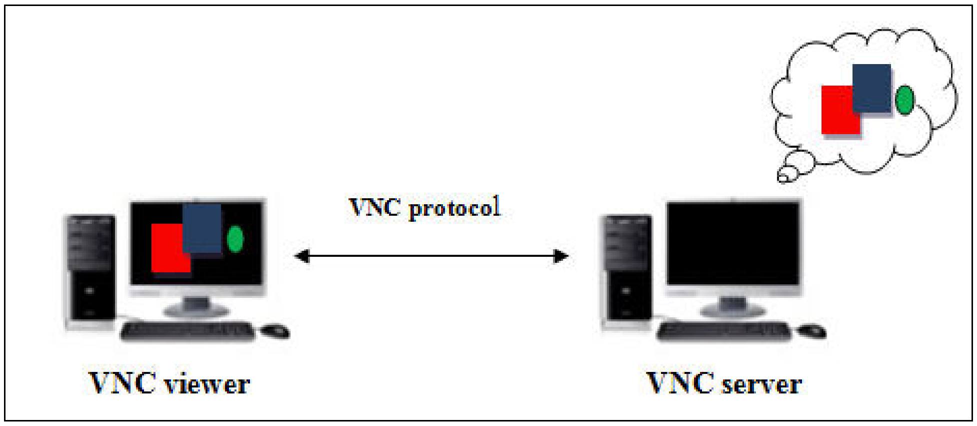
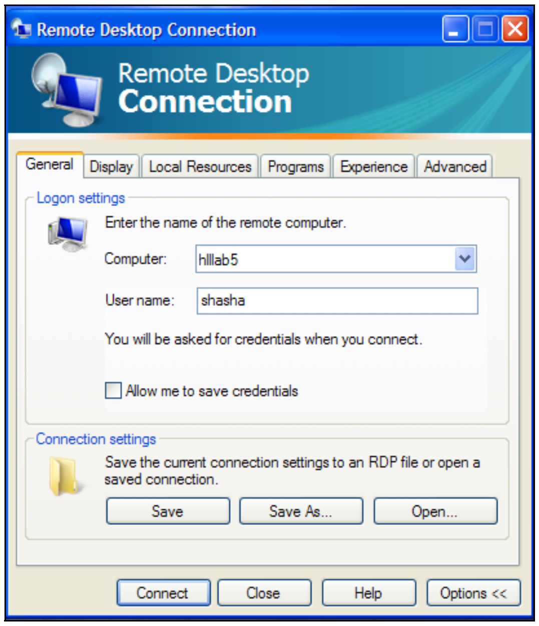


Figure 6. VNC work schema

VNC is commonly used as cross-platform remote desktop protocol. The default TCP ports are 5900 through 5906. Each separate system corresponds to a port. A Java viewer is available in many implementations, such as RealVNC on ports 5800 through 5806 [15].

2.4.1 Remote Desktop Protocol

The Remote Desktop Protocol (RDP) is a multi-channel, Microsoft Windows-specific protocol featuring audio and remote printing. The control panel for RDP is shown in Figure 7. By default, all Microsoft Windows XP and Vista editions include a pre-installed remote desktop connection (RDC) client application. Additionally, the client is available for free download for Microsoft Windows operating systems and most Linux distributions. The RDP server listens by default on TCP port 3389 [16].



Enter the name of the remote computer.

Computer: hlllab5

User name: shasha

You will be asked for credentials when you connect.

☐ Allow me to save credentials

Save the current connection settings to an RDP file or open a saved connection.

Figure 7. Remote Desktop Connection

One the server, RDP uses its own video driver to render display output by placing rendering information into a TCP stream using the RDP protocol. When the client receives this information it completes the rendering on the local display using Microsoft's Win32 graphic device interface API calls. For the input, client mouse and keyboard events are redirected from the client to the server. On the server, RDP uses it own on-screen keyboard and mouse driver to receive these keyboard and mouse events and to input them into the appropriate local drivers [17].

2.4.2 X window system

The X window system (often known as X11 or X) is a windowing system which implements the X display protocol and provides windowing on bitmap displays. X originated at MIT in 1984 and the current protocol version is version 11.

A client-server model is used in X, i.e., with an X server and X clients. This X server communicates with various client programs. The X server enables an application (an X client) to display its output on a remote device and to get input from remote input devices. The X client might be a computationally intensive simulation running on a remote UNIX machine, but it can display its result on a local X Window desktop machine. Using an X server a user can execute graphical software on several machines at once [18].

Figure 8 shows the relationship between the X server, its input devices (e.g., keyboard and mouse), and its output onto a screen. In this example, a web browser and a terminal emulator run on the user's local workstation and another X client runs on a remote machine, but is controlled by the user via their X server. The local X display provides display services to the X server and any remote application using its services is X client. The X server and X client may run on the same machine or on different machines.

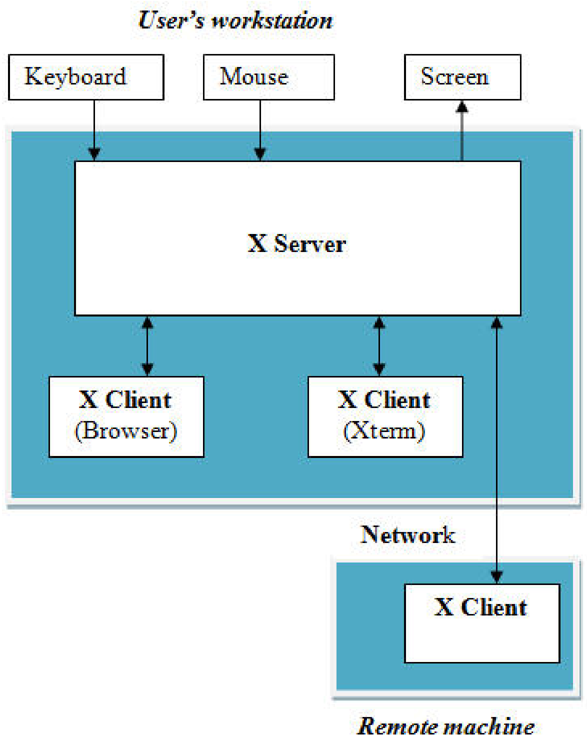


Figure 8. X windows system

2.5 Powering the individual devices

It is necessary to consider the power consumption of each of the device, the power consumption needed to communicate between the devices, and how to power each of devices. Network devices, such as PC, cellular phone, and PDA traditionally received power from wall outlets – with the PDA and cellular phone using this power to charge a battery which is used to power the device when it is not connected to the wall outlet. Devices connected to wired networks are increasingly frequently implementing technologies such a Power over Ethernet – thus they do not directly connect to a wall outlet. Nowadays, USB has become a popular method of connecting and powering computer peripherals – thus avoiding the need for each device to be connected to either a wall outlet or to have a battery of its own. For the purpose of this thesis project we assume that each device has its own source of power or if it is physically wired to another device it might obtain some of all of its power via this wired interface. We will explicitly not consider the problems of operating times, rate of power consumption, etc.; but will rather focus on how the device can learn about the other device & their services and how devices can be aggregated (in terms of protocols).

3 Implementation

This chapter begins with a discussion of the method used to achieve device aggregation. Afterwards, the complete system architecture for device aggregation and the laboratory network environment where the tests will be conducted is fully detailed (from the physical to logical connections). Finally, the protocols and software used are introduced, along with how to develop programs for the PDA.

3.1 Methods

To achieve the device aggregation with data networking, the first step is to implement the service discovery. When the new device enters into the local area network, the new service should be discovered. Secondarily, the application will select the required service in the local area network.

An example will be used to motivate the reason this thesis. Consider that a user wants to display the graphical output of an application which he is using on his cellular phone to surf on Internet. We will assume that the user is near a “bigger display”, that both the user’s cellular phone and the display have some sort of local area network connectivity (here we will assume that this is via a wireless local area network (WLAN)), and that the application could generate a richer visual output if it could utilize this display. However, before the user can make use of the larger display it is necessary to configure the cellular phone so that it knows what services (and devices) are available via the WLAN. Thus the cellular phone could learn of the existence of the “bigger display” and its X windows/RDP/or VNC service that could be made available to the user. After discovering the device, the cell phone can connect to the appropriate device and service in order to use it.

From the above scenario we can see that there are two preconditions for the service: the devices must have some means to communication and they must have some means of knowing what services are available (i.e., that there is so advantage in communicating). Therefore, we developed a system based on the Dynamic Host Client protocol (DHCP) (to dynamically get or assign an address), the Service Location Protocol (SLP) for service discovery, and Virtual Network Computing (VNC) for remote desktop control. We will try to realize this configuration in order to test it in a laboratory network environment. As noted previously this thesis focuses on the first two components (the preconditions for network applications to be used), while the use of VNC is described elsewhere [35].

In this implementation, devices including a KTH+HP SmartBadge version 4 (here after mostly referred to a badge), a PDA, and cellular phone are used. Each of these devices and its operating system will be described in Section 3.2.2. The badge is a USB bus master and it is equipped with a WLAN interface, thus it can control and manage all the intelligent devices forming a personal area network. In this thesis project I have only interconnected it with a PDA and cellular phone (although for testing purposes the WLAN connection through an access point has been used to connect to a variety of other computers for both testing and software development purposes). A DHCP server and Device Agent (an SLP entity -- that will be explained in section 3.3.2) run on the badge - thus the badge can perform dynamic IP address assignment for the other device and it can also provide a central repository for service location information. A DHCP client and

Service Agent (another SLP entity) run on the PDA enabling the PDA to be assigned an IP address and to advertise the location of services. Similarly, a DHCP client and Service Agent should run on cellular phone In order to implement the application – user the "bigger display" as mentioned above, a User Agent (another SLP entity) must also be run on the cellular phone to locate the "bigger display" service. However, neither the protocols nor the necessary software for the cellular phone have been implemented as part of this thesis project.

3.2 System Architecture

System architecture is the most crucial part of this chapter, since it is the foundation for the design and implementation of the system. This architecture is based on using the badge as a central controller to perform the device aggregation. Thus all the other devices will communicate with the badge, using the appropriate interface. In order to develop and test this architecture the requisite equipment was set up in a laboratory environment. Next we will describe this environment in detail, followed by a close examination of the protocols that will be used.

3.2.1 Laboratory environment

The laboratory environment consists of both a network infrastructure and a number of devices. First we will introduce the different networks which were available and describe what computers were attached to each network. Figure 9 illustrates the network infrastructure and introduces some of the relevant computers that were attached to the different subnetworks.

The KTHOPEN WLAN is available throughout the lab and the surrounding campus area. This WLAN is based upon an IEEE 802.11b WLAN. This network is operated by the campus networking service and requires that users login in via web page redirection to an authentication server. This network check that the device is still attached to a given access point by sending periodic ICMP echo requests and if they are not returned, then the devices access to the network is terminated. This means that this network is not very usable by handheld devices that like to power down their WLAN interface. A second WLAN is also available, this is called KTHOPEN-WPA and utilized WPA for security. This network has the advantage that devices do not need to continue to answer ICMP echo requests – however, the version of Windows running on the PDA does not support WPA very well. Thus we make use of a local private WLAN access point. This access point is connected to a LAN which is not connected to the campus networks – thus it does not require that users authenticate themselves, but the only devices that can be reached are a small number of PCs and badges physically near this access point. This third WLAN uses the ESSID "ece8883" (the course number of a course taught by Prof. Mark T. Smith at Georgia Institute of Technology where the access point and the badges were last used).

This local WLAN and the computers which are physically attached to the access point by wired Ethernet connects are on a private IP network with IP addresses in the range 192.168.2/24 (i.e., with a 255.255.255.0 network mask). One of the computers which is attached to this network is a PC named "reheat" (at address 192.168.2.100). This computer is the host for the cross compiler used to generate code for the badge (see section 3.2.2.3.3). Reheat has a second Ethernet which is connected to the department's wired network (which has the address range 130.237.15/24). Note that reheat does not forward IP traffic between these two interfaces.

A wired broadband internet connection (connected via 1Gbps Ethernet connections to a local Ethernet switch) is used to provide an isolated private network for use in the laboratory. This network uses IP addresses in the range 192.168.1/24. There is a firewall which connects this private network with the department's wired network. Some of the computers attached to this network are permitted out through this firewall, but no machines outside this firewall are allowed to originate traffic into computers inside the firewall.

One of the important machines for development and testing is hlllab5, which is connected to all three of the wired networks which have been described above. To be specific it has one interface connected with the IP address 130.237.15.243, another with the IP address 192.168.1.215, and a third interface with the IP address 192.168.2.2. To carry out software development and testing the user can use this computer to reach any of the wired parts of the laboratory network. A separate computer connected to the 130.237.15/24 has a WLAN interface connected to an 18dBi Yagi-Uda antenna that can be used with Wireshark to capture traffic on the WLAN [36].

An additional PC names "ccslex4" is connected on to the firewalled IP subnet and has an IP address in the range 192.168.1/24. This machine is connected via a serial cable to the badge to provide a console for program development, initial configuration, and testing. Once the WLAN interface of the badge has been assigned an address in the 192.168.2/24 network, then this serial interface is generally no longer needed. However, during the course of the testing, this interface was used (twice) to reload the initial FLASH file system on the badge, which had become corrupted by attempting to copy more bytes to the file system than the file system had capacity for. Note that this computer can be remotely accessed using ssh via the user from any of the computers on the192.168.1/24; this includes "hlllab5". Note that any computer on this subnet with an available serial interface could have been used.

The specific badge (badge6) used for this project is assigned the IP address 192.168.2.56 and it will act as a DHCP server to the other WLAN equipped devices in the lab WLAN. Note that this means that the AP point must **not** act as a DHCP server. The badge is equipped with a USB interface and can act as a USB master to control other USB devices. It was originally planned that a handheld cording keyboard would be included in the aggregate device via this interface, but this was not tested as part of my thesis project (however, this keyboard was used in the testing of the VNC based remote display via a laptop computer). Because of the limited amount of memory on the badge, "reheat" is used as a Network File System (NFS) server and the badge acts a NFS client computer. This allows the badge access files on "reheat" over the WLAN network, as easily as if the files were stored in the local memory. This also speeds up software development – since the cross compiler for the badge is also located on "reheat".

Additionally, "hllab5" and "ccslex4" were used during the initial testing of service discovery. This testing lay the foundation to port the program to the badge and to develop a program for service discovery for the PDA (and potentially later for a cellular phone).

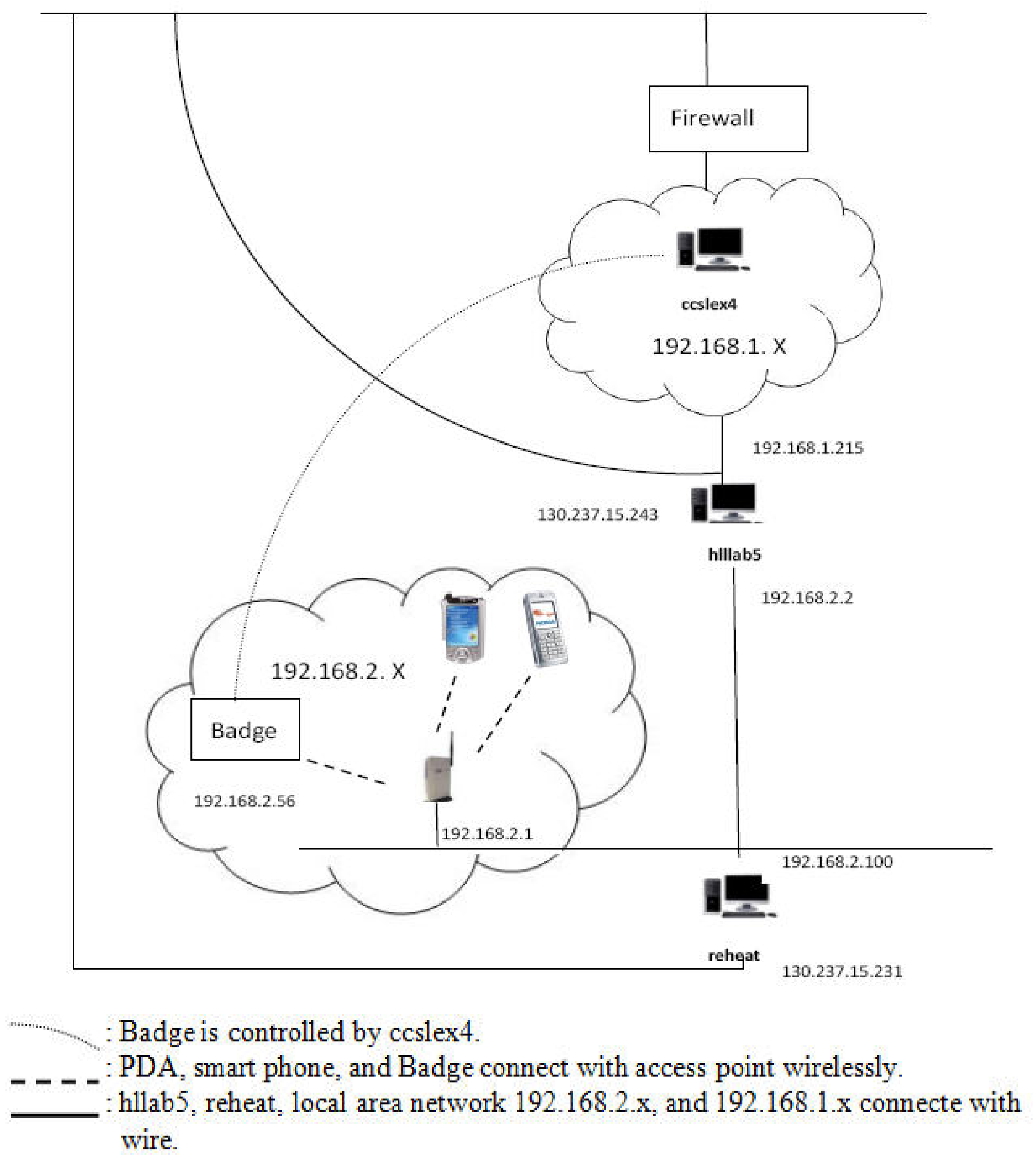

Firewall
ccslex4
192.168.1. X
192.168.1.215
130.237.15.243
hlllab5
192.168.2.2
192.168.2. X
Badge
192.168.2.56
192.168.2.1
192.168.2.100
reheat
130.237.15.231
: Badge is controlled by ccslex4.
: PDA, smart phone, and Badge connect with access point wirelessly.
: hllab5, reheat, local area network 192.168.2.x, and 192.168.1.x connecte with wire.

Figure 9. Laboratory network environment

The following subsections will describe each of the devices used and the tools used, including each of the protocols and all the software running on each device.

3.2.2 Devices used

3.2.2.1 PDA

A personal digital assistant (PDA) is a handheld computer. Such devices are also known as ‘palmtop computers’ [21]. Newer PDAs have color screens and audio capabilities, enabling them to be used as mobile phones, (smart phones), web browsers, or portable media players. Many PDAs can access the Internet, intranets, or extranets via

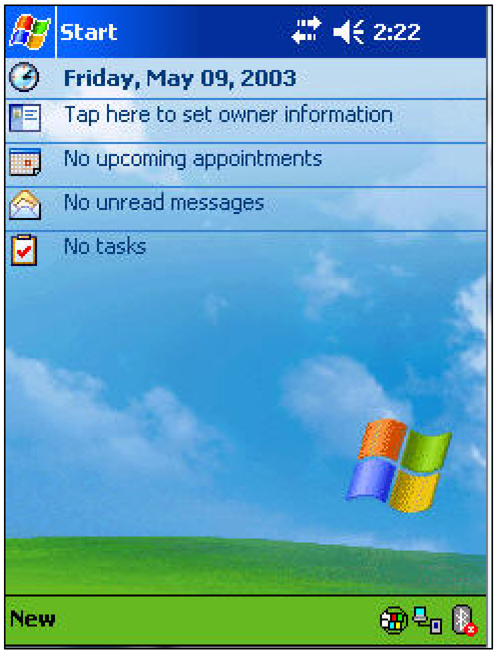
Wi-Fi or Wireless Wide-Area Networks (WWANs). Many PDAs employ touch screen technology [21]. PDAs are used to store information that can be accessed at anytime and anywhere [21].

In this thesis project, an HP iPAQ 5550 PDA is used (see Figure 10). This PDA weighs 207g and measures 8.4cm by 13.8cm by 1.59cm [22]. It has a transflective display capable of displaying over 65,000 colors and uses a Marvell (formerly Intel®) 400 MHz processor with XScale technology (PXA255)[22]. The operating system is Microsoft® Pocket PC 2003. The device is equipped with 128 MB SDRAM, 48 MB Flash ROM (of which 17MB is available as a File Store), both a WLAN 802.11b & Bluetooth v1.1 wireless interfaces, Biometric security, a Secure Digital slot, and advanced power management[22]. The H5550 comes with a 1,250mAh, lithium-ion battery and can be charged via USB when the device is in its USB cradle and connected to a PC or a wall connected transformer is connected to this cradle [22].

Figure 10. HP ipaq 5550

Additionally each user has an iPAQ PC Card Expansion Pack Plus - which enables them to add a PC Card to the iPAQ. This sleeve can accommodate a PC Card type II card. Earlier experiments have examined the use of PC Card based wireless wide area network interfaces.

3.2.2.1.1 Microsoft® Pocket PC 2003



Friday, May 09, 2003
Tap here to set owner information
No upcoming appointments
No unread messages
No tasks

Figure 11. Microsoft® Pocket PC 2003

The preinstalled OS is Microsoft® Pocket PC 2003. This OS is also called "Windows® Mobile 2003" [22]. Windows Mobile is a compact operating system combined with a suite of basic applications for mobile devices based on the Microsoft Win32 API [23]. This operating system is designed to be similar to the desktop versions of Windows, both in function and in appearance [23]. However, developing applications for such a device requires that the application developer keep in mind the smaller screen and that the user is unlikely to have a physical keyboard. Additionally, there is no console window – although some third party consoles do exist. Thus the developer is expected to develop their program either to require no user interface or to build a graphical interface using the windows API.

The operating system includes built in support for a Bluetooth audio headset (for audio input and output), a calendaring system (which has been used in an earlier thesis to provide location aware alarms [37]). Other students have developed applications for this platform, in earlier thesis projects, and there exists a wide variety of software for this platform.

There are several options for developers to use when deploying an application for this platform: writing native code with Visual C++, writing Managed code that works with the .NET Compact Framework, or using Server-side code that can be deployed using Internet Explorer Mobile or a mobile client on the user's device [23]. The .NET Compact Framework is a subset of the .NET Framework, but while it shares many components with the desktop .NET Framework it I not completely compatible. Microsoft has released Windows Mobile Software development kits (SDKs) that work in conjunction with their Visual Studio development environment [23]. These SDKs include emulators that enable developers to test and debug their applications while developing them on their desktop or laptop computer. Microsoft makes Visual Studio 2008 / 2005 Professional Editions, and server / database available to students as free downloads [23]. In addition language-specific versions (Visual Basic, C++, C#, Web) of Visual Studio Express and database components are freely available to anyone via the Microsoft site [23].

3.2.2.1.2 ActiveSync

Synchronizing data with a PC is an important function of PDAs. It is done through synchronization software such as the ActiveSync data synchronization program developed by Microsoft. ActiveSync contact information, calendars, mail, and other files to be synchronized between the PDA and a PC.

This synchronization function was used to develop software on a laptop and to transfer this software to the PDA. In this case, the laptop's operation system is Windows Vista. After connecting the PDA with the laptop via USB, the PDA will display (as shown in Figure 12) a screen indicating that it has successfully synchronized the selected files (and folders). Just as NFS was used to make files quickly available on the badge, the ActiveSync connection via USB or WLAN makes file prepared on the laptop readily available on the PDA.

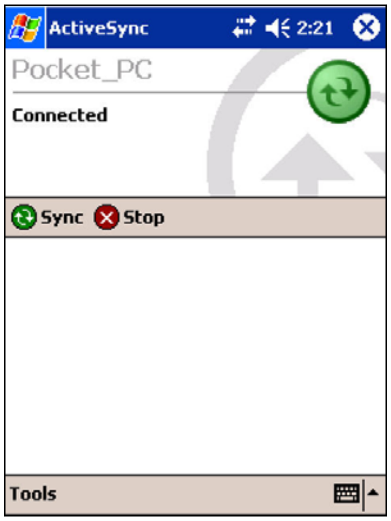


Figure 12. ActiveSync successfully

It is also possible to move files between the laptop by setting up a web server on the laptop and then accessing the files using the built-in copy of Internet Explorer on the PDA. However, this requires more user interaction, thus requiring much more effort – particularly if more than a single file is needed. However, this method might be suitable for distributing an application to a large number of users – who could be located anywhere in the internet.

3.2.2.2 Cell phone

As originally planned this thesis project was to use the Nokia E70 (see Figure 13) as the target cellular phone. The reason that the Nokia E70 was selected was that it was a smartphone with a keyboard. The E series range of smartphone run the S60 3rd Edition on top of Symbian OS version 9.1. There are two models of this phone, the specific phone used was the E70-1 which supports tri-band (900, 1800, 1900MHz) GSM and UMTS [25]. Additionally, the E70 is capable of GSM, 3G, WiFi and Bluetooth connectivity [25]. It should be noted that this smartphone is capable of interacting with multiple Bluetooth devices at the same time.

However, because of a lack of time, the implementation for the Nokia E70 was not finished. However, it was used extensively for the VNC portion of the project (see [35]).

Figure 13. Nokia E70

3.2.2.3 SmartBadge version 4

In the thesis, the KTH+HP SmartBadge version 4, here after referred to simple as the badge (see Figure 14). This board was used because it is capable of acting as USB bus master and because it had support for WLAN (via either a CF or PC Card form factor WLAN card). This is necessary because while cellular phones, PDAs, etc. have USB interfaces, they are not capable of being USB bus masters. Therefore one can not easily connect multiple devices together using USB (unless one of them is laptop or desktop computer!). However, by using a device which is capable of being a USB bus master and a USB hub, multiple USB equipped devices can be interconnected.

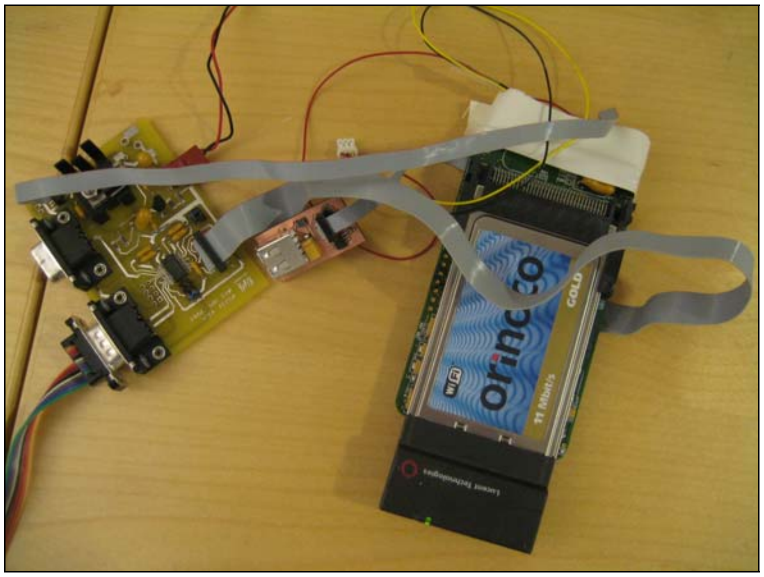


Figure 14. SmartBadge version 4. The serial interface and power supply are connected to the badge for initial configuration and development. The USB interface can be seen to the left of the badge. The small white connector can be used to attach external sensors to the on-board ADC converter.

3.2.2.3.1 How to get started with Badge 4

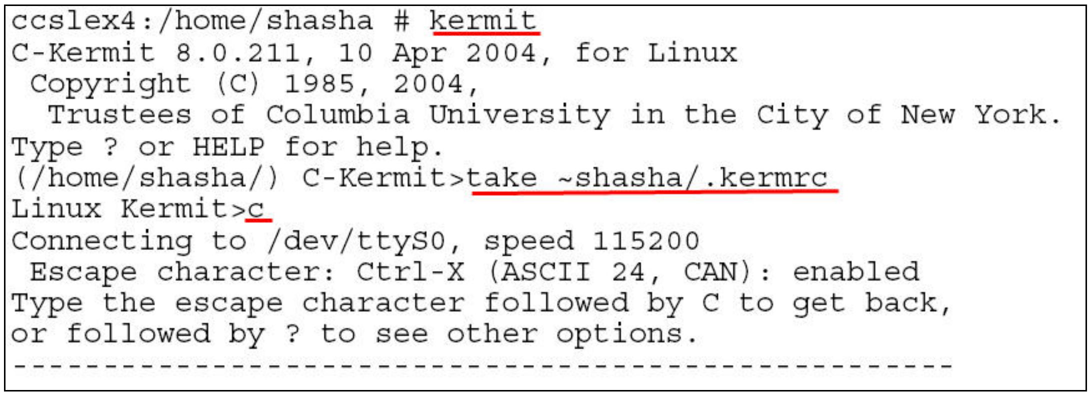
For Linux, the recommended terminal emulator was minicom but I used Kermit to connect to serial port of the badge to configure it. Either program can be used for connecting to the Badge4's serial port via a PC's serial port. In either case, the terminal emulator must be configured for 115200 baud, 8 bits, no parity, and one stop bit (115200, 8N1) [26]. Both hardware and software flow control should be **disabled** [26]. After configuring Kermit on "ccslex4" and saving the configuration in ~shasha/.kermrc. This file contains the following lines:

```
set line /dev/ttyS0
set speed 115200
set flow-control none
set carrier-watch off
set prompt Linux Kermit>
set escape 24
```

After the configuration file is created, you can use the command below to connect to the Badge4 serial port.

```
ccslex4: # kermit
C-Kermit> take ~shasha/.kermrc
Linux Kermit> c
```

A screen dump of this interaction is shown in Figure 15 below. Seeing the shell prompt ("#") from the badge means you have connected successfully.



```
ccslex4:/home/shasha # kermit
C-Kermit 8.0.211, 10 Apr 2004, for Linux
 Copyright (C) 1985, 2004,
  Trustees of Columbia University in the City of New York.
Type ? or HELP for help.
(/home/shasha/) C-Kermit>take ~shasha/.kermrc
Linux Kermit>c
Connecting to /dev/ttyS0, speed 115200
 Escape character: Ctrl-X (ASCII 24, CAN): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------
```

Figure 15. Successfully connecting to the Badge from a PC running Linux

To simplify subsequent operations on the badge, I created a file "startbadge" in ***/home/shasha*** in the badge's file system. This shell script is used to configure the badge's parameters such as IP address and hostname, perform a "mount" command to mount an NFS file system from another computer (where I do the cross compilation of code for the badge), and so on. The contents of "startbadge" are shown in Figure 16. The lines beginning with "#" are comments.

```
#!/bin/sh
#
# set ESSID
#iwconfig eth0 essid ece8883
#
# set badge's ip address
ifconfig eth0 inet 192.168.2.56 netmask 255.255.255.0 broadcast 192.168.2.255
#
hostname badge6
#
# mount remote file system
mount -o rw,nolock,intr 192.168.2.100:/home/badge6 /opt/Badge4
#
# set the date and time
#rdate 192.168.2.100
#
# install badge sensors
modprobe badge4_sensors
```

Figure 16. Contents of the "startbadge" file

3.2.2.3.2 NFS mounting

Because of the limited amount of memory on the badge, “reheat” is used as a Network File System (NFS) server and the badge acts as an NFS client – this allows the badge to access files on “reheat” over the network as easily as if the files were stored in the local memory.

- NFS server

First, the machine from which you want to export the file system needs to be setup (this machine is the NFS server) [26]. In my project, this machine is “reheat”. The file ***/etc/exportfs*** is modified to include the directory to be exported. In our case ***/etc/exportfs*** has one line to export ***/opt/Badge4*** to our Badge (192.168.2.56 is IP address of the client configured above in “startbadge”):

```
/opt/Badge4 192.168.2.56(rw,no_root_squash)
```

After modifying ***/etc/exportfs***, it is important to run:

```
/usr/sbin/exportfs -av
```

Following the above command, both the portmappper and NFS subsystem should be restarted through running:

```
/sbin/service portmap restart
/sbin/service nfs restart
```

Since there are multiple badges in use in this same network, I specify the host name and IP address for my Badge in “startbadge”. Note also that the NFS clients have to be allowed through the firewall on the NFS server machine (if a firewall is running).

The configuration of “reheat” had previously been done; therefore, I did not actually have to perform the configuration steps on “reheat”. Next configuration of the NFS client (to mount the remote NFS file system) is presented.

- NFS client

NFS mounting from the Badge is accomplished by typing the following commands at the Linux prompt [26]:

```
mkdir /opt/Badge4
mount -o rw,nolock,intr 192.168.2.100:/home/badge6 /opt/Badge4
```

192.168.2.100 is the IP address of the NFS server (a.k.a “reheat”). Once the badge’s serial port is connected to the PC and you are able to communicate with it via the Kermit program, then you can type the following command on the badge to configure the IP address, host name, and mount the remote directory:

```
#cd /home/shasha
#./startbadge
# cd /opt/Badge4
```

Now, the badge is able to remotely access files on “reheat” over the network connection, just as if the files are stored on the badge. The local files which appear to be in /opt/Badge4 are actually the files stored at reheat in /home/badge6 /opt/Badge4. As we will see in the next section, this makes development of applications for the badge rather straight forward. Additionally, the use of the NFS server removes the constraints on what can be available via the file system. The limitation is that you must have network connectivity or these files will not be accessible.

3.2.2.3.3 Developing programs for the badge

A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is run [27]. Cross compiler tools are generally used to compile code for embedded systems or multiple platforms. Such a tool must be used for a platform where it is inconvenient or impossible to compile directly on the target platform. For example, a microcontroller with only a minimal amount of memory may not have sufficient resources to run the compiler, therefore a cross compiler must be used. Using cross compilers has become more common with the increasing use of paravirtualization -- where a single physical system may have multiple operating systems and multiple machine architectures in use at a given time.

Because the badge has only a limited amount of local memory, we can consider it to be an embedded platform; therefore a cross compiler must be used to compile the program in order to make an executable file for execution on the badge.

The preferred method for compiling for this platform is to use the uClibs crosses toolchain. The uClibs cross toolchain is initially located in ***/opt/Badge4/2.95-uclibc-0.9.12.tar.bz2*** [26].

For example, to compile a program “udhcp.c” on “reheat”, the following command should be used:

```
#arm-uclibs-gcc –o udhcpd udhcp.c
```

“udhcpd” is the resulting executable file and it should be copied to the badge (or to /home/badge6 /opt/Badge4 which is remotely mounted on the badge), in order to be executed on Badge. Note that the use of the NFS cross mounted file system means that the user can quickly editing, compile, and execute programs – without needing to explicitly copy files between the development system and the target system (where the programs are to be run).

3.3 Protocols and software used

The Dynamic Host Configuration Protocol (DHCP) and Service location Protocol (SLP) are the main protocols used in this thesis project. The specific version of DHCP used was the udhcp (pronounced “micro-DHCP”) implementation, as this was determined to be the most suitable DHCP server for badge. A major reason for the selection of udhcp was that the badge uses busybox (a collection of programs compiled together to reduce their combined memory footprint) and udhcp is designed to be compiled as part of busybox. Details of this will be presented in the following section. OpenSLP is open source software that implements the Service Location Protocol. OpenSLP was used as a reference when developing my own program for the PDA (this will be described in detail in section 3.3.2).

3.3.1 DHCP

The Dynamic Host Configuration Protocol (DHCP) is a protocol used by networked devices (clients) to obtain the parameters necessary for their operation in an Internet Protocol network [28]. This protocol reduces the system administrator’s workload, allowing devices to be added to the network with little or no manual configuration [28]. In this thesis, several devices are to be interconnected to form a personal area network. Therefore, DHCP is used as the protocol to assign an IP address to each device’s interface(s). Initially, on a single PDA and a smart phone are used, but additional devices can be added to this personal area network.

As Figure 17 shows, DHCP consists of four phases: discovery, lease offer, request, and lease acknowledgement. A DHCP client broadcasts a discovery packet on the physical subnet to find available DHCP servers. The IP destination of this broadcast is 255.255.255.255 and the source of broadcast is 0.0.0.0. This source address is used because at this point in time the client does not know what its own IP network address is. (Note that here we are assuming that we are operating in an IPv4 network, rather than using IPv6 auto-configuration.)

When a DHCP server receives the broadcast packet from the DHCP client, the server will determine if it has an address available to assign to this client's interface, if so, then a DHCP lease offer will be sent by DHCP server. This packet contains the MAC address of the client, the IP address assigned, and lease information. The packet may also contain additional information, but we will focus here on these three elements of the lease.

The DHCP client will select a lease from one DHCP server if it receives several responses from DHCP servers. The client will send a DHCP request broadcast to tell all the DHCP servers which server it accepts to assign it an IP address:

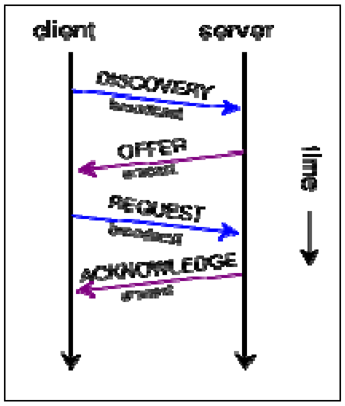


Figure 17. DHCP operation schema

Depending on the implementation, a DHCP server has three methods of allocating an IP address: dynamic allocation, automatic allocation, and static allocation [28]. Dynamic allocation means that the DHCP server is allowed to reclaim and reallocate IP addresses. Automatic allocation is like dynamic allocation, but the DHCP server keeps a table of past IP address assignments, so that it can preferentially assign to a client the same IP address that the client previously had [28]. Finally, using static allocation a DHCP server utilizes a table specifying MAC address/IP address pairs to allocate IP address to DHCP clients. In this project we will use a combination of static allocation (for well known devices – this is primarily for testing) and automatic allocation (i.e., dynamic assignment, but with some history).

3.3.1.1 Udhcp

The udhcp server/client is deliberately targeted at embedded environments [29]. Other linux DHCP servers exist (such as the well know ISC DHCP server), but are targeted at larger systems such as PCs (with more RAM/disk space/etc.) [29]. As a result, udhcp does not have as large a feature set as some other DHCP packages [29].

In this thesis, a full DHCP server would not run on badge as its memory requirements would be too great (especially if other applications are to be run). The badge has only 4 Megabytes (MB) of FLASH memory which stores the operating system and most of the file system – along with 32 MB of regular memory for running applications, the OS, and to hold the dynamic parts of the file system. Compiled against uClibs, the udhcp server and client binaries are each around 18kbytes and when compiled as a combined binary, 28kbytes. Therefore, udhcp is a perfect fit for our badge's required DHCP capabilities [29].

The udhcp server lease file is in binary format making the additional storage space required for IP and MAC addresses minimal [29]. Udhcp also has the option of storing lease times in absolute form, or relative form, for systems without a clock [29]. The lease file can also be saved periodically or by using a signal for systems with flash memory [29]. This last aspect is useful in our setting as the dynamic file is kept in DRAM, thus if the power were turned off the file contents would be lost – but by periodically saving the file to FLASH the contents can be preserved. Hence the leases can be honored, with only a small number getting lost (those between the time of the last commit to FLASH and a sudden power failure).

Udhcp-0.9.6 was used as the basis for the DHCP server implementation for the badge. This program was modified for the purposes of this thesis and cross compiled for the badge. The main source code of Udhcp-0.9.6 with the changes is shown in Appendix A. Note that the modifications were of two kinds: (1) to enable cross compiling and (2) to add extra functionality. The extra functionality which was added was the ability to specify static IP addresses to be assigned to devices based upon a table of IP address MAC address pairs. This table is specified in the extended configuration file.

The cross compiling of udhcp-0.9.6 to create a DHCP server (udhcpd) was done on “reheat” with the command line:

```
#arm-uclibs-gcc –o udhcpd dhcpd.c
```

The excutable file “udhcpd” should be copied into the appropriate directory for later execution. In our case, we have placed the file in /sbin/. A configuration file “/etc/udhcp.conf” should be created on te badge. This file contains configuration data for udhcpd, the DHCP server. This file is a free-form ASCII text file. It consists of statements containing parameters and declarations. The parameters for udhcpd specify the maximum lease time, gateway IP address, and so on. Declarations are used to describe the topology of the network and IP address to be assigned to specific subnets. The contents of “udhcpd.conf” used for our experiments will be shown in section 4.1.

3.3.1.2 PocketDHCP

Sam C. Lin’s PocktDHCP V0.22 [30] was used as the DHCP client running on the PDA. When a DHCP server is available on a network, PocketDHCP allows you to determine the current network interface configurations and to release a leased IP address assignment or to renew such an assignment. PocketDHCP is also a very useful utility to find out your assigned IP address, name server, gateway, etc. The application also shows the IP address of the DHCP server as well as the time and date of the last allocation, along with the expiration time for the IP address lease. A detailed example will be shown in the next chapter.

3.3.2 SLP

Service discovery is an essential step in my thesis if a user with a wirelessly device enters into the new environment and want to use services in the surrounding area [31]. In my thesis, I decided to use SLP as a service discovery protocol for some reasons through comparisons of three service discovery protocols in Section 2.3.2.4. First, SLP is an open standard and has open source software OpenSLP implementation for C under Linux and for Java. It is the foundation for the programming on PDA in future and makes the whole implement process easier. Second, the query language of the Service Location Protocol is fairly capable. It does not only allow simple matching for equality or prefixes of names, but also allows comparisons with mathematical operators such as “<=”,”>=”, which is particularly interesting when used with location based services [31]. It is the most important searching feature in service discovery when the user wants to find services in a given area.

3.3.2.1 SLP implementation specification

Because Section 2.3.2.4 introduced SLP specifically, I will explain SLP from implementation view here.

3.3.2.1.1 SLP agents and message mechanism

First, agent is an important software entity inSLP that processes SLP messages and there are three types SLP agents:

- **User Agent (UA)**: The SLP *User Agent* is a software entity that is looking for one or more *services*. Usually this agent is implemented (at least partially), using a library to which client applications link. This library provides client applications with a simple interface for accessing SLP registered service information [32].
- **Service Agent (SA)**: The SLP *Service Agent* is a software entity that advertises the location of one or more services [32]. SLP advertisement is designed to be both scalable and effective, minimizing the use of network bandwidth through the use of targeted multicast messages, and unicast responses to queries [32].
- **Directory Agent (DA)**: The SLP *Directory Agent* is a software entity that acts as a centralized repository for service location information [32]. Both Service Agents and User Agents make it a priority to discover available Directory Agents, since using a Directory Agent minimizes the number of multicast messages that must be sent [32].

SLP agents communicate with each other using 11 different types of messages. These messages are [32]:

- **Service Request (SrvRqst)**
  Message sent by *User Agent*s to Service Agents and Directory Agents to request the location of a service.
- **Service Reply (SrvRply)**
  Message sent by *Service Agents* and *Directory Agents* in response to a SrvRqst message. The SrvRply contains the URL of the requested service.
- **Service Registration (SrvReg)**
  Message sent by *Service Agents* to Directory Agents containing information about a service that is available.
- **Service Deregister (SrvDeReg)**
  Message sent by *Service Agents* to inform Directory Agents that a service is no longer available.

- **Service Acknowledge (SrvAck)**
  A generic acknowledgment that is sent by *Directory Agents* to Service Agents in response to SrvReg and SrvDeReg messages.
- **Attribute Request (AttrRqst)**
  Message sent by *User Agent*s to request the attributes of a service.
- **Attribute Reply (AttrRply)**
  Message sent by *Service Agents* and *Directory Agents* in response to a AttrRqst. The AttrRply contains the list of attributes that were requested.
- **Service Type Request (SrvTypeRqst)**
  Message sent by *User Agent*s to Service Agents and Directory Agents requesting the types of services that are available.
- **Service Type Reply (SrvTypeRply)**
  Message by *Service Agents* and *Directory Agents* in response to a SrvTypeRqst. The SrvTypeRply contains a list of requested service types.
- **DA Advertisement (DAAdvert)**
  Message sent by *Directory Agents* to let Service Agents and User Agents know where they are.
- **SA Advertisement (SAAdvert)**
  Message sent by Service Agents to let User Agents know where they are.

3.3.2.1.2 Use of TCP, UDP and Multicast in Service Location

The Service Location Protocol requires both UDP (connectionless) and TCP (connection oriented) transport protocols. TCP connections are used for bulk transfer, such as a service registration message, but only when necessary [33]. Connections are always initiated by an agent request or registration, but never by a replying Directory Agent. Service Agents and User Agents use ephemeral ports for transmitting information to the service location port, which is UDP or TCP port number 427 [33].

The Service Location Protocol was designed for use in networks where DHCP is available, or where multicast is supported at the network layer [33]. Broadcasts are used in place of multicast, if a network is not connected to other networks. Because DHCP is available and the wireless local area network is a single subnet in this thesis, therefore broadcast is used instead of multicast. This means that the messages described in the previous section can either be *unicast* to one agent at a time or *broadcast* to all agents in the subnet.

3.3.2.2 OpenSLP

The OpenSLP project has developed an open-source implementation of IETF’s Service Location Protocol (RFC 2608 **Error! Reference source not found.**) for commercial and non-commercial application. The interface conforms to the interface defined in “An API for Service Location” (RFC 2614 **Error! Reference source not found.**). OpenSLP is known to compile and run on many platforms, including Linux and Win32.

The program “slpd” provides Service Agent (and possibly Directory Agent) functionality, along with the ability to maintain a consistent state with respect to the locations of other SLP agents on the network. In this thesis, the badge acts as a static Directory Agent, while the PDA and cellular phone act as Service Agents. Therefore, slpd should be run on the badge, PDA, and cellular phone. In order to produce a binary version of the software for the badge, the OpenSLP source code was placed on “reheat” and the source code of OpenSLP modified to enable it to be compiled by the cross compiler (The update OpenSLP is saved in hlllab5 /home/shasha/update_sourcesoftware). For the PDA, a custom program was developed for the PDA to provide a Service Agent function (it was designed to be similar to and compatible with the OpenSLP). However, porting of the software to the Symbian operating system used by the cellular phone has not been done due to the limited time available for this project.

As mentioned as Section 3.3.2.1.1, the User Agent is usually implemented as a library to which client applications link, providing this client application with a simple interface for accessing SLP registered service information [32]. In OpenSLP, the SLP library (libslp) provides User Agent functionality internally to the application, thus there is no for a local slpd. However, due to limited time, such a simple interface with various client applications has not been part of this thesis project, but should be added in the future.

In OpenSLP, there are two important files, a configuration file “slp.conf” and a static registration file “slp.reg”. These files affect the operation of the OpenSLP daemon (slpd) and any application that uses the OpenSLP library.

The “slp.conf” diverges slightly from the API defined in RFC2614 **Error! Reference source not found.**, because some settings from RFC 2614 were not considered useful or were very difficult to implement by the OpenSLP implementers. The “slp.conf” file syntax follows RFC 2614, with a list of key/value pairs separated by newlines; comment lines begin with a ‘#’ or a ‘;’. An example of the “slp.conf” file for a DA and an SA will be shown in Section 3.3.3.1.

The registration configuration file (“slp.reg”) is very useful in conjunction with OpenSLP as it allows users to statically register legacy services (i.e., these services do not need to implement SLP themselves). The syntax for a registration file follows RFC 2614. This file is read by the OpenSLP daemon (slpd) and all of the registrations specified in this registration file are maintained by this copy of slpd and will remain registered as long as this slpd is alive. An example of a “slp.reg” file, as used for the tests in this thesis will be shown in Chapter **Error! Reference source not found.**, along with a detailed explanation of its contents.

3.3.3 Programming on PDA

In this thesis project, the Service Agent needs to run on each intelligent device. Unfortunately, each of them has a different operating system – other than the badge, none of them runs Linux. As mentioned OpenSLP was written in C for running on Linux and in Java; therefore a new program for each intelligent device must be implemented. The following sections will specific the design of a program developed to provide SLP for the PDA.

3.3.3.1 Preparations for programming on PDA

There are several different platforms in the project including Linux for the badge, Linux on PCs, Microsoft Pocket PC 2003 on the PDA, and Symbian on the cellular phone. The same functionality needs to be implemented for different platforms, specifically to implement a Service Agent (SA). If a new intelligent device with a new platform enters into the personal area network, having an implementation of an SLP SA for it is necessary – otherwise it can not use SLP to find services provided by the other devices.

OpenSLP is the good reference implementation to use when developing SLP for other devices. To make use of OpenSLP to facility my programming, I began by configuring, compiling, and installing OpenSLP on a Linux PC. Once OpenSLP was running on a number of Linux PCs successfully, then the source code could be ported to other platforms. Above we have discussed the port to the badge, in this section we will discuss the development of an SLP SA for a DPA.

The test process for OpenSLP running on Linux PCs will be described in the next chapter. The message mechanism and the format of packets captured between two PCs provided important input during the development of a SA for the PDA.

3.3.3.2 Compiler environment

Choosing a compiler is a crucial beginning for programming. The Microsoft Visual Studio compiler was selected because I want to program in C for a PDA running Microsoft's PocketPC 2003. Additionally, I wanted to use my laptop for this program development and it is running the Windows Vista operation system.

We will begin with an introduction to Microsoft Visual Studio, then the Microsoft Visual 2008 with the Pocket PC 2003 SE Emulator (Figure 18). Microsoft Visual Studio is the main Integrated Development Environment (IDE) from Microsoft [34]. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code as well as managed code for all platforms supported by Microsoft Windows, Windows Mobile, .NET Framework, .NET Compact Framework, and Microsoft Silverlight [34]. Visual Studio supports many programming language (including C/C++, VB.NET, and C#). Microsoft Visual Studio 2008 is focused on development for Windows Vista, Windows 2007 office system, and Web applications [34]. However, there is a Pocket PC 2003 SE Emulator in Microsoft Visual Studio 2008 using a virtual machine to run the full Pocket PC 2003 environment. This emulation environment enabled me to test the application for the PDA quickly and easily.

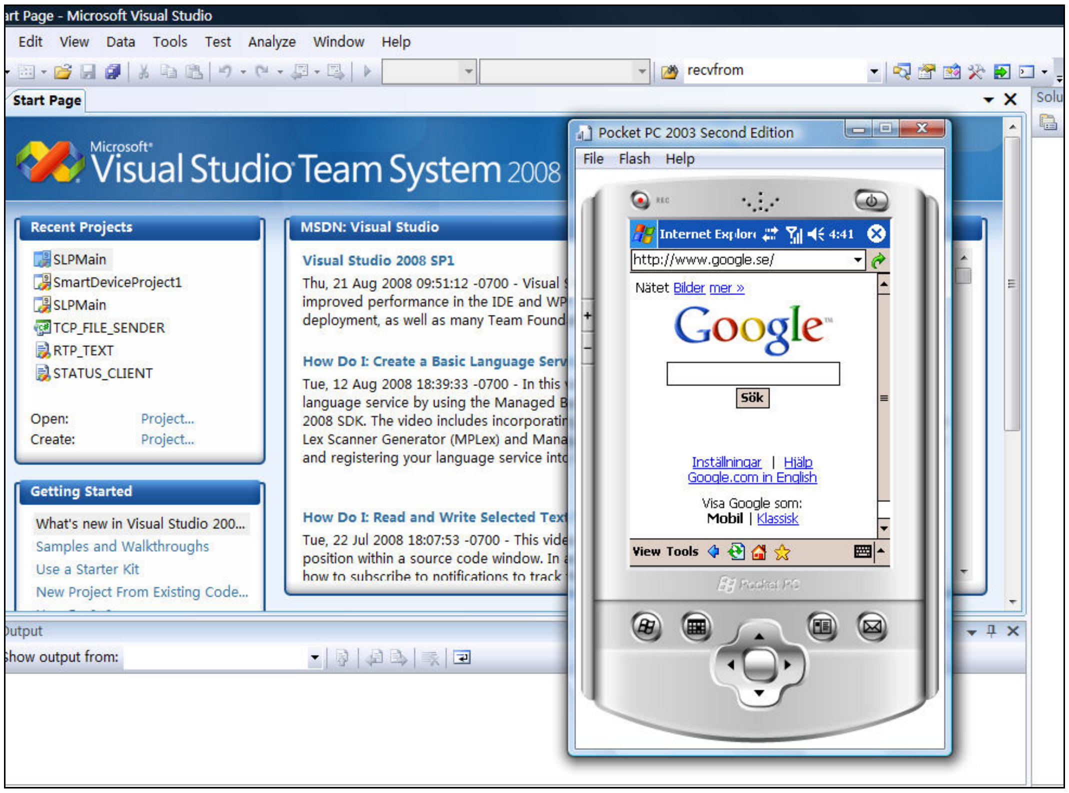
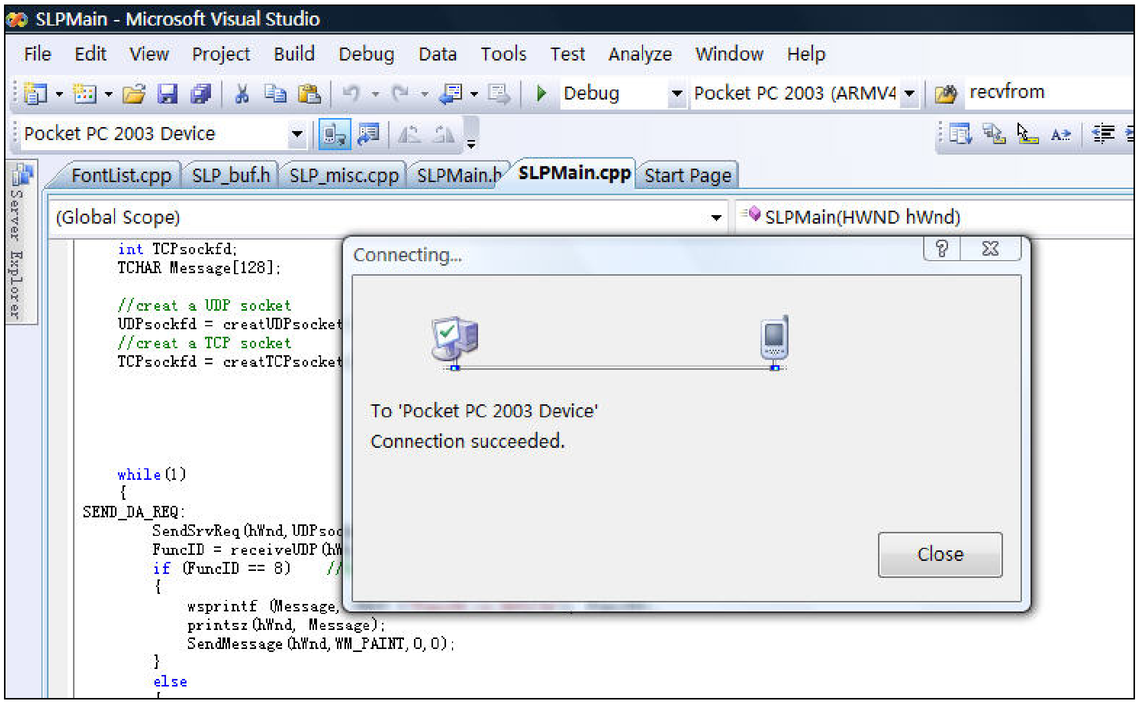

rt Page - Microsoft Visual Studio
Edit View Data Tools Test Analyze Window Help
recvfrom
Start Page
Microsoft Visual Studio Team System 2008
Recent Projects
SLPMain
SmartDeviceProject1
SLPMain
TCP_FILE_SENDER
RTP_TEXT
STATUS_CLIENT
Open: Project...
Create: Project...
Getting Started
What's new in Visual Studio 200...
Samples and Walkthroughs
Use a Starter Kit
New Project From Existing Code...
MSDN: Visual Studio
Visual Studio 2008 SP1
Thu, 21 Aug 2008 09:51:12 -0700 - Visual
improved performance in the IDE and WP
deployment, as well as many Team Found
How Do I: Create a Basic Language Serv
Tue, 12 Aug 2008 18:39:33 -0700 - In this
language service by using the Managed B
2008 SDK. The video includes incorporatin
Lex Scanner Generator (MPLex) and Mana
and registering your language service into
How Do I: Read and Write Selected Text
Tue, 22 Jul 2008 18:07:53 -0700 - This vide
position within a source code window. In
how to subscribe to notifications to track
utput
how output from:
Pocket PC 2003 Second Edition
File Flash Help
Internet Explorer 4:41
http://www.google.se/
Nätet Bilder mer »
Google
Sök
Inställningar | Hjälp
Google.com in English
Visa Google som:
Mobil | Klassisk
View Tools
Pocket PC

Figure 18. Microsoft Visual Studio 2008 with Pocket PC 2003 SE Emulator

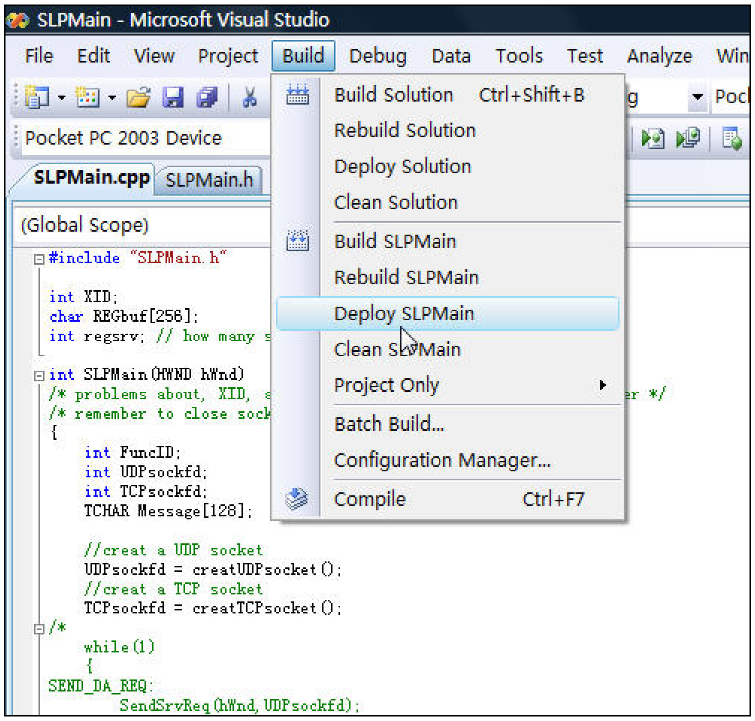
Because UDP and TCP are needed in the program to be developed for the PDA, I configured Pocket PC SE 2003's options to enable it to connect with network. However, in the end I found it is easier to deploy the program directly to H5550 (using a USB connection) and to test the program in the real network environment – therefore, the network configuration for the Pocket PC SE 2003 emulator will not be introduced. After the H5550 is connected via USB to the laptop, then Visual Studio will show Figure 19 that means the Pocket PC was successfully detected. Now the program can be deployed to H5550 after it is built (Figure 20).


SLPMain - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Analyze Window Help
Debug
Pocket PC 2003 (ARMV4
recvfrom
Pocket PC 2003 Device
FontList.cpp SLP_buf.h SLP_misc.cpp SLPMain.h SLPMain.cpp Start Page
(Global Scope)
SLPMain(HWND hWnd)
Server Explorer
Connecting...
Close

```
    int TCPsockfd;
    TCHAR Message[128];

    //creat a UDP socket
    UDPsockfd = creatUDPsocket
    //creat a TCP socket
    TCPsockfd = creatTCPsocket




    while(1)
    {
SEND_DA_REQ:
        SendSrvReq(hWnd, UDPsoc
        FuncID = receiveUDP(hW
        if (FuncID == 8)    //
        {
            wsprintf (Message,
            printsz(hWnd, Message);
            SendMessage(hWnd,WM_PAINT,0,0);
        }
        else
```

To 'Pocket PC 2003 Device'
Connection succeeded.

Figure 19. Connection with H5550 succeed



```
#include "SLPMain.h"

int XID;
char REGbuf[256];
int regsrv; // how many s

int SLPMain(HWND hWnd)
/* problems about, XID,              er */
/* remember to close sock
{
    int FuncID;
    int UDPsockfd;
    int TCPsockfd;
    TCHAR Message[128];

    //creat a UDP socket
    UDPsockfd = creatUDPsocket();
    //creat a TCP socket
    TCPsockfd = creatTCPsocket();
/*
    while(1)
    {
SEND_DA_REQ:
        SendSrvReq(hWnd, UDPsockfd);
```

Figure 20. Deploy program to H5550

Now that all of preparations are completed, the programming for the PDA can begin. Section 3.3.3.3 introduces the design of the program for on the PDA and main functions of this program.

3.3.3.3 Design

The design for a SLP Service Agent for the PDA is mainly based on the Service Agent functionality in OpenSLP. However, the design is simplified to only two functions: sending a packet to the DA and receiving a packet from the DA. The flow chart (Figure 21) shows the algorithm used in my program.

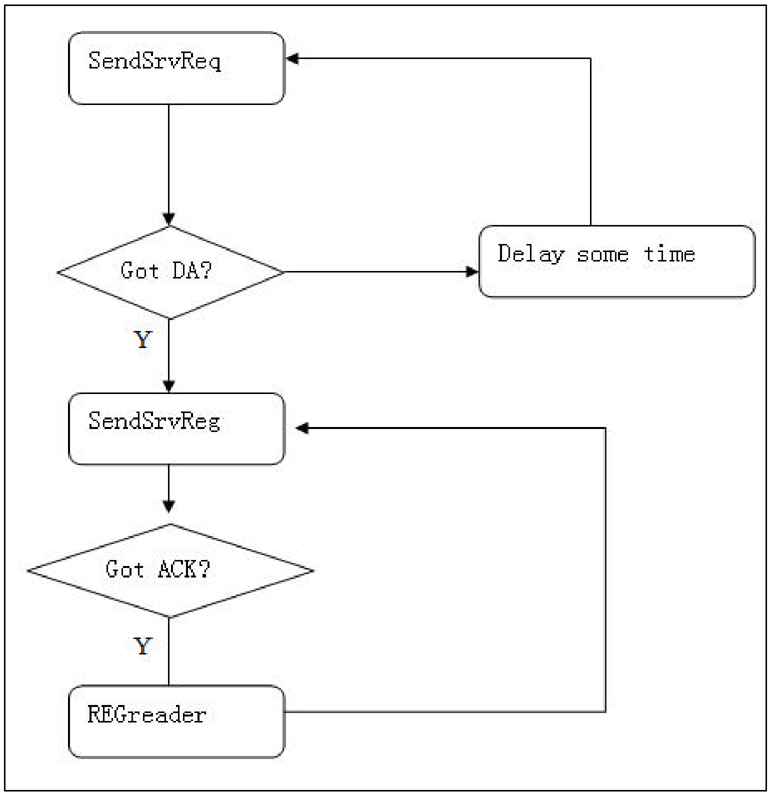
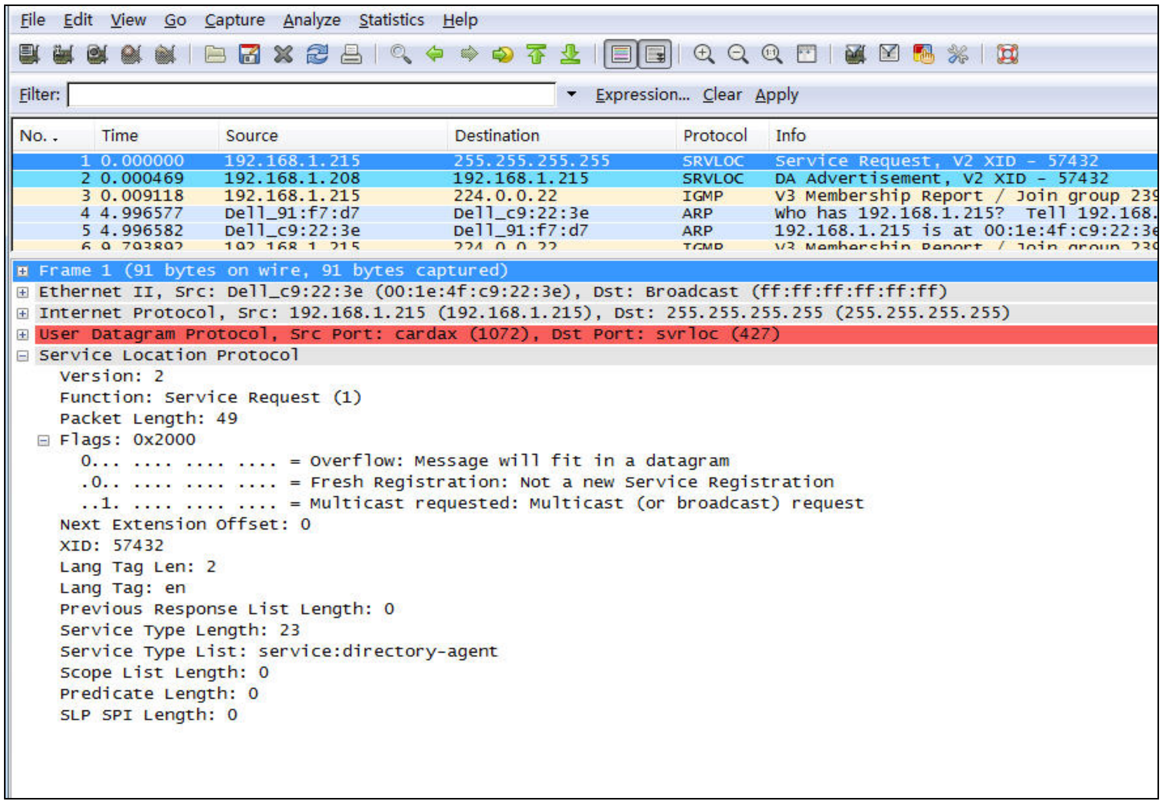


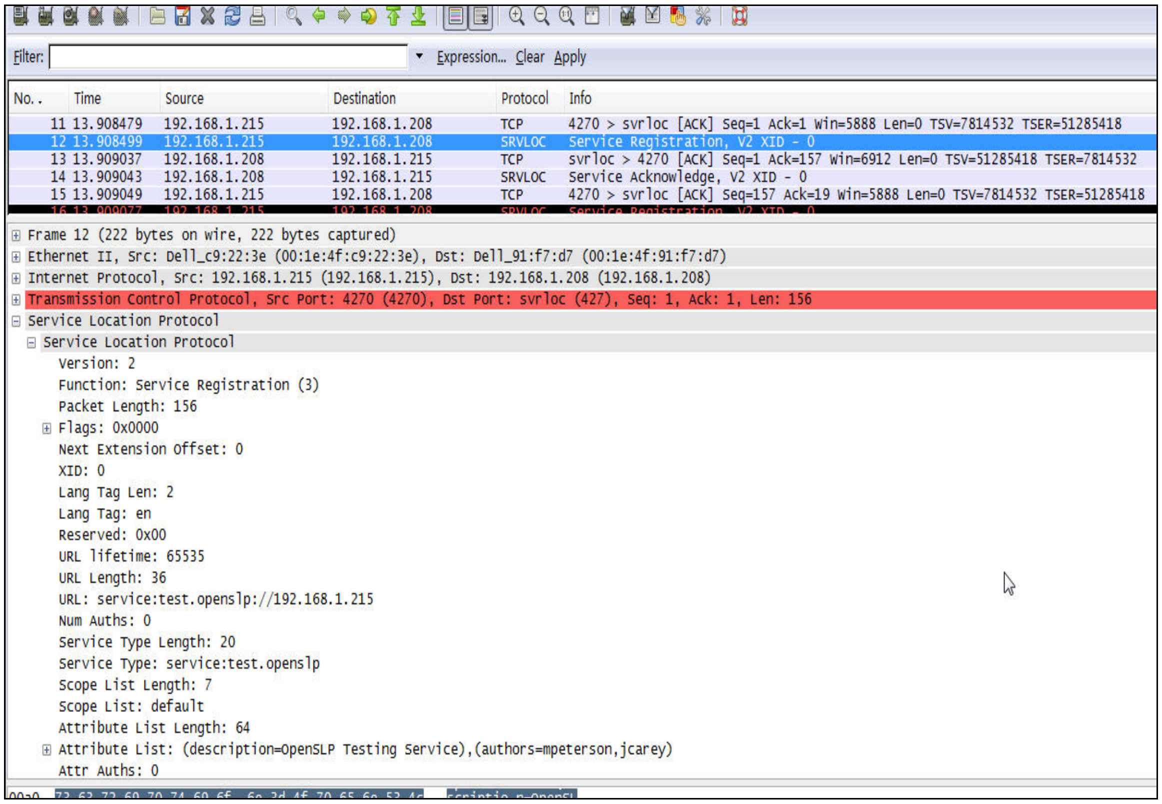
Figure 21. Simple flow chart of SA program on PDA

Because the OpenSLP “slpd” is running as a Directory Agent on the badge, the packet format sent by Service Agent from PDA should be the same as would be sent by OpenSLP. Therefore, I captured the packet sent between two PCs which had OpenSLP running on them successfully, in order to see the exact information in each packet. The two main packets sent from an SA are a Service Request to the DA and a Service Registration. The Service Request message is sent using UDP, while the Service Registration message is sent using TCP (Figure 22, Figure 23). The packet which is sent by the DA or SA uses the exact same format as the packet in these two figures.


File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. .  Time       Source          Destination      Protocol  Info
  1 0.000000   192.168.1.215   255.255.255.255  SRVLOC    Service Request, V2 XID - 57432
  2 0.000469   192.168.1.208   192.168.1.215    SRVLOC    DA Advertisement, V2 XID - 57432
  3 0.009118   192.168.1.215   224.0.0.22       IGMP      V3 Membership Report / Join group 239
  4 4.996577   Dell_91:f7:d7   Dell_c9:22:3e    ARP       Who has 192.168.1.215?  Tell 192.168.
  5 4.996582   Dell_c9:22:3e   Dell_91:f7:d7    ARP       192.168.1.215 is at 00:1e:4f:c9:22:3e

⊞ Frame 1 (91 bytes on wire, 91 bytes captured)
⊞ Ethernet II, Src: Dell_c9:22:3e (00:1e:4f:c9:22:3e), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol, Src: 192.168.1.215 (192.168.1.215), Dst: 255.255.255.255 (255.255.255.255)
⊞ User Datagram Protocol, Src Port: cardax (1072), Dst Port: svrloc (427)
⊟ Service Location Protocol
    Version: 2
    Function: Service Request (1)
    Packet Length: 49
  ⊟ Flags: 0x2000
      0... .... .... .... = Overflow: Message will fit in a datagram
      .0.. .... .... .... = Fresh Registration: Not a new Service Registration
      ..1. .... .... .... = Multicast requested: Multicast (or broadcast) request
    Next Extension Offset: 0
    XID: 57432
    Lang Tag Len: 2
    Lang Tag: en
    Previous Response List Length: 0
    Service Type Length: 23
    Service Type List: service:directory-agent
    Scope List Length: 0
    Predicate Length: 0
    SLP SPI Length: 0
```

Figure 22. Service Request packet sent by Service Agent



```
No. .  Time        Source          Destination     Protocol  Info
 11 13.908479   192.168.1.215   192.168.1.208   TCP       4270 > svrloc [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSV=7814532 TSER=51285418
 12 13.908499   192.168.1.215   192.168.1.208   SRVLOC    Service Registration, V2 XID - 0
 13 13.909037   192.168.1.208   192.168.1.215   TCP       svrloc > 4270 [ACK] Seq=1 Ack=157 Win=6912 Len=0 TSV=51285418 TSER=7814532
 14 13.909043   192.168.1.208   192.168.1.215   SRVLOC    Service Acknowledge, V2 XID - 0
 15 13.909049   192.168.1.215   192.168.1.208   TCP       4270 > svrloc [ACK] Seq=157 Ack=19 Win=5888 Len=0 TSV=7814532 TSER=51285418

⊞ Frame 12 (222 bytes on wire, 222 bytes captured)
⊞ Ethernet II, Src: Dell_c9:22:3e (00:1e:4f:c9:22:3e), Dst: Dell_91:f7:d7 (00:1e:4f:91:f7:d7)
⊞ Internet Protocol, Src: 192.168.1.215 (192.168.1.215), Dst: 192.168.1.208 (192.168.1.208)
⊞ Transmission Control Protocol, Src Port: 4270 (4270), Dst Port: svrloc (427), Seq: 1, Ack: 1, Len: 156
⊟ Service Location Protocol
  ⊟ Service Location Protocol
      Version: 2
      Function: Service Registration (3)
      Packet Length: 156
    ⊞ Flags: 0x0000
      Next Extension Offset: 0
      XID: 0
      Lang Tag Len: 2
      Lang Tag: en
      Reserved: 0x00
      URL lifetime: 65535
      URL Length: 36
      URL: service:test.openslp://192.168.1.215
      Num Auths: 0
      Service Type Length: 20
      Service Type: service:test.openslp
      Scope List Length: 7
      Scope List: default
      Attribute List Length: 64
    ⊞ Attribute List: (description=OpenSLP Testing Service),(authors=mpeterson,jcarey)
      Attr Auths: 0
```

Figure 23. Service Registration packet sent by Service Agent

The source code of program for the PDA is in Appendix B. Unfortunately the code current only correctly sends the Service Request to the DA. The next chapter will describe all of the tests conducted with the software and program in my thesis. This is followed by an analysis of these results.

3.3.4 Remote display software

The implementation of remote display software was an important application for the thesis. To complete the device aggregation, we must combine the simple remote display application with the SLP implementation. As mentioned in Section 3.3.2.1.1, the UA in SLP is usually implemented as a library to which client applications link. This library provides client applications with a simple interface for accessing SLP registered service information. The client application would be the remote display software. However, as the remote display work is not yet finished, it could not be tested.

4 Testing and Analysis

In this chapter, each of the implementation mentioned in the previous chapter (as being available and working) will be tested. First, the DHCP server on the badge and a DHCP client on the PDA will be tested because having a DHCP assigned IP address is essential for subsequent tests.

SLP was tested in four stages. The first stage tests if OpenSLP works correctly on two Linux PCs. If this test is successful, this means that the OpenSLP implementation satisfies my basic requirement and can be used. The next stage tests if OpenSLP works well on badge and a linux PC – with the OpenSLP daemon “slpd” running as a DA on the badge. This test is done to ensure that the OpenSLP daemon “slpd” can successfully run on the badge – where it needs to run. The next stage tests if the SA program for the PDA and OpenSLP daemon “slpd” running as a DA on a linux PC works successfully. The last stage tests the SA on the PDA with the OpenSLP daemon “slpd” as DA running on the badge in a personal area network.

While testing SLP, Wireshark is used as a packet sniffer. If the packets captured by Wireshark are similar to those in the test between two PCs, then SLP is working successfully.

4.1 The DHCP server and the DHCP client

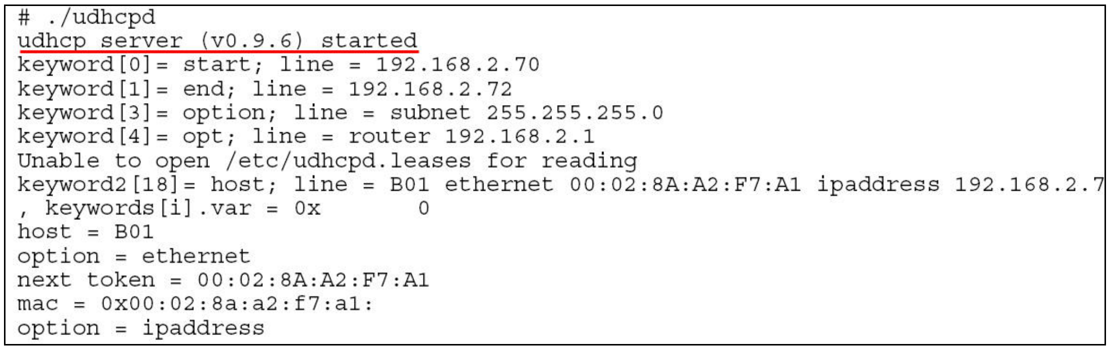
The first test utilizes a DHCP server running on the badge and a DHCP client running on the PDA. The DHCP server’s configuration file (“udhcpd.conf”) is shown below. The DHCP server use a static IP allocation now to facilitate testing (because it is easier to detect which device in the personal area network is which based upon a static IP address – rather than needing to consult the lease file or having to recognize the MAC addresses of each of the devices).

```
#The start and end of the IP lease block
start 192.168.2.70
end 192.168.2.72
option    subnet    255.255.255.0
opt       router    192.168.2.1
#assign the static IP address to the host with Mac address

host B01 ethernet 00:02:8A:A2:F7:A1 ipaddress 192.168.2.71
```

After all configurations for the DHCPA server on the badge has been place in the badge’s /etc directory, then the following command starts the udhcp server on the badge:

```
#./udhcpd
```

If the udhcp server is running successfully, the programs output will be similar to that shown in Figure 24:

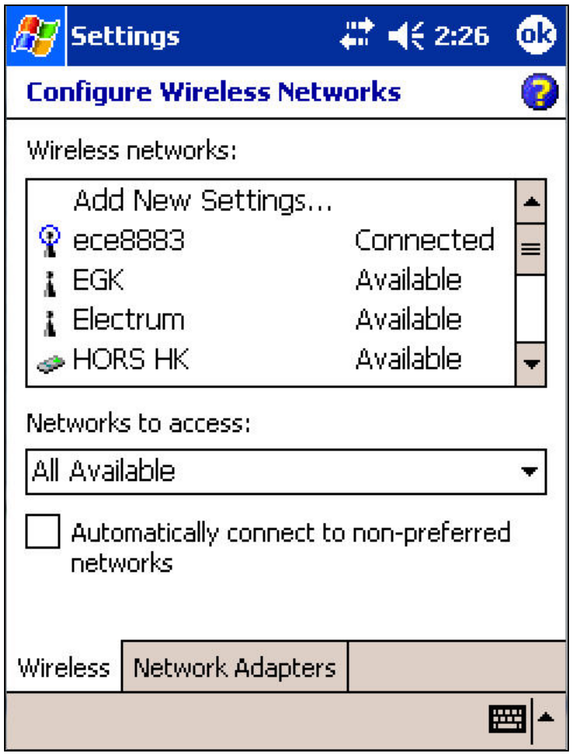


```
# ./udhcpd
udhcp server (v0.9.6) started
keyword[0]= start; line = 192.168.2.70
keyword[1]= end; line = 192.168.2.72
keyword[3]= option; line = subnet 255.255.255.0
keyword[4]= opt; line = router 192.168.2.1
Unable to open /etc/udhcpd.leases for reading
keyword2[18]= host; line = B01 ethernet 00:02:8A:A2:F7:A1 ipaddress 192.168.2.7
, keywords[i].var = 0x       0
host = B01
option = ethernet
next token = 00:02:8A:A2:F7:A1
mac = 0x00:02:8a:a2:f7:a1:
option = ipaddress
```

Figure 24. udhcp server started

Note that the above output is for debugging purposes and could be disabled, but it shows that the configuration file has been successfully parse and the IP address statically assigned to the specified MAC address.

ADHCP client, in this case PocketDHCP V0.22, was installed on the PDA. With the DHCP server on the badge is running, I connected the PDA to the wireless local area network “ece8883” (see section 3.2.1) as follows:



Wireless networks:

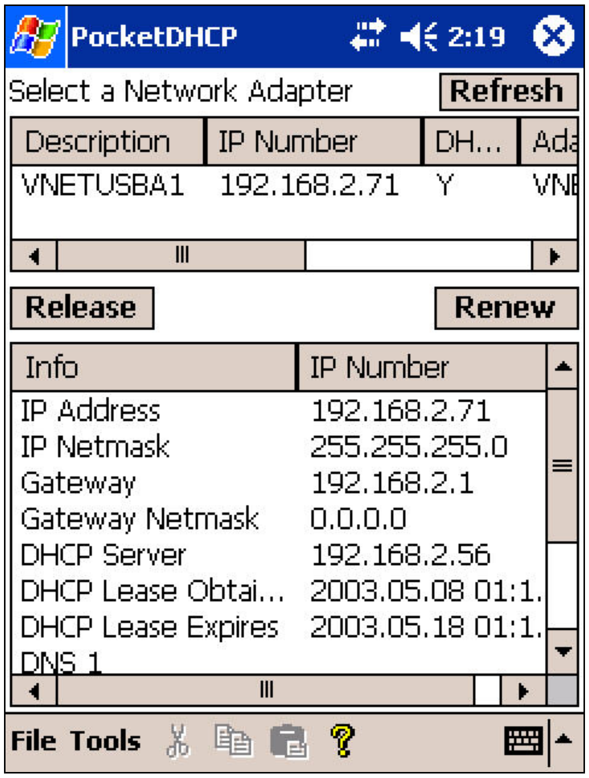
Add New Settings...	
ece8883	Connected
EGK	Available
Electrum	Available
HORS HK	Available

Networks to access: All Available

☐ Automatically connect to non-preferred networks

Figure 25. H5550 connected to the network with ESSID “ece8883”

Using PocketDHCP I requested it to renew the IP address. The output of Pockdhcp is shown in **Error! Reference source not found.** will show the specific IP address assignment information on H5550.



Description	IP Number	DH...	Ad
VNETUSBA1	192.168.2.71	Y	VN

Info	IP Number
IP Address	192.168.2.71
IP Netmask	255.255.255.0
Gateway	192.168.2.1
Gateway Netmask	0.0.0.0
DHCP Server	192.168.2.56
DHCP Lease Obtai...	2003.05.08 01:1.
DHCP Lease Expires	2003.05.18 01:1.
DNS 1	

Figure 26. DHCP details shown by PocketDHCP

As show in the figure “192.168.2.71” is the IP address assigned to the based upon the configuration file shown above. “Y” means DHCP is used to assign the IP address. The lower part of the screen shows additional information, such as the IP address, netmask, gateway, IP address of the DHCP server, etc... The lease is valid for 10 days from **Error! Reference source not found.**. Note that because the badge was not synchronized with a time source nor was the date and time set manually, it uses the above absurd date and time in 2003!

Despite the silly date and time for the lease, the DHCP server on the badge and the DHCP client on PDA worked together successfully.

4.2 Openslp on two PCs

During this test stage, I utilized “ccslex4” and “hlllab5” as the two linux PCs to test the basic operation of OpenSLP with the DA running on “ccslex4” and the SA running on“hlllab5” (Figure 27).

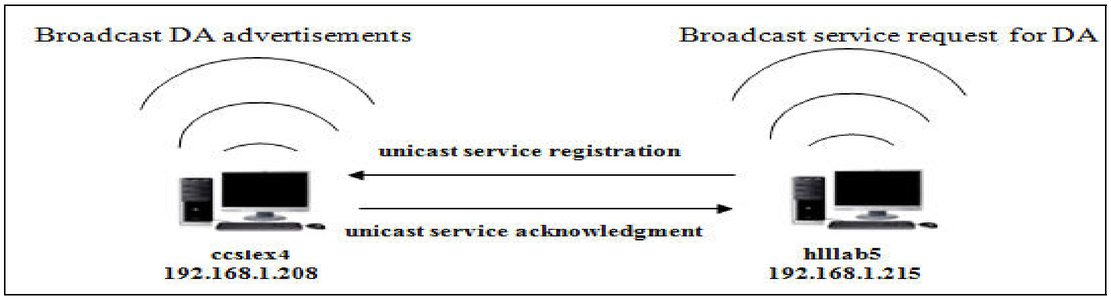

Broadcast DA advertisements
Broadcast service request for DA
unicast service registration
unicast service acknowledgment
ccslex4
192.168.1.208
hlllab5
192.168.1.215

Figure 27. DA and SA on PCs

OpenSLP was installed on the two PCs. The configuration file (“slp.conf”) was modified to create a DA or SA, the resulting two “slp.conf” files are show below and the main changes will be explained.

The changes to the “slp.conf” configuration file for the DA running on “ccslex4” are shown below; all of the other contents of the default slp.conf file not shown are the same as the default file. The comments explain why each line is specified as it is:

```
# This line enables slpd to function as a DA
net.slp.isDA=true
# This is a 32 bit integer giving the number of seconds between the DA heartbeats.
# I configured this interval to be 1 second.
net.slp.DAHeartBeat=1
# Broadcast is used in the place of multicast as there is single subnet to which I
# want the packets sent to so, I force broadcast to be used.
Net.slp.isBroadcastOnly=true
# I make “ccslex4” as DA listen on interface 192.168.1.208
Net.slp.interfaces=192.168.1.208
```

The result of the above configuration will be that the DA will send a link local broadcast on the interface which has the address 192.168.1.208. This is necessary as if the machine has multiple interfaces, it will not know which interface to send this advertisement on. The final line also causes the DA to listen on this same interface for unicast service registrations.

The “slp.conf” configuration for “hlllab5” uses the default configuration file.

```
# Because “slp.conf” is by defaultan SA, the default configuration does not need to
be changed.
#Broadcast is used in the place of multicast as there is a single subnet to which I
# want the broadcasts to be sent so I force broadcast to be used.
Net.slp.isBroadcastOnly=true
# I enable SA to send active DA discovery SrvRqsts periodically.
net.slp.DAActiveDiscoveryInterval=1
# For testing, I set the period to 500 ms to give the maximum amount of time
# to perform active DA discovery requests.
Net.slp.DADiscoveryMaximumWait=500
# Force “hlllab5” as an SA to listen on interface 192.168.1.215
Net.slp.interfaces=192.168.1.215
```

As a result of the above configuration “hlllab5” will send broadcast messages every 500ms looking for a DA.

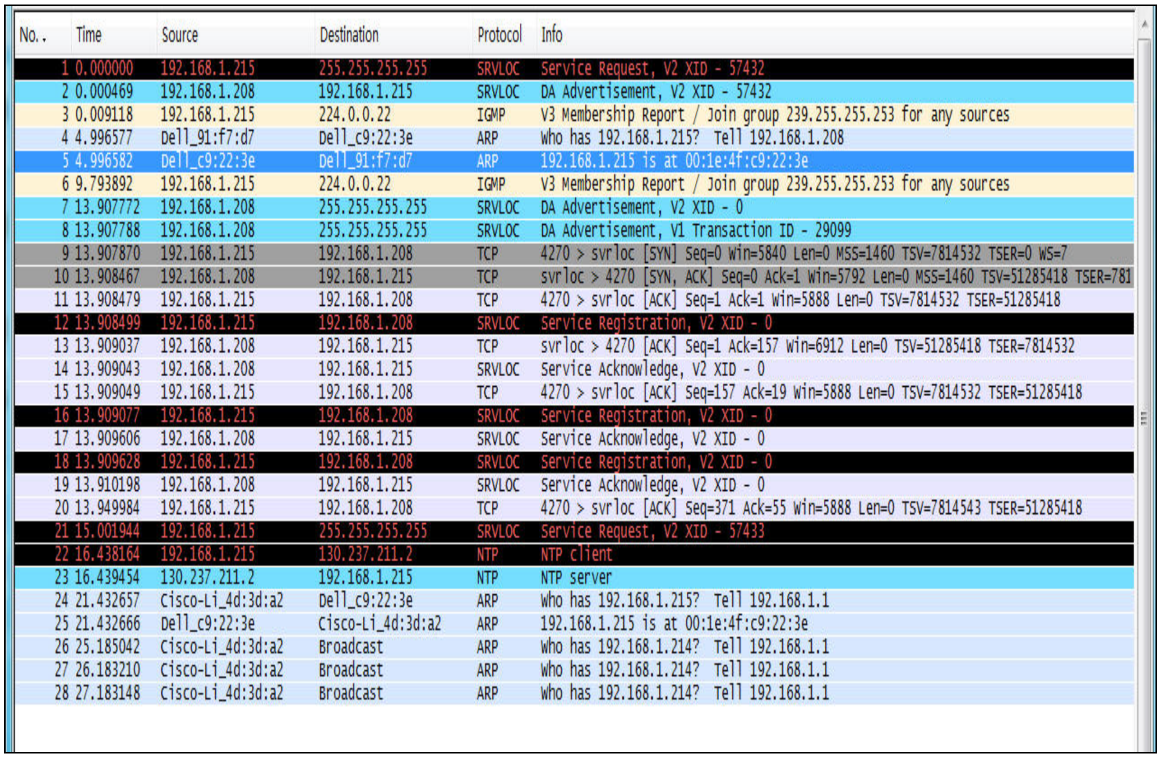
The next step is to configure some services which the SA can register. For testing three services have been listed in the file “slpd.reg” on “hllab5”:

```
##Register a OpenSLP test service
service:test.openslp://192.168.1.215,en,65535
description=OpenSLP Test Service
authors=shasha
##Register a ssh service
service:ssh.openslp://192.168.1.215,en,65535
description= “Secure Shell”
##Register a telnet service
service:ssh.telnet://192.168.1.215,en,65535
```

After the configuration files have been set up, then “slpd” can be started on the two linux PCs with command:

```
#/usr/sbin/slpd
```

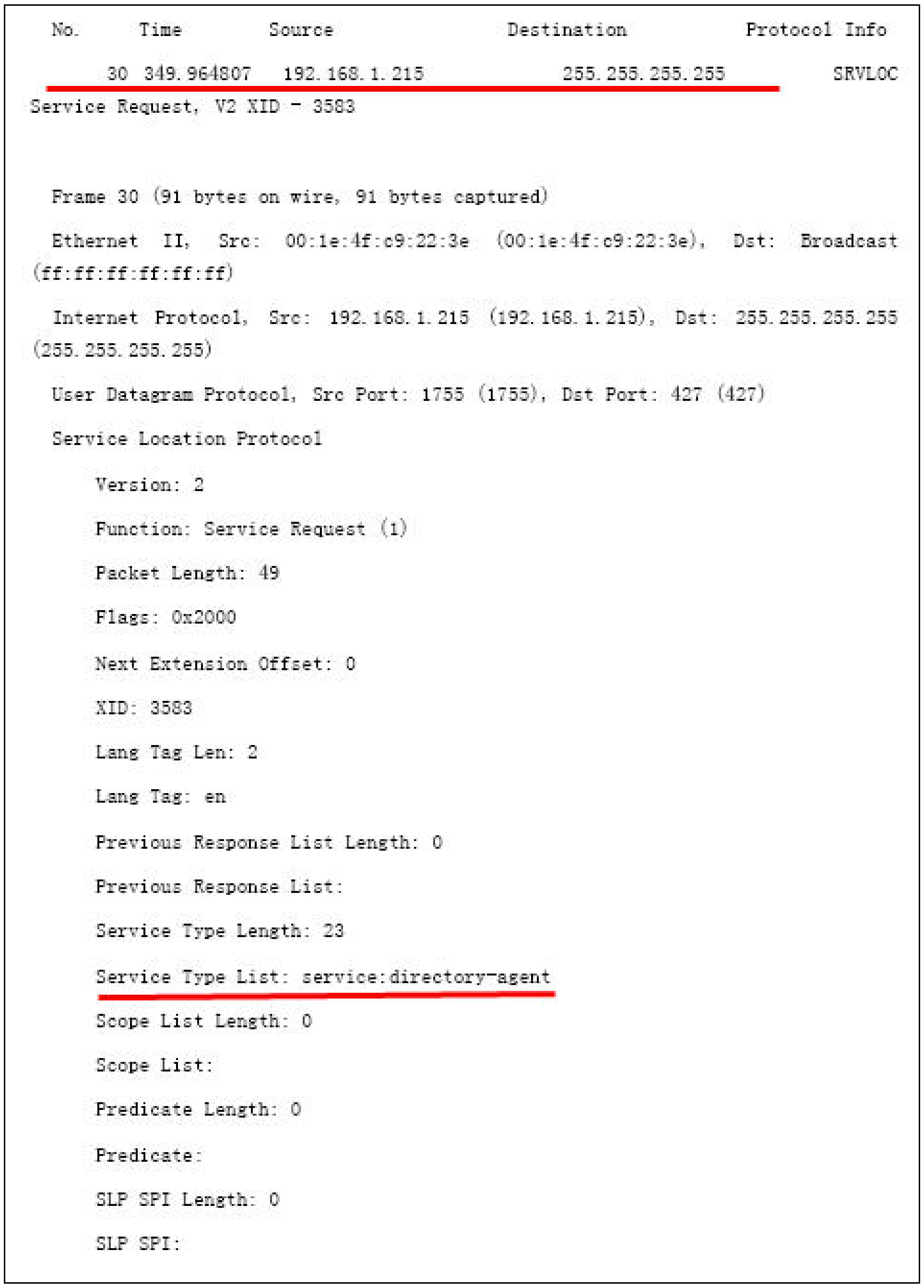
The following packets between the two linux PCs were captured with Wireshark:



No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.215	255.255.255.255	SRVLOC	Service Request, V2 XID - 57432
2	0.000469	192.168.1.208	192.168.1.215	SRVLOC	DA Advertisement, V2 XID - 57432
3	0.009118	192.168.1.215	224.0.0.22	IGMP	V3 Membership Report / Join group 239.255.255.253 for any sources
4	4.996577	Dell_91:f7:d7	Dell_c9:22:3e	ARP	Who has 192.168.1.215? Tell 192.168.1.208
5	4.996582	Dell_c9:22:3e	Dell_91:f7:d7	ARP	192.168.1.215 is at 00:1e:4f:c9:22:3e
6	9.793892	192.168.1.215	224.0.0.22	IGMP	V3 Membership Report / Join group 239.255.255.253 for any sources
7	13.907772	192.168.1.208	255.255.255.255	SRVLOC	DA Advertisement, V2 XID - 0
8	13.907788	192.168.1.208	255.255.255.255	SRVLOC	DA Advertisement, V1 Transaction ID - 29099
9	13.907870	192.168.1.215	192.168.1.208	TCP	4270 > svrloc [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TSV=7814532 TSER=0 WS=7
10	13.908467	192.168.1.208	192.168.1.215	TCP	svrloc > 4270 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 TSV=51285418 TSER=781
11	13.908479	192.168.1.215	192.168.1.208	TCP	4270 > svrloc [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSV=7814532 TSER=51285418
12	13.908499	192.168.1.215	192.168.1.208	SRVLOC	Service Registration, V2 XID - 0
13	13.909037	192.168.1.208	192.168.1.215	TCP	svrloc > 4270 [ACK] Seq=1 Ack=157 Win=6912 Len=0 TSV=51285418 TSER=7814532
14	13.909043	192.168.1.208	192.168.1.215	SRVLOC	Service Acknowledge, V2 XID - 0
15	13.909049	192.168.1.215	192.168.1.208	TCP	4270 > svrloc [ACK] Seq=157 Ack=19 Win=5888 Len=0 TSV=7814532 TSER=51285418
16	13.909077	192.168.1.215	192.168.1.208	SRVLOC	Service Registration, V2 XID - 0
17	13.909606	192.168.1.208	192.168.1.215	SRVLOC	Service Acknowledge, V2 XID - 0
18	13.909628	192.168.1.215	192.168.1.208	SRVLOC	Service Registration, V2 XID - 0
19	13.910198	192.168.1.208	192.168.1.215	SRVLOC	Service Acknowledge, V2 XID - 0
20	13.949984	192.168.1.215	192.168.1.208	TCP	4270 > svrloc [ACK] Seq=371 Ack=55 Win=5888 Len=0 TSV=7814543 TSER=51285418
21	15.001944	192.168.1.215	255.255.255.255	SRVLOC	Service Request, V2 XID - 57433
22	16.438164	192.168.1.215	130.237.211.2	NTP	NTP client
23	16.439454	130.237.211.2	192.168.1.215	NTP	NTP server
24	21.432657	Cisco-Li_4d:3d:a2	Dell_c9:22:3e	ARP	Who has 192.168.1.215? Tell 192.168.1.1
25	21.432666	Dell_c9:22:3e	Cisco-Li_4d:3d:a2	ARP	192.168.1.215 is at 00:1e:4f:c9:22:3e
26	25.185042	Cisco-Li_4d:3d:a2	Broadcast	ARP	Who has 192.168.1.214? Tell 192.168.1.1
27	26.183210	Cisco-Li_4d:3d:a2	Broadcast	ARP	Who has 192.168.1.214? Tell 192.168.1.1
28	27.183148	Cisco-Li_4d:3d:a2	Broadcast	ARP	Who has 192.168.1.214? Tell 192.168.1.1

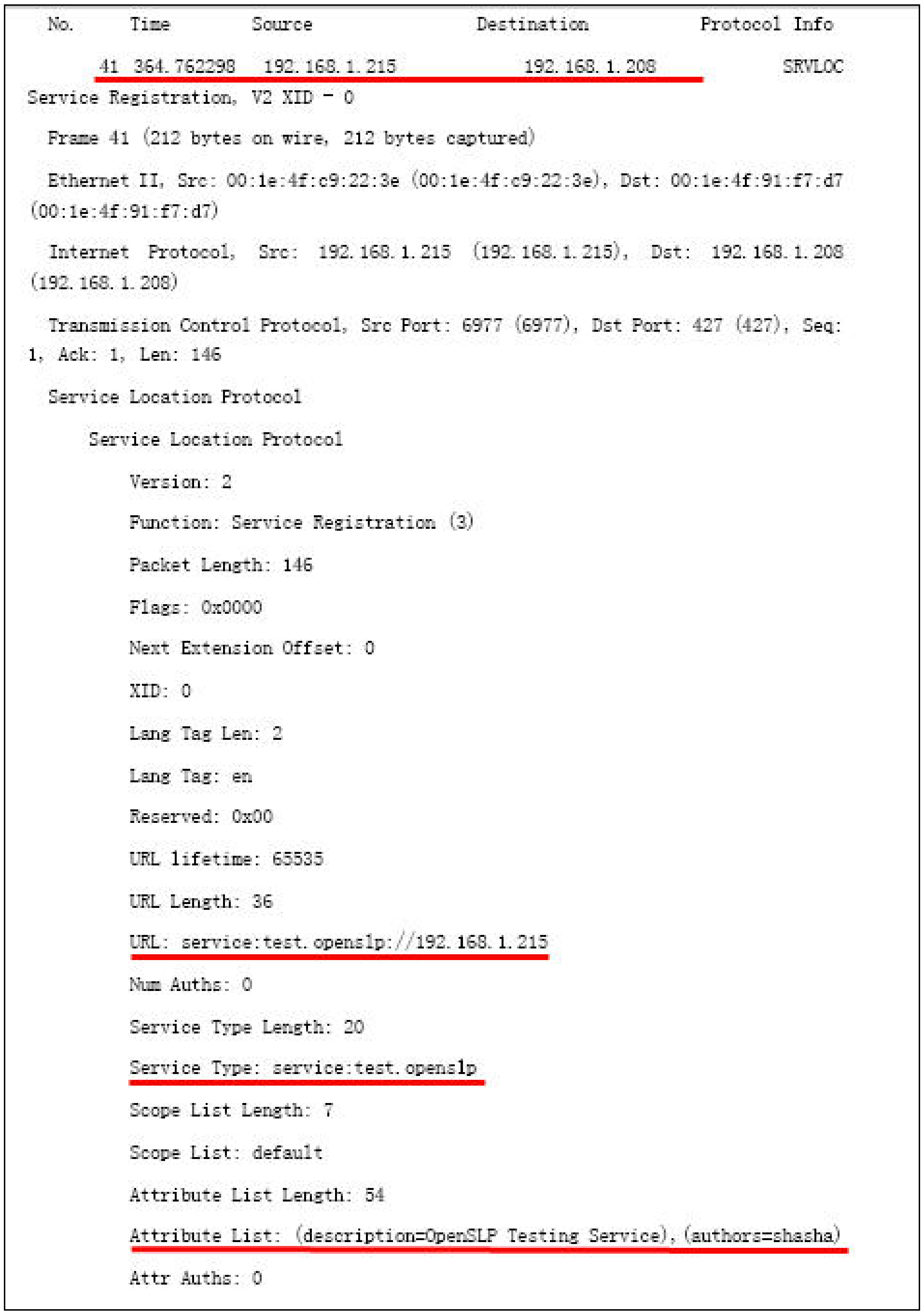
Figure 28. SLP packet between SA and DA on two PCs

Two of these packets will be shown in detail. These were used as examples for next tests because of the uniform packets format. The red line marks the important information in the packet.



```
  No.     Time        Source                    Destination             Protocol Info
       30  349.964807   192.168.1.215             255.255.255.255          SRVLOC
Service Request, V2 XID - 3583


  Frame 30 (91 bytes on wire, 91 bytes captured)
  Ethernet  II,  Src:  00:1e:4f:c9:22:3e  (00:1e:4f:c9:22:3e),  Dst:  Broadcast
(ff:ff:ff:ff:ff:ff)
  Internet  Protocol,  Src:  192.168.1.215  (192.168.1.215),  Dst:  255.255.255.255
(255.255.255.255)
  User Datagram Protocol, Src Port: 1755 (1755), Dst Port: 427 (427)
  Service Location Protocol
      Version: 2
      Function: Service Request (1)
      Packet Length: 49
      Flags: 0x2000
      Next Extension Offset: 0
      XID: 3583
      Lang Tag Len: 2
      Lang Tag: en
      Previous Response List Length: 0
      Previous Response List:
      Service Type Length: 23
      Service Type List: service:directory-agent
      Scope List Length: 0
      Scope List:
      Predicate Length: 0
      Predicate:
      SLP SPI Length: 0
      SLP SPI:
```

Figure 29. A Service Request packet sent by "hlllab5"



```
  No.     Time        Source                Destination           Protocol  Info

     41  364.762298   192.168.1.215              192.168.1.208             SRVLOC
Service Registration, V2 XID - 0

  Frame 41 (212 bytes on wire, 212 bytes captured)

  Ethernet II, Src: 00:1e:4f:c9:22:3e (00:1e:4f:c9:22:3e), Dst: 00:1e:4f:91:f7:d7
(00:1e:4f:91:f7:d7)

  Internet  Protocol,  Src:  192.168.1.215  (192.168.1.215),  Dst:  192.168.1.208
(192.168.1.208)

  Transmission Control Protocol, Src Port: 6977 (6977), Dst Port: 427 (427), Seq:
1, Ack: 1, Len: 146

  Service Location Protocol

      Service Location Protocol

          Version: 2

          Function: Service Registration (3)

          Packet Length: 146

          Flags: 0x0000

          Next Extension Offset: 0

          XID: 0

          Lang Tag Len: 2

          Lang Tag: en

          Reserved: 0x00

          URL lifetime: 65535

          URL Length: 36

          URL: service:test.openslp://192.168.1.215

          Num Auths: 0

          Service Type Length: 20

          Service Type: service:test.openslp

          Scope List Length: 7

          Scope List: default

          Attribute List Length: 54

          Attribute List: (description=OpenSLP Testing Service),(authors=shasha)

          Attr Auths: 0
```

Figure 30. A Service Registration packet sent by "hlllab5"

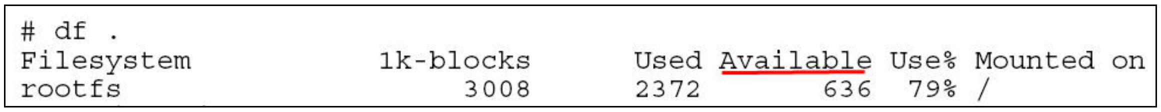
4.3 DA on Badge and SA on PC

After the SLP protocol was tested running between two linux PCs, I moved the DA to the badge. This required some changes the source code of OpenSLP daemon "slpd" to enable it to be cross compiled (these were much like the changes made to udhcpd). When the OpenSLP installation script is executed, the daemon, libraries, and configuration files are copied to various directories. I manually copied these files to the correct directories on the badge (Table 4-1).

Table 4-1. Openslp file copied

Reheat	Badge
Openslp-1.2.1/slpd/slpd	==> /usr/sbin/slpd
Openslp-1.2.1/libslp/libslp.so	==> /usr/lib/libslp.so
Openslp-1.2.1/libslp/libslp.a	==> /usr/lib/libslp.a
Openslp-1.2.1/etc/slp.conf	==> /etc/slp.conf
Openslp-1.2.1/etc/slp.reg	==> /etc/slp.reg

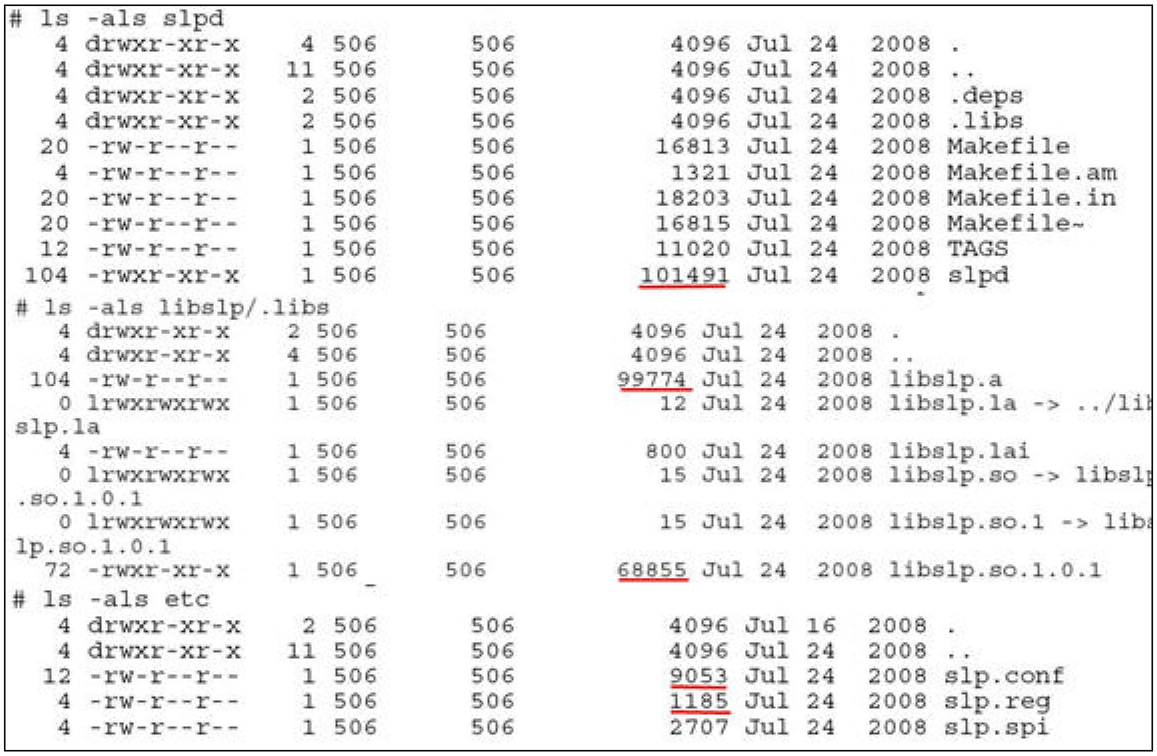
It must be noted that space must be available to copy these files to (keeping in mind that the badge has a limited memory). If there is not sufficient space, there is a risk of corrupting the FLASH file system, which will require reloading the initial FLASH file system and redoing all of the configuration steps! The figure shows the amount free space on the badge, just after a new FLASH file system was installed and the startbadge file placed in my home directory.



```
# df .
Filesystem           1k-blocks      Used Available Use% Mounted on
rootfs                    3008      2372       636  79% /
```

Figure 31. Available space on the badge before installing the SLP related files.

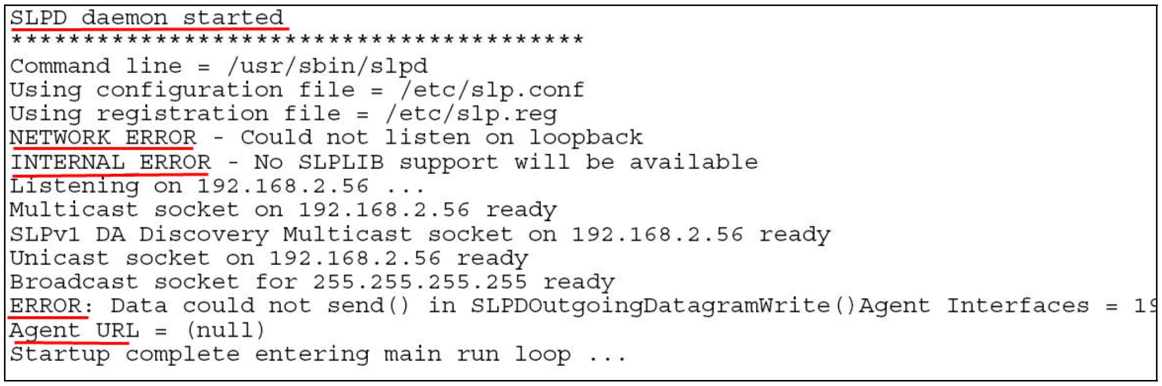
By listing the directory for slpd, we can estimate the amount of storage required by daemon, library files, and configuration files. The directory listing is shown below, with the byte size of the necessary files underlined in red.



```
# ls -als slpd
   4 drwxr-xr-x    4 506      506          4096 Jul 24  2008 .
   4 drwxr-xr-x   11 506      506          4096 Jul 24  2008 ..
   4 drwxr-xr-x    2 506      506          4096 Jul 24  2008 .deps
   4 drwxr-xr-x    2 506      506          4096 Jul 24  2008 .libs
  20 -rw-r--r--    1 506      506         16813 Jul 24  2008 Makefile
   4 -rw-r--r--    1 506      506          1321 Jul 24  2008 Makefile.am
  20 -rw-r--r--    1 506      506         18203 Jul 24  2008 Makefile.in
  20 -rw-r--r--    1 506      506         16815 Jul 24  2008 Makefile~
  12 -rw-r--r--    1 506      506         11020 Jul 24  2008 TAGS
 104 -rwxr-xr-x    1 506      506        101491 Jul 24  2008 slpd
# ls -als libslp/.libs
   4 drwxr-xr-x    2 506      506          4096 Jul 24  2008 .
   4 drwxr-xr-x    4 506      506          4096 Jul 24  2008 ..
 104 -rw-r--r--    1 506      506         99774 Jul 24  2008 libslp.a
   0 lrwxrwxrwx    1 506      506            12 Jul 24  2008 libslp.la -> ../li
slp.la
   4 -rw-r--r--    1 506      506           800 Jul 24  2008 libslp.lai
   0 lrwxrwxrwx    1 506      506            15 Jul 24  2008 libslp.so -> libsl
.so.1.0.1
   0 lrwxrwxrwx    1 506      506            15 Jul 24  2008 libslp.so.1 -> libs
lp.so.1.0.1
  72 -rwxr-xr-x    1 506      506         68855 Jul 24  2008 libslp.so.1.0.1
# ls -als etc
   4 drwxr-xr-x    2 506      506          4096 Jul 16  2008 .
   4 drwxr-xr-x   11 506      506          4096 Jul 24  2008 ..
  12 -rw-r--r--    1 506      506          9053 Jul 24  2008 slp.conf
   4 -rw-r--r--    1 506      506          1185 Jul 24  2008 slp.reg
   4 -rw-r--r--    1 506      506          2707 Jul 24  2008 slp.spi
```

Figure 32. Daemon, library files, and configuration files memory requirement

Therefore, these files can be copied to the badge because the total amount storage required is less than the 636k bytes which are available. After copying these files, the OpenSLP daemon "slpd" can be started on the badge. Because this daemon should run as a DA, the badge's configuration file should be changed as the same was on the PC which was acting as a DA – **except that the interface address has to be changed to the IP address assigned to the badge.** Finally, the daemon starts successfully, but there are some errors exist when I check the file "slpd.log" under "/var/log/" on the badge. The differences from the correct log file (from ccslex4) in the previous test, are underlined in red:



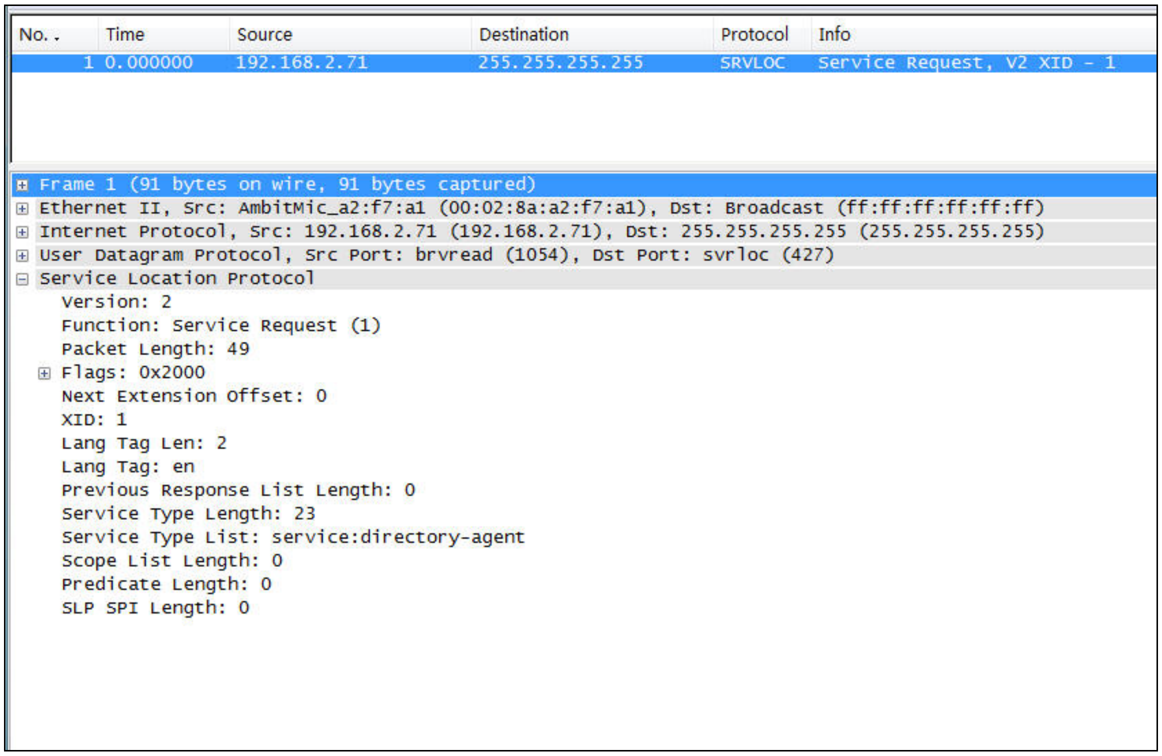
```
SLPD daemon started
***************************************
Command line = /usr/sbin/slpd
Using configuration file = /etc/slp.conf
Using registration file = /etc/slp.reg
NETWORK ERROR - Could not listen on loopback
INTERNAL ERROR - No SLPLIB support will be available
Listening on 192.168.2.56 ...
Multicast socket on 192.168.2.56 ready
SLPv1 DA Discovery Multicast socket on 192.168.2.56 ready
Unicast socket on 192.168.2.56 ready
Broadcast socket for 255.255.255.255 ready
ERROR: Data could not send() in SLPDOutgoingDatagramWrite()Agent Interfaces = 1
Agent URL = (null)
Startup complete entering main run loop ...
```

Figure 33. "slpd.log" file when Badge run DA

Unfortunately I have not yet found the reason why the DA on the badge does not work. This should be a part of any future work.

4.4 DA on PC and SA program on PDA

To test my SA program on the PDA, a test network environment needed to be set up. Because the PDA connects via WLAN to an access point, i.e. wireless local area network “ece8883” another PC was configured to act as a DA on the 192.168.2/24 network, as the PDA’s IP address is “192.168.2.71”. The DA on “ccelex4” could not be used because it is connected to a different network and the limited broadcast (255.255.255.255) will not be forwarded from one subnet to another. Therefore, another linux PC was configured with the IP address “192.168.2.4”. However, the DA from “ccslex4” did not work on this new PC. I do not yet understand why. These too remain as part of future work. However, the SA program on H5550 successfully sends the DA service request packet to broadcast address “255.255.255.255”, as show in Figure 34. SA program sends the Service Request. SA program source code is in Appendix B.



```
No. .   Time        Source          Destination      Protocol  Info
      1 0.000000    192.168.2.71    255.255.255.255  SRVLOC    Service Request, V2 XID - 1

⊞ Frame 1 (91 bytes on wire, 91 bytes captured)
⊞ Ethernet II, Src: AmbitMic_a2:f7:a1 (00:02:8a:a2:f7:a1), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol, Src: 192.168.2.71 (192.168.2.71), Dst: 255.255.255.255 (255.255.255.255)
⊞ User Datagram Protocol, Src Port: brvread (1054), Dst Port: svrloc (427)
⊟ Service Location Protocol
    Version: 2
    Function: Service Request (1)
    Packet Length: 49
  ⊞ Flags: 0x2000
    Next Extension Offset: 0
    XID: 1
    Lang Tag Len: 2
    Lang Tag: en
    Previous Response List Length: 0
    Service Type Length: 23
    Service Type List: service:directory-agent
    Scope List Length: 0
    Predicate Length: 0
    SLP SPI Length: 0
```

Figure 34. SA program sends the Service Request

4.5 DA on Badge and SA program on PDA

Because of the tests have not been completed, testing can not be continued, but must be a part of future work.

5 Conclusions

5.1 Conclusions

As mentioned above, the thesis contains two parts: discovery service implementation and remote desktop control implementation. For the discovery service implementation, I successfully build the whole device aggregation network environment, implemented SLP between two PC and sent packet with a PDA acting as a SA. The whole system architecture is shown in Figure 35. In the future additional intelligent devices such as a camera, Bluetooth headset, and GPS receiver should be able to enter into this network, be assigned an address, and discover the available services. Each new intelligent device should act as SA (in the same manner as the PDA in this thesis).

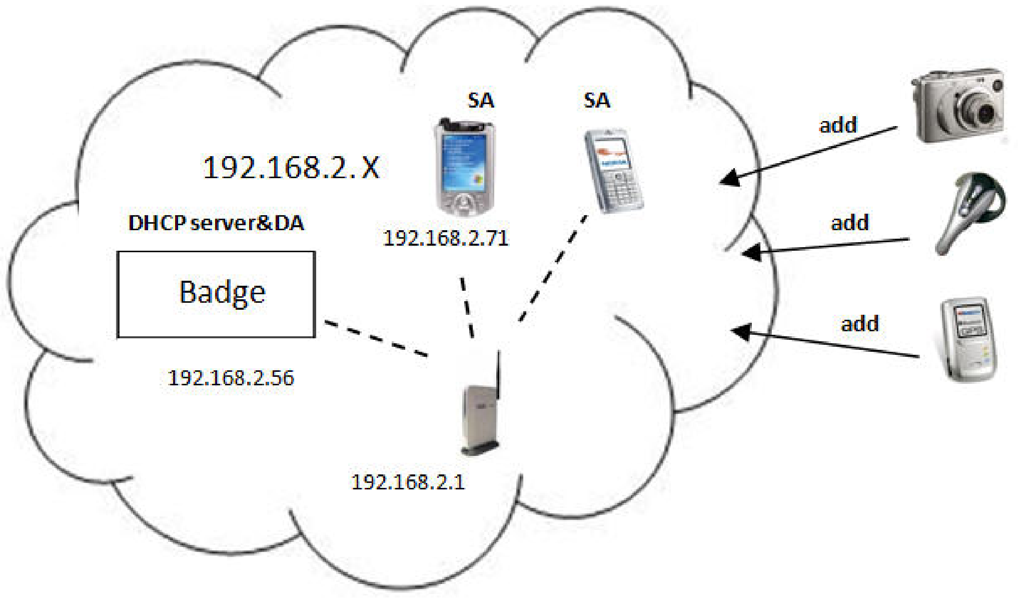


Figure 35. Personal area network for device aggregation

In a wireless local area network, the DHCP server will assign each new device and address either automatically from a pool of addresses or statically. This success of the DHCP server is meaningful for not only this implementation, but also because an IP address is essential for each network-device. I have only test a H5550 PDA and a Linux PC as DHCPs clients. The DHCP client for a Symbian cellular phone has not been tested yet.

I have not completed testing of the discovery service on the badge, because a number of implementation steps could be followed, affecting the subsequent testing. However, the SLP protocol has been implemented successfully on two PCs and the DA almost runs successfully on the badge. However, there remain some errors which cause the badge to not send DA packets.

To demonstrate an SA program on an intelligent device, I have programmed the necessary functions on a PDA running Microsoft's Pocket PC 2003. However, only one function of the two functions has been successfully tested at the present time.

All methods and tools used are introduced in the thesis, so that students who continue to work on this thesis topic will have a good starting point. The following sections will describe some suggested future work in detail.

5.2 Future work

Device aggregation with data networking will be a direction of development for the next generation communication. Several scenarios were described in Chapter 2. The remaining future work for successful device aggregation is significant and will require many years to achieve the full potential of device aggregation. This thesis will hopefully serve as a first step toward this future.

5.2.1 Improvement discovery service

As a first step, the implementation and testing of discovery service between the badge and the PDA should be finished. The OpenSLP daemon "slpd" on the badge must run without errors and the test should be done and the packets exchanged captured. Because the badge connects to a WLAN, another PC with a wireless network card is necessary to capture the WLAN traffic between the badge and the SA; as this traffic unicast will not be forwarded through the WLAN access point. The completion of the SA program is as important a task as a correctly running DA on the badge. The debugging work should be continued in coordination with test between the SA on the PDA and a DA on a linux PC. Once the DA Service Request and Service registration packets are sent successfully, then the SA program on the PDA is finished. This will leave the last test between the DA on the badge and the SA on PDA – which should work successful once each of the two pass the earlier tests.

The discovery service needs to be made available on other intelligent devices, such as cellular phones. As in this thesis, the first step will be to see that there is a DHCP client on the cell phone, as it will need its own IP address in the WLAN. A Symbian version of the program developed for the PDA needs to be created. As a result of this thesis project, this step should be easier as the functionality of this SA is well known and quite limited. For each different type of device, another SA program is necessary; but their functions are quiet similar.

Another important step will be to write an SA program that can use the service information from a Bluetooth device's service discover to generate SLP service registrations. This step will require some research and design to determine what information should be registered and how this information would be used.

5.2.2 Combining the discovery service and remote desktop control application

Remote desktop application research has not yet been completed, but communication between several of the devices was successfully shown. Therefore, a simple application should be developed which combines device and service discovery with the remote desktop application. This will require a SLP library to which client applications can link. This library should provide client applications with a simple interface for accessing SLP registered service information. Note that the UA could be run on another PC or run on each intelligent device (as each device can initiate client applications). A test is scenario is shown in the following three pictures. When a new device enters the wireless local area network, it will request an IP address and register its services with the DA running on the badge. When the remote desktop control application is started on the cellular phone, a UA running on the cellular phone will query the DA for a bigger display service. If such a service has been registered, then the cellular phone will get a service reply from the DA running on the badge telling it the address to connect connected to (for example the address of the PDA) as a result the cellular phone's desktop can be displayed on the bigger display. Thus it will bring the project to a successful finish.

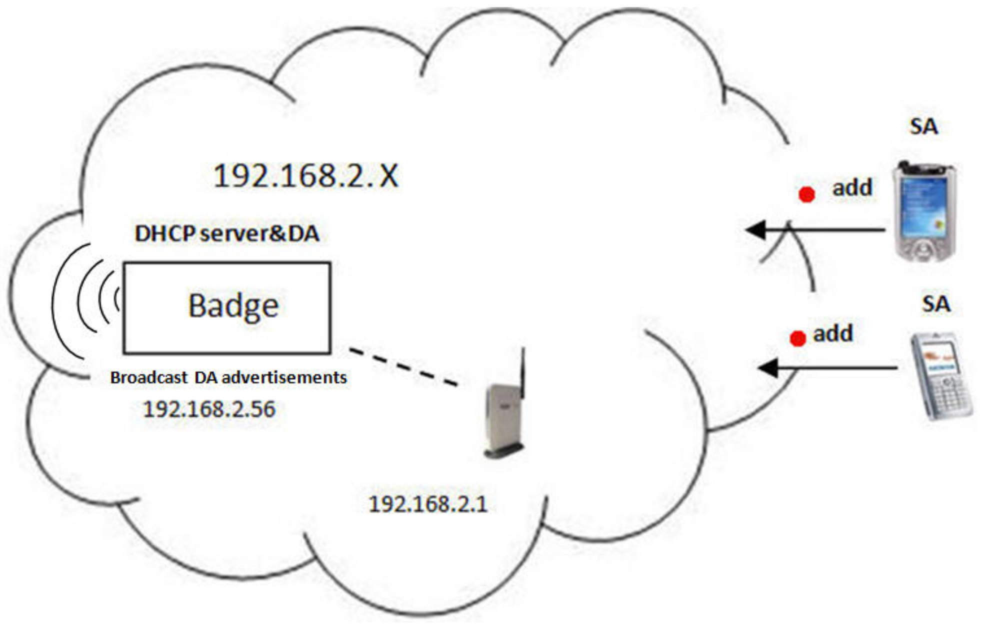


Figure 36. New devices enter into the network

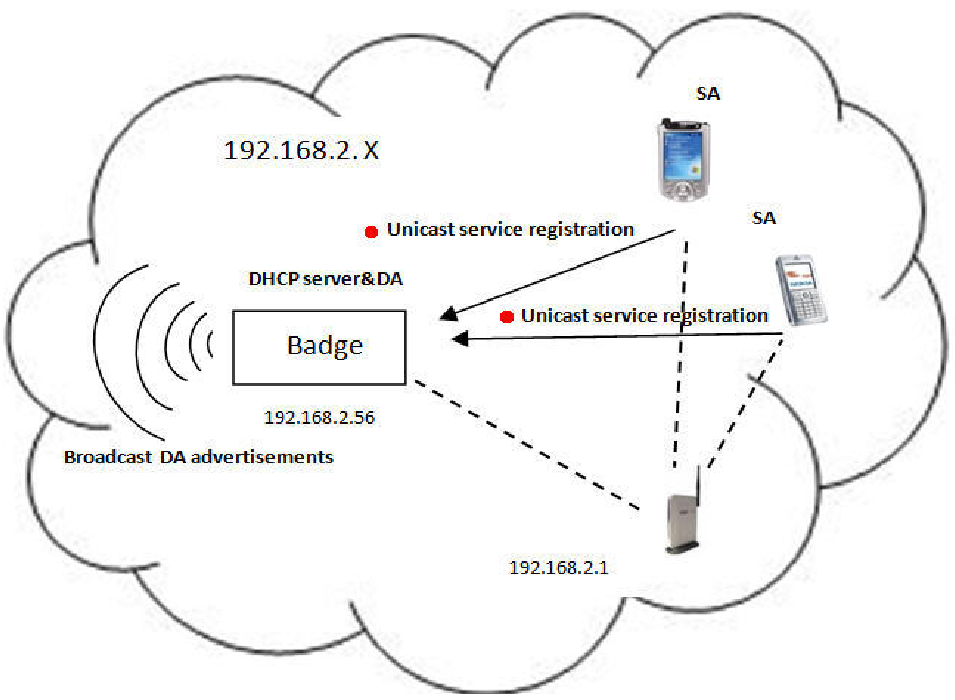

192.168.2.X
SA
SA
Unicast service registration
Unicast service registration
DHCP server&DA
Badge
192.168.2.56
Broadcast DA advertisements
192.168.2.1

Figure 37. Devices register service

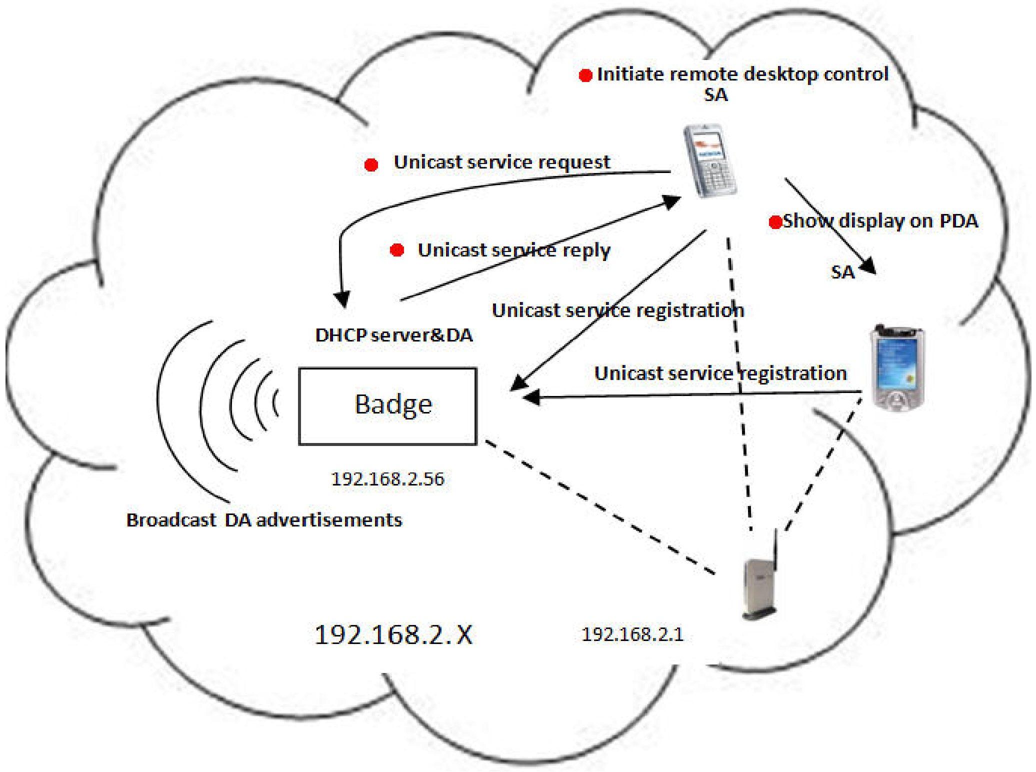


Figure 38. PDA acts as a bigger display to the cellular phone

5.2.3 Security

Security is very important point for any system architecture and its role in device aggregation is clear. Two important security aspects which should be solved in a future thesis are: (1) authentication is required when new devices enter into the personal area network (in order to protect the network) and (2) authentication is also required when one device want to connect to another device which could provide a certain service for it. Thus it will be important to understand how to configure devices so that the owner can decide who can use them.

5.2.4 Power management

Power management (such as PSM [19]) to reduce the power consumed in wireless LANs could be explored in the future. Because the devices used in the device aggregation examined in this thesis used a local PAN their power consumption will be low – if and only if they can sleep much of the time and they can know when to wake up. An area of increasing interested for the future is high speed wide area interfaces (such as devices using 3G cellular networks). An interesting question is to what extent these devices can use their wide area interface while still consuming only limited power.

References

[1] Mark Weiser, ”Ubiquitous Computing” March, 1996, available at: http://www.ubiq.com/hypertext/weiser/UbiHome.html, last visited April 2008

[2] Bill N. Schilit and Uttam Sengupta, “Device ensembles”, published by the IEEE Computer Society, 2004

[3] “Universal Serial Bus”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Universal_Serial_Bus, last modified on 4 May, 2008

[4] “What is Firewire”, Available at: http://www.tech-faq.com/firewire.shtml, last visited at April, 2008

[5] “What is bluetooth”, Available at: http://www.cycnet.com/cms/2004/englishcorner/digest/kepu/200512/t20051208_42170.htm, last visited at April, 2008

[6] “Wi-Fi”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Wi-Fi, Last modified on 1 May 2008

[7] “What is Jini”, Jini.org, Available at: http://www.jini.org/wiki/What_is_Jini%3F, last modified on 8 March 2007

[8] “Jini Network Technology”, Sun Microsystems, Inc, 2001, Available at: www.sun.com/jini.

[9] “Universal Plug and Play”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Universal_Plug_and_Play, last modified on 27 April 2008

[10] “Service Location Protocol”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Service_Location_Protocol, last modified on 4th September 2008

[11] E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service location Protocol, Version 2”, IETF RFC 2608, June 1999. Available at: http://www.ietf.org/rfc/rfc2608.txt

[12] “Peer to peer”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Peer-to-peer#Classifications_of_peer-to-peer_networks, last modified on May 2008

[13] “Cross weaver”, Regents of the University of California, Available at: http://guir.berkeley.edu/projects/crossweaver/, last updated Sunday December 21 2003

[14] “Remote desktop software”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Remote_desktop_software, last modified on 26 April 2008

[15] “Virtual Network Computing”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Virtual_Network_Computing, last modified on 1 May 2008

[16] “Remote desktop Protocol”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Remote_Desktop_Protocol, last modified on 4 May 2008

[17] “Remote desktop Protocol”, From Microsoft develop network, Available at: http://msdn.microsoft.com/en-us/library/aa383015.aspx

[18] “X Window System”, Form Wikipedia, Available at: http://en.wikipedia.org/wiki/X_Window_System, last modified on 2 May 2008

[19] Kevin Klues, “Power Management in Wireless Networks”, 9 May 2006, Available at: http://www.cse.wustl.edu/~jain.

[20] Oscar_Santillana, “RTP redirection using a handheld device with Minisip”, Masters Thesis, Department of Communication Systems, School of Information and Communication Technology, Royal Institute of Technology (KTH); Stockholm, Sweden, March 2007 http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/070301-Oscar_Santillana-FinalVersion-with-cover.pdf

[21] “Personal digital assistant”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Personal_digital_assistant, last modified on 12th September 2008

[22] Gerald Q. Maguire Jr., “Hewlett-Packard Company (HP) iPAQ Pocket PC h5550: Specifications”, Available at: http://web.it.kth.se/~maguire/ipaq-notes.html, Latest update 8 August 2008

[23] “Windows Mobile”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Windows_Mobile#Windows_Mobile_2003 , last modified on 14th September 2008

[24] Chris De Herrera, Windows Mobile 2003. Pocket PC Magazine. Retrieved 14 September 2007.

[25] “Nokia E70”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Nokia_E70, last modified on 9th September 2008

[26] Mat Hans, Christopher Hoover, Gerald Q. Maguire Jr., and Mark Smith. “Badge 4 Embedded Development Kit Bastille Day Release”, Available at: https://www.ece.gatech.edu/mailman/lisinfo/hp_badge4.

[27] “Cross compiler”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Cross_compiler, last modified on 12th September 2008

[28] “Dynamic Host Configuration Protocol”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Dhcp, last modified on 13th September 2008

[29] “udhcp Server/Client Package”, Available at: http://udhcp.busybox.net/. 4th December 2004.

[30] Sam C. Lin, PocktDHCP V0.22, Available at: http://www.lincomatic.com/wireless/pocketdhcp.zip, 6 July 2002

[31] “Ubiquitous Computing Using SIP”, Stefan Berger, Henning Schulzrinne, Stylianos Sidiroglou, Xiaoto Wu, Department of Computer Science in Colombia University NewYork, June 2003

[32] “An Introduction to the Service Location Protocol (SLP)”, Caldera Systems. Inc,Novell. Inc, Available at: http://www.openslp.org/doc/html/IntroductionToSLP/index.html.

[33] E.Guttman, C.Perkins, and S. Kaplan, “Service Location Protocol RFC2165”, J.Veizades, IETF RFC 2165, June 1997

[34] “Microsoft Visual Studio”, From Wikipedia, Available at: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio , last modified on 11th September 2008

[35] David Sabaté Mogica, “Remote Desktop”, a Batchelors thesis project report, Department of Communication Systems, School of Information and Communication Technology, Royal Institute of Technology (KTH); Stockholm, Sweden, (draft) Fall 2007

[36] Haruumi Shiode, In-building Location Sensing Based on WLAN Signal Strength: Realizing a Presence User Agent Masters Thesis, Department of Communication Systems, School of Information and Communication Technology, Royal Institute of Technology (KTH); Stockholm, Sweden, March 2007
http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080314-Haruumi_Shiode-with-cover.pdf

[37] Yu Sun, Context-aware applications for a Pocket PC, Department of Communication Systems, School of Information and Communication Technology, Royal Institute of Technology (KTH); Stockholm, Sweden, December 2007
http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/071220-Sun_Yu-with-cover.pdf

Appendix A. Source code of Udhcp-0.9.6 *with changes*

- dhcpd.h

```
/* dhcpd.h */
#ifndef _DHCPD_H
#define _DHCPD_H

#include <netinet/ip.h>
#include <netinet/udp.h>

#include "leases.h"

/************************************/
/* Defaults _you_ may want to tweak */
/************************************/

/* the period of time the client is allowed to use that address */
#define LEASE_TIME              (60*60*24*10) /* 10 days of seconds */

/* where to find the DHCP server configuration file */
#define DHCPD_CONF_FILE         "/etc/udhcpd.conf"

/*****************************************************************/
/* Do not modify below here unless you know what you are doing!! */
/*****************************************************************/

/* DHCP protocol -- see RFC 2131 */
#define SERVER_PORT             67
#define CLIENT_PORT             68

#define DHCP_MAGIC              0x63825363

/* DHCP option codes (partial list) */
#define DHCP_PADDING                    0x00
#define DHCP_SUBNET                     0x01
#define DHCP_TIME_OFFSET        0x02
#define DHCP_ROUTER                     0x03
#define DHCP_TIME_SERVER        0x04
#define DHCP_NAME_SERVER0x05
#define DHCP_DNS_SERVER                 0x06
#define DHCP_LOG_SERVER                 0x07
#define DHCP_COOKIE_SERVER              0x08
#define DHCP_LPR_SERVER                 0x09
#define DHCP_HOST_NAME                  0x0c
#define DHCP_BOOT_SIZE                  0x0d
#define DHCP_DOMAIN_NAME                0x0f
#define DHCP_SWAP_SERVER 0x10
#define DHCP_ROOT_PATH                  0x11
#define DHCP_IP_TTL             0x17
#define DHCP_MTU                0x1a
#define DHCP_BROADCAST                  0x1c
```

Appendix A.

```
#define DHCP_NTP_SERVER         0x2a
#define DHCP_WINS_SERVER  0x2c
#define DHCP_REQUESTED_IP 0x32
#define DHCP_LEASE_TIME         0x33
#define DHCP_OPTION_OVER  0x34
#define DHCP_MESSAGE_TYPE       0x35
#define DHCP_SERVER_ID          0x36
#define DHCP_PARAM_REQ          0x37
#define DHCP_MESSAGE            0x38
#define DHCP_MAX_SIZE           0x39
#define DHCP_T1                 0x3a
#define DHCP_T2                 0x3b
#define DHCP_VENDOR             0x3c
#define DHCP_CLIENT_ID          0x3d

#define DHCP_END          0xFF


#define BOOTREQUEST             1
#define BOOTREPLY         2

#define ETH_10MB          1
#define ETH_10MB_LEN            6

#define DHCPDISCOVER            1
#define DHCPOFFER         2
#define DHCPREQUEST             3
#define DHCPDECLINE       4
#define DHCPACK                 5
#define DHCPNAK                 6
#define DHCPRELEASE             7
#define DHCPINFORM        8

#define BROADCAST_FLAG          0x8000

#define OPTION_FIELD      0
#define FILE_FIELD        1
#define SNAME_FIELD       2

/* miscellaneous defines */
#define TRUE              1
#define FALSE             0
#define MAC_BCAST_ADDR          "\xff\xff\xff\xff\xff\xff"
#define OPT_CODE 0
#define OPT_LEN 1

struct option_set {
        unsigned char *data;
        struct option_set *next;
};

struct server_config_t {
        u_int32_t server;       /* Our IP, in network order */
```

```
	u_int32_t start;		/* Start address of leases, network order */
	u_int32_t end;			/* End of leases, network order */
	struct option_set *options;	/* List of DHCP options loaded from the config file */
	char *interface;		/* The name of the interface to use */
	int ifindex;			/* Index number of the interface to use */
	unsigned char arp[6];		/* Our arp address */
	unsigned long lease;		/* lease time in seconds (host order) */
	unsigned long max_leases;	/* maximum number of leases (including reserved address)
*/
	char remaining;		/* should the lease file be interpreted as lease time remaining, or
					 * as the time the lease expires */
	unsigned long auto_time;	/* how long should udhcpd wait before writing a config file.
					 * if this is zero, it will only write one on SIGUSR1 */
	unsigned long decline_time;	/* how long an address is reserved if a client returns a
					 * decline message */
	unsigned long conflict_time;	/* how long an arp conflict offender is leased for */
	unsigned long offer_time;	/* how long an offered address is reserved */
	unsigned long min_lease;	/* minimum lease a client can request*/
	char *lease_file;
	char *pidfile;
	char *notify_file;		/* What to run whenever leases are written */
	u_int32_t siaddr;		/* next server bootp option */
	char *sname;			/* bootp server name */
	char *boot_file;		/* bootp boot file option */
};

extern struct server_config_t server_config;
extern struct dhcpOfferedAddr *leases;

#ifdef NEVER
struct host_config_t {
	int hostindex;			/* Index number of this host */
	u_int32_t hostip;		/* Our IP, in network order */
	unsigned char arp[6];		/* Our arp address */
      char *hname;			/* host name */
};


#define MAX_hosts 10
#endif

#endif
```

Appendix A.

- dhcpd.c

```
/* dhcpd.c
 *
 * Moreton Bay DHCP Server
 * Copyright (C) 1999 Matthew Ramsay <matthewr@moreton.com.au>
 *                    Chris Trew <ctrew@moreton.com.au>
 *
 * Rewrite by Russ Dill <Russ.Dill@asu.edu> July 2001
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <time.h>
#include <sys/time.h>

#include "debug.h"
#include "dhcpd.h"
#include "arpping.h"
#include "socket.h"
#include "options.h"
#include "files.h"
#include "leases.h"
#include "packet.h"
#include "serverpacket.h"
#include "pidfile.h"
```

Appendix A.

```
/* globals */
struct dhcpOfferedAddr *leases;
struct server_config_t server_config;

int initial_host_index=0;


/* Exit and cleanup */
static void exit_server(int retval)
{
        pidfile_delete(server_config.pidfile);
        CLOSE_LOG();
        exit(retval);
}


/* SIGTERM handler */
static void udhcpd_killed(int sig)
{
        sig = 0;
        LOG(LOG_INFO, "Received SIGTERM");
        exit_server(0);
}


static void print_leases(void)
{
  unsigned long i;
  int j;
  struct dhcpOfferedAddr *lease;
  time_t curr = time(0);

  struct in_addr addr;
  u_int32_t expires;

  for (i = 0; i < server_config.max_leases; i++) {
    printf("lease [%d]:", (int) i);
    lease = &leases[i];

    for (j = 0; j < 6; j++) {
      printf("%02x", lease->chaddr[j]);
      if (j != 5) printf(":");
    }

    addr.s_addr = lease->yiaddr;
    printf(" ip: %-15s", inet_ntoa(addr));
    expires = lease->expires;

    if (!expires)
      printf("expired\n");
    else {
      expires=expires-curr;
```

```
	if (expires > 60*60*24) {
		printf("%d days, ", expires / (60*60*24));
		expires %= 60*60*24;
	}
	if (expires > 60*60) {
		printf("%d hours, ", expires / (60*60));
		expires %= 60*60;
	}
	if (expires > 60) {
		printf("%d minutes, ", expires / 60);
		expires %= 60;
	}
	printf("%d seconds\n", expires);
    }

  }
}


#ifdef COMBINED_BINARY
int udhcpd(int argc, char *argv[])
#else
int main(int argc, char *argv[])
#endif
{
		fd_set rfds;
		struct timeval tv;
		int server_socket;
		int bytes, retval;
		struct dhcpMessage packet;
		unsigned char *state;
		char *server_id, *requested;
		u_int32_t server_id_align, requested_align;
		unsigned long timeout_end;
		struct option_set *option;
		struct dhcpOfferedAddr *lease;
		struct sockaddr_in *sin;
		int pid_fd;

		/* server ip addr */
		int fd = -1;
		struct ifreq ifr;

		argc = argv[0][0]; /* get rid of some warnings */

		OPEN_LOG("udhcpd");
		LOG(LOG_INFO, "udhcp server (v%s) started", VERSION);

		pid_fd = pidfile_acquire(server_config.pidfile);
		pidfile_write_release(pid_fd);

		memset(&server_config, 0, sizeof(struct server_config_t));
	read_config(DHCPD_CONF_FILE, 1); /* parse the configuration file the first time */
	if ((option = find_option(server_config.options, DHCP_LEASE_TIME))) {
		memcpy(&server_config.lease, option->data + 2, 4);
		server_config.lease = ntohl(server_config.lease);
	}
	else server_config.lease = LEASE_TIME;

	leases = malloc(sizeof(struct dhcpOfferedAddr) * server_config.max_leases);
	memset(leases, 0, sizeof(struct dhcpOfferedAddr) * server_config.max_leases);
	read_leases(server_config.lease_file);

	read_config(DHCPD_CONF_FILE, 2); /* parse the configuration file the second time */
#ifdef DEBUG
	print_leases();
#endif

	if((fd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) >= 0) {
		ifr.ifr_addr.sa_family = AF_INET;
		strcpy(ifr.ifr_name, server_config.interface);
		if (ioctl(fd, SIOCGIFADDR, &ifr) == 0) {
			sin = (struct sockaddr_in *) &ifr.ifr_addr;
			server_config.server = sin->sin_addr.s_addr;
			DEBUG(LOG_INFO, "%s (server_ip) = %s", ifr.ifr_name, inet_ntoa(sin-
>sin_addr));
		} else {
			LOG(LOG_ERR, "SIOCGIFADDR failed!");
			exit_server(1);
		}
		if (ioctl(fd, SIOCGIFINDEX, &ifr) == 0) {
			DEBUG(LOG_INFO, "adapter index %d", ifr.ifr_ifindex);
			server_config.ifindex = ifr.ifr_ifindex;
		} else {
			LOG(LOG_ERR, "SIOCGIFINDEX failed!");
			exit_server(1);
		}
		if (ioctl(fd, SIOCGIFHWADDR, &ifr) == 0) {
			memcpy(server_config.arp, ifr.ifr_hwaddr.sa_data, 6);
			DEBUG(LOG_INFO, "adapter hardware address
%02x:%02x:%02x:%02x:%02x:%02x",
				server_config.arp[0], server_config.arp[1], server_config.arp[2],
				server_config.arp[3], server_config.arp[4], server_config.arp[5]);
		} else {
			LOG(LOG_ERR, "SIOCGIFHWADDR failed!");
			exit_server(1);
		}
	} else {
		LOG(LOG_ERR, "socket failed!");
		exit_server(1);
	}
	close(fd);
```

Appendix A.

```
#ifndef DEBUGGING
	pid_fd = pidfile_acquire(server_config.pidfile); /* hold lock during fork. */
	switch(fork()) {
	case -1:
		perror("fork");
		exit_server(1);
		/*NOTREACHED*/
	case 0:
		break; /* child continues */
	default:
		exit(0); /* parent exits */
		/*NOTREACHED*/
		}
	close(0);
	setsid();
	pidfile_write_release(pid_fd);
#endif


	signal(SIGUSR1, write_leases);
	signal(SIGTERM, udhcpd_killed);

	timeout_end = time(0) + server_config.auto_time;
	while(1) { /* loop until universe collapses */

		server_socket = listen_socket(INADDR_ANY, SERVER_PORT, 
server_config.interface);
		if(server_socket == -1) {
			LOG(LOG_ERR, "couldn't create server socket -- au revoir");
			exit_server(0);
		}

		FD_ZERO(&rfds);
		FD_SET(server_socket, &rfds);
		if (server_config.auto_time) {
			tv.tv_sec = timeout_end - time(0);
			if (tv.tv_sec <= 0) {
				tv.tv_sec = server_config.auto_time;
				timeout_end = time(0) + server_config.auto_time;
				write_leases(0);
			}
			tv.tv_usec = 0;
		}
		retval = select(server_socket + 1, &rfds, NULL, NULL, server_config.auto_time ? 
&tv : NULL);
		if (retval == 0) {
			write_leases(0);
			timeout_end = time(0) + server_config.auto_time;
			close(server_socket);
			continue;
		} else if (retval < 0) {
			DEBUG(LOG_INFO, "error on select");
			close(server_socket);
```

```
				continue;
			}

			bytes = get_packet(&packet, server_socket); /* this waits for a packet - idle */
			close(server_socket);
			if(bytes < 0)
				continue;

			if((state = get_option(&packet, DHCP_MESSAGE_TYPE)) == NULL) {
				DEBUG(LOG_ERR, "couldnt get option from packet -- ignoring");
				continue;
			}

			lease = find_lease_by_chaddr(packet.chaddr);
			switch (state[0]) {
			case DHCPDISCOVER:
				DEBUG(LOG_INFO,"received DISCOVER");

				if (sendOffer(&packet) < 0) {
					LOG(LOG_ERR, "send OFFER failed -- ignoring");
				}
				break;
			case DHCPREQUEST:
				DEBUG(LOG_INFO,"received REQUEST");

				requested = get_option(&packet, DHCP_REQUESTED_IP);
				server_id = get_option(&packet, DHCP_SERVER_ID);

				if (requested) memcpy(&requested_align, requested, 4);
				if (server_id) memcpy(&server_id_align, server_id, 4);

				if (lease) {
					if (server_id) {
						/* SELECTING State */
						DEBUG(LOG_INFO, "server_id = %08x",
ntohl(server_id_align));
						if (server_id_align == server_config.server && requested
&&
						    requested_align == lease->yiaddr) {
							sendACK(&packet, lease->yiaddr);
						}
					} else {
						if (requested) {
							/* INIT-REBOOT State */
							if (lease->yiaddr == requested_align)
								sendACK(&packet, lease->yiaddr);
							else sendNAK(&packet);
						} else {
							/* RENEWING or REBINDING State */
							if (lease->yiaddr == packet.ciaddr)
								sendACK(&packet, lease->yiaddr);
							else {
								/* don't know what to do!!!! */
```

```
                                                        sendNAK(&packet);
                                                }
                                        }
                                }
                        } /* else remain silent */
                    print_leases();
                     break;
                case DHCPDECLINE:
                        DEBUG(LOG_INFO,"received DECLINE");
                        if (lease) {
                                memset(lease->chaddr, 0, 16);
                                lease->expires = time(0) + server_config.decline_time;
                        }
                        break;
                case DHCPRELEASE:
                        DEBUG(LOG_INFO,"received RELEASE");
                        if (lease) lease->expires = time(0);
                        break;
                case DHCPINFORM:
                        DEBUG(LOG_INFO,"received INFORM");
                        send_inform(&packet);
                        break;
                default:
                        LOG(LOG_WARNING, "unsupported DHCP message (%02x) -- ignoring",
state[0]);
                }
        }
        return 0;
}
```

Appendix B. Source code of SA for the H5550

- FontList.h

```
//======================================================================
// Header file
//
// Written for the book Programming Windows CE
// Copyright (C) 2001 Douglas Boling
//======================================================================
// Returns number of elements
#ifndef __FontList__
#define __FontList__

#include "Global_Var.h"

//----------------------------------------------------------------------
// Generic defines and data types
//
struct decodeUINT {                          // Structure associates
    UINT Code;                               // messages
                                             // with a function.
    LRESULT (*Fxn)(HWND, UINT, WPARAM, LPARAM);
};
struct decodeCMD {                           // Structure associates
    UINT Code;                               // menu IDs with a
    LRESULT (*Fxn)(HWND, WORD, HWND, WORD);  // function.
};



//----------------------------------------------------------------------
// Window prototypes and defines
//

typedef struct {
    int nNumFonts;
    TCHAR szFontFamily[LF_FACESIZE];
} FONTFAMSTRUCT;

typedef FONTFAMSTRUCT *PFONTFAMSTRUCT;

typedef struct {
    INT yCurrent;
    HDC hdc;
} PAINTFONTINFO;
typedef PAINTFONTINFO *PPAINTFONTINFO;



int InitClient (HINSTANCE);
int TermClient (HINSTANCE, int);
```

```
//----------------------------------------------------------------------
// Function prototypes
//
int InitApp (HINSTANCE);
HWND InitInstance (HINSTANCE, LPWSTR, int);
int TermInstance (HINSTANCE, int);

// Window procedures
LRESULT CALLBACK FrameWndProc (HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK ClientWndProc (HWND, UINT, WPARAM, LPARAM);

// Message handlers
LRESULT DoCreateFrame (HWND, UINT, WPARAM, LPARAM);
LRESULT DoSizeFrame (HWND, UINT, WPARAM, LPARAM);
LRESULT DoDestroyFrame (HWND, UINT, WPARAM, LPARAM);

LRESULT DoCreateClient (HWND, UINT, WPARAM, LPARAM);
LRESULT DoPaintClient (HWND, UINT, WPARAM, LPARAM);
LRESULT DoVScrollClient (HWND, UINT, WPARAM, LPARAM);
LRESULT DoSLPMain (HWND, UINT, WPARAM, LPARAM);

void printsz(HWND hWnd, TCHAR szOut[]);

#define MYMSG_STARTSLP   (WM_USER + 10)

#define dim(x) (sizeof(x) / sizeof(x[0]))

#define CLIENTWINDOW    TEXT ("ClientWnd")    // Client window ID

#define FAMILYMAX   24

//----------------------------------------------------------------------
// Generic defines used by application
#define  IDC_CMDBAR 1                     // Command bar ID
#define  IDC_CLIENT 2

#endif
```

Appendix B.

- Global_var.h

```
 #ifndef __GLOBAL_VAR__
#define __GLOBAL_VAR__
#include <windows.h>

extern HINSTANCE hInst;
extern BOOL fFirst;
extern BOOL fReallyFirst;
//extern PAINTFONTINFO global_pfi;
extern INT global_nFontHeight;
extern INT yCurrent;
extern BOOL SLPfirst;
extern TCHAR global_szOut[10240];
extern INT offset;
extern RECT global_rect;
extern HANDLE g_hThread;
extern DWORD   dwThreadID;
//extern SCROLLINFO si;
#endif
```

Appendix B.

- SLP_Buf.h

```
#if(!defined SLP_BUFFER_H_INCLUDED)
#define SLP_BUFFER_H_INCLUDED

/*=========================================================================
==*/
typedef struct _SLPBuffer
/*=========================================================================
==*/
{
   unsigned char*   start;
   /* ALWAYS points to the start of the xmalloc() buffer  */

   unsigned char*   curpos;
   /* "slider" pointer.  Range is ALWAYS (start < curpos < end) */

   unsigned char*   end;
   /* ALWAYS set to point to the byte after the last meaningful byte */
   /* Data beyond this index may not be valid */
}SLPBuffer,*PSLPBuffer;

/*=========================================================================
==*/
typedef struct _SLPRegParams
/* Used to pass parameters to functions that deals with handle based SLP   */
/* API calls                                                 */
/*=========================================================================
==*/
{
   int            lifetime;
   //int            fresh;    //used in header
   int            urllen;
   const char*     url;
          //int                    numauths;
   int            srvtypelen;
   const char*     srvtype;
   int            scopelistlen;
   const char*     scopelist;
   int            attrlistlen;
   const char*     attrlist;
          //int                    attrauths;
   //SLPRegReport*   callback;
   //void*          cookie;
}SLPRegParams,*PSLPRegParams;

/*=========================================================================
==*/
typedef struct _SLPSrvReq
// havent been used
/*=========================================================================
==*/
{
          //int                    multicast;              //used in header
```

```
        int                         prlistlen;
        int                         srvtypelen;
        const char*           srvtypelist;
        int                         scopelistlen;
        int                         predlen;
        int                         slpspilen;
}SLPSrvReq,*PSLPSrvReq;


#endif
```

Appendix B.

- SLP_misc.h

```
#ifndef __SLPMISC__
#define __SLPMISC__

void ToUINT16(unsigned char *charptr, unsigned int val);
void ToUINT24(unsigned char *charptr, unsigned int val);

unsigned short AsUINT16(const char *charptr);
unsigned int AsUINT24(const char *charptr);

#endif
```

Appendix B.

- SLPMain.h

```
#ifndef __SLPMain__
#define __SLPMain__

#include <windows.h>
#include <winsock.h>
#include <malloc.h>
#include "SLP_buf.h"
#include "FontList.h"
#include "SLP_misc.h"

//#define __DEBUG__                          //when debuging, define it

#define SLP_RESERVED_PORT          427
#define SLP_TCP_PORT                         4270
#define SLP_BCAST_ADDRESS                    0xffffffff  /* 255.255.255.255 */
#define SLP_LOCAL_ADDRESS                    0xc0a80165      /* 192.168.1.101 */
#define SLP_DA_ADDRESS                       0xc0a80166      /* 192.168.1.102 */ //only for
testing
#define SLP_FLAG_OVERFLOW      0x8000
#define SLP_FLAG_FRESH         0x4000
#define SLP_FLAG_MCAST          0x2000
#define MTU                                        1400

#define xmalloc(x) malloc((x))
#define xfree(x) free((x))

int SLPMain(HWND hWnd);
int SendSrvReq(HWND hWnd, int UDPsocketfd);
int SendSrvReg(HWND hWnd, int TCPsocketfd);
int creatUDPsocket(void);
int creatTCPsocket(void);
int receiveUDP(HWND hWnd, int sockfd); /* now these two function are quite the same*/
int receiveTCP(HWND hWnd, int sockfd); /* can be removed one, later*/
int REGreader(void);
int mystrcmp(char* source, char* keyword);

#endif
```

Appendix B.

● ClientWnd.cpp

```
//======================================================================
// ClientWnd - Client window code for FontList2
//
// Written for the book Programming Windows CE
// Copyright (C) 2001 Douglas Boling

//======================================================================
#include <windows.h>                 // For all that Windows stuff
#include "FontList.h"                // Program-specific stuff
#include "SLPMain.h"

void printsz(HWND hWnd, TCHAR szOut[])
{
        RECT rect;
        GetClientRect (hWnd, &rect);
        offset += wsprintf (global_szOut+offset, szOut);
        DrawText(GetDC(hWnd),global_szOut,lstrlen(global_szOut),&rect,DT_LEFT|DT_WORDB
REAK);
}
//----------------------------------------------------------------------
// Global data
//
FONTFAMSTRUCT ffs[FAMILYMAX];
INT sFamilyCnt = 0;
INT sVPos = 0;
INT sVMax = 0;

// Message dispatch table for ClientWindowProc
const struct decodeUINT ClientMessages[] = {
   WM_CREATE, DoCreateClient,
   WM_PAINT, DoPaintClient,
   WM_VSCROLL, DoVScrollClient,
        MYMSG_STARTSLP, DoSLPMain,
};
//----------------------------------------------------------------------
// InitClient - Client window initialization
//
int InitClient (HINSTANCE hInstance) {
   WNDCLASS wc;

   // Register application client window class.
   wc.style = 0;                          // Window style
   wc.lpfnWndProc = ClientWndProc;          // Callback function
   wc.cbClsExtra = 0;                      // Extra class data
   wc.cbWndExtra = 0;                      // Extra window data
   wc.hInstance = hInstance;                // Owner handle
   wc.hIcon = NULL,                        // Application icon
   wc.hCursor = LoadCursor (NULL, IDC_ARROW);// Default cursor
   wc.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
   wc.lpszMenuName =  NULL;                 // Menu name
   wc.lpszClassName = CLIENTWINDOW;         // Window class name
```

```
    if (RegisterClass (&wc) == 0) return 1;

    return 0;
}
//----------------------------------------------------------------------
// TermClient - Client window cleanup
//

int TermClient (HINSTANCE hInstance, int nDefRC) {
    return nDefRC;
}
//======================================================================
// Font callback functions
//----------------------------------------------------------------------
// FontFamilyCallback - Callback function that enumerates the font
// families.
//
int CALLBACK FontFamilyCallback (CONST LOGFONT *lplf,
                                 CONST TEXTMETRIC *lpntm,
                                 DWORD nFontType, LPARAM lParam) {
    int rc = 1;

    // Stop enumeration if array filled.
    if (sFamilyCnt >= FAMILYMAX)
        return 0;
    // Copy face name of font.
    lstrcpy (ffs[sFamilyCnt++].szFontFamily, lplf->lfFaceName);

    return rc;
}
//----------------------------------------------------------------------
// EnumSingleFontFamily - Callback function that enumerates the font
// families
//
int CALLBACK EnumSingleFontFamily (CONST LOGFONT *lplf,
                                   CONST TEXTMETRIC *lpntm,
                                   DWORD nFontType, LPARAM lParam) {
    PFONTFAMSTRUCT pffs;

    pffs = (PFONTFAMSTRUCT) lParam;
    pffs->nNumFonts++;    // Increment count of fonts in family.
    return 1;
}
//----------------------------------------------------------------------
// PaintSingleFontFamily - Callback function that enumerates the font
// families.
//
int CALLBACK PaintSingleFontFamily (CONST LOGFONT *lplf,
                                    CONST TEXTMETRIC *lpntm,
                                    DWORD nFontType, LPARAM lParam) {
    PPAINTFONTINFO ppfi;
    TCHAR szOut[256];
    INT nFontHeight, nPointSize;
    TEXTMETRIC tm;
    HFONT hFont, hOldFont;

    ppfi = (PPAINTFONTINFO) lParam;  // Translate lParam into
                                     // structure pointer.

    // Create the font from the LOGFONT structure passed.
    hFont = CreateFontIndirect (lplf);

    // Select the font into the device context.
    hOldFont = (HFONT__ *)SelectObject (ppfi->hdc, hFont);

    // Get the height of the default font.
    GetTextMetrics (ppfi->hdc, &tm);
    nFontHeight = tm.tmHeight + tm.tmExternalLeading;

    // Compute font size.
    nPointSize = (lplf->lfHeight * 72) /
                 GetDeviceCaps(ppfi->hdc,LOGPIXELSY);

    // Format string and paint on display.
    wsprintf (szOut, TEXT ("%s   Point:%d"), lplf->lfFaceName,
             nPointSize);
    ExtTextOut (ppfi->hdc, 25, ppfi->yCurrent, 0, NULL,
               szOut, lstrlen (szOut), NULL);

    // Update new draw point.
    ppfi->yCurrent += nFontHeight;
    // Deselect font and delete.
    SelectObject (ppfi->hdc, hOldFont);
    DeleteObject (hFont);
    return 1;
}
//======================================================================
// Message handling procedures for ClientWindow
//----------------------------------------------------------------------
// ClientWndProc - Callback function for application window
//
LRESULT CALLBACK ClientWndProc (HWND hWnd, UINT wMsg, WPARAM wParam,
                                LPARAM lParam) {
    INT i;
    //
    // Search message list to see if we need to handle this
    // message. If in list, call procedure.
    //
    for (i = 0; i < dim(ClientMessages); i++) {
        if (wMsg == ClientMessages[i].Code){
                        return (*ClientMessages[i].Fxn)(hWnd, wMsg, wParam, lParam);
                }
    }

    return DefWindowProc (hWnd, wMsg, wParam, lParam);
}
```

Appendix B.

```
//----------------------------------------------------------------------
// DoCreateClient - Process WM_CREATE message for window.
//
LRESULT DoCreateClient (HWND hWnd, UINT wMsg, WPARAM wParam,
                        LPARAM lParam) {
    HDC hdc;
    INT i, rc;
            RECT *rect;

            rect = &global_rect;
            GetClientRect (hWnd, rect);
            rect->bottom = 1000;

    //Enumerate the available fonts.
    hdc = GetDC (hWnd);
    rc = EnumFontFamilies ((HDC)hdc, (LPTSTR)NULL, FontFamilyCallback, 0);

    for (i = 0; i < sFamilyCnt; i++) {
        ffs[i].nNumFonts = 0;
        rc = EnumFontFamilies ((HDC)hdc, ffs[i].szFontFamily,
                              EnumSingleFontFamily,
                              (LPARAM)(PFONTFAMSTRUCT)&ffs[i]);
    }
    ReleaseDC (hWnd, hdc);
    return 0;
}
//----------------------------------------------------------------------
// DoPaintClient - Process WM_PAINT message for window.
//
LRESULT DoPaintClient (HWND hWnd, UINT wMsg, WPARAM wParam,
                       LPARAM lParam) {

    PAINTSTRUCT ps;
    RECT *rect;
    HDC hdc;
    TEXTMETRIC tm;
    INT i;
    PAINTFONTINFO pfi;
    SCROLLINFO si;

            //pfi=&global_pfi;
    hdc = BeginPaint (hWnd, &ps);

            rect = &global_rect;
    //GetClientRect (hWnd, &rect);


    // Get the height of the default font.
    GetTextMetrics (hdc, &tm);
    global_nFontHeight = tm.tmHeight + tm.tmExternalLeading;
    // Initialize struct that is passed to enumerate function.
    yCurrent = global_rect.top - sVPos;
    pfi.hdc = hdc;
```

```
            //rect.bottom += sVPos;
            //rect.top += sVPos;

            DrawText(pfi.hdc,global_szOut,lstrlen(global_szOut),&global_rect,DT_LEFT|DT_WORDB
REAK);

            if (SLPfirst==TRUE)
            {
                        SendMessage(hWnd,MYMSG_STARTSLP,0,0);
                        SLPfirst = FALSE;
            }

    /*
            while(1)

        // Format output string and paint font family name.
        wsprintf (global_szOut, TEXT ("Hello Shasha!"));
        ExtTextOut (hdc, 5, yCurrent, 0, NULL,
                  global_szOut, lstrlen (global_szOut), NULL);
        yCurrent += global_nFontHeight;
                        Sleep(500L);
    }
            *//////////////////////////////////////////////
    // Compute the total height of the text in the window.
    if (fFirst) {
        sVPos = 0;
        sVMax = (yCurrent - rect->top) - (rect->bottom - rect->top);

        si.cbSize = sizeof (si);
        si.nMin = 0;
        si.nMax = yCurrent;
        si.nPage = rect->bottom - rect->top;
        si.nPos = sVPos;
        si.fMask = SIF_DISABLENOSCROLL; //SIF_ALL;
        SetScrollInfo (hWnd, SB_VERT, &si, TRUE);
        fFirst = FALSE;
    }
    EndPaint (hWnd, &ps);
    return 0;
}
//----------------------------------------------------------------------
// DoVScrollClient - Process WM_VSCROLL message for window.
//
LRESULT DoVScrollClient (HWND hWnd, UINT wMsg, WPARAM wParam,
                        LPARAM lParam) {
    RECT *rect;
    SCROLLINFO si;
    INT sOldPos = sVPos;

            rect = &global_rect;
    //GetClientRect (hWnd, &rect);
    switch (LOWORD (wParam)) {
    case SB_LINEUP:
        sVPos -= 10;
        break;

    case SB_LINEDOWN:
        sVPos += 10;
        break;

    case SB_PAGEUP:
        sVPos -= rect->bottom - rect->top;
        break;

    case SB_PAGEDOWN:
        sVPos += rect->bottom - rect->top;
        break;

    case SB_THUMBPOSITION:
        sVPos = HIWORD (wParam);
        break;
    }
    // Check range.
    if (sVPos < 0)
        sVPos = 0;
    if (sVPos > sVMax)
        sVPos = sVMax;

    // If scroll position changed, update scroll bar and
    // force redraw of window.
    if (sVPos != sOldPos) {
        si.cbSize = sizeof (si);
        si.nPos = sVPos;
        si.fMask = SIF_POS;
        SetScrollInfo (hWnd, SB_VERT, &si, TRUE);

        InvalidateRect (hWnd, NULL, TRUE);
    }
    return 0;

}


LRESULT DoSLPMain (HWND hWnd, UINT wMsg, WPARAM wParam,
                                             LPARAM lParam) {

              //HANDLE g_hThread;
              //DWORD   dwThreadID;
              printsz(hWnd, TEXT ("DoSLPMain!\r\n"));

    //SLPMain();

              g_hThread =
CreateThread(NULL,2048,(LPTHREAD_START_ROUTINE)SLPMain,hWnd,0,&dwThreadID);
              if (g_hThread==NULL)
              {
                      //printsz(hWnd, &global_pfi, global_nFontHeight, TEXT ("Thread Error!\r\n"));

              }
              return 0;
}
```

Appendix B.

- Fontlist.cpp

```
//======================================================================
// FontList2 - Lists the available fonts in the system
//

// Written for the book Programming Windows CE
// Copyright (C) 2001 Douglas Boling
//======================================================================
#include <windows.h>                 // For all that Windows stuff
#include <commctrl.h>                // Command bar includes
#include "FontList.h"              // Program-specific stuff
//----------------------------------------------------------------------
// Global data
//
const TCHAR szAppName[] = TEXT ("FontList2");


// Message dispatch table for FrameWindowProc
const struct decodeUINT FrameMessages[] = {
   WM_SIZE, DoSizeFrame,
   WM_CREATE, DoCreateFrame,
   WM_DESTROY, DoDestroyFrame,
};
//======================================================================
// Program entry point
//
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPWSTR lpCmdLine, int nCmdShow) {
   MSG msg;
   int rc = 0;
   HWND hwndFrame;

   // Initialize application.
   rc = InitApp (hInstance);
   if (rc) return rc;

   // Initialize this instance.
   hwndFrame = InitInstance (hInstance, lpCmdLine, nCmdShow);
   if (hwndFrame == 0)
      return 0x10;

   // Application message loop
   while (GetMessage (&msg, NULL, 0, 0)) {
      TranslateMessage (&msg);
      DispatchMessage (&msg);
   }
   // Instance cleanup
   return TermInstance (hInstance, msg.wParam);
}
//----------------------------------------------------------------------
// InitApp - Application initialization
//
```

Appendix B.

```
int InitApp (HINSTANCE hInstance) {
    WNDCLASS wc;

#if defined(WIN32_PLATFORM_PSPC)
    // If Pocket PC, allow only one instance of the application.
    HWND hWnd = FindWindow (szAppName, NULL);
    if (hWnd) {
        SetForegroundWindow ((HWND)(((DWORD)hWnd) | 0x01));
        return -1;
    }
#endif
    // Register application frame window class.
    wc.style = 0;                         // Window style
    wc.lpfnWndProc = FrameWndProc;           // Callback function
    wc.cbClsExtra = 0;                       // Extra class data
    wc.cbWndExtra = 0;                       // Extra window data
    wc.hInstance = hInstance;                // Owner handle
    wc.hIcon = NULL,                         // Application icon
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);// Default cursor
    wc.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
    wc.lpszMenuName =  NULL;                  // Menu name
    wc.lpszClassName = szAppName;             // Window class name

    if (RegisterClass (&wc) == 0) return 1;

    // Initialize client window class.
    if (InitClient (hInstance) != 0) return 2;
    return 0;
}
//----------------------------------------------------------------------
// InitInstance - Instance initialization
//
HWND InitInstance (HINSTANCE hInstance, LPWSTR lpCmdLine, int nCmdShow) {
    HWND hWnd;

    // Save program instance handle in global variable.
    hInst = hInstance;

    // Create frame window.
    hWnd = CreateWindow (szAppName, TEXT ("Font List 2"), WS_VISIBLE,
                        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, NULL, NULL, hInstance, NULL);
    // Return fail code if window not created.
    if (!IsWindow (hWnd)) return 0;

    // Standard show and update calls
    ShowWindow (hWnd, nCmdShow);

    UpdateWindow (hWnd);
    return hWnd;
}
//----------------------------------------------------------------------
// TermInstance - Program cleanup
```

```
//
int TermInstance (HINSTANCE hInstance, int nDefRC) {

    return nDefRC;
}
//======================================================================
// Message handling procedures for FrameWindow
//----------------------------------------------------------------------
// FrameWndProc - Callback function for application window
//
LRESULT CALLBACK FrameWndProc (HWND hWnd, UINT wMsg, WPARAM wParam,
                               LPARAM lParam) {
    INT i;
    //
    // Search message list to see if we need to handle this
    // message. If in list, call procedure.
    //
    for (i = 0; i < dim(FrameMessages); i++) {
        if (wMsg == FrameMessages[i].Code)
            return (*FrameMessages[i].Fxn)(hWnd, wMsg, wParam, lParam);
    }
    return DefWindowProc (hWnd, wMsg, wParam, lParam);
}
//----------------------------------------------------------------------
// DoCreateFrame - Process WM_CREATE message for window.
//
LRESULT DoCreateFrame (HWND hWnd, UINT wMsg, WPARAM wParam,
                       LPARAM lParam) {
    HWND hwndCB, hwndClient;
    INT sHeight;
    LPCREATESTRUCT lpcs;

    // Convert lParam to pointer to create structure.
    lpcs = (LPCREATESTRUCT) lParam;

    // Create a command bar.
    hwndCB = CommandBar_Create (hInst, hWnd, IDC_CMDBAR);
    // Add exit button to command bar.
    CommandBar_AddAdornments (hwndCB, 0, 0);
    sHeight = CommandBar_Height (GetDlgItem (hWnd, IDC_CMDBAR));

    //
    // Create client window.  Size it so that it fits under
    // the command bar and fills the remaining client area.
    //
    hwndClient = CreateWindow (CLIENTWINDOW, TEXT (""),
                               WS_VISIBLE | WS_CHILD | WS_VSCROLL,
                               lpcs->x, lpcs->y + sHeight,
                               lpcs->cx, lpcs->cy - sHeight,
                               hWnd, (HMENU)IDC_CLIENT,
                               lpcs->hInstance, NULL);

    // Destroy frame if client window not created.
    if (!IsWindow (hwndClient))
        DestroyWindow (hWnd);
    return 0;
}
//----------------------------------------------------------------------
// DoSizeFrame - Process WM_SIZE message for window.
//
LRESULT DoSizeFrame (HWND hWnd, UINT wMsg, WPARAM wParam, LPARAM lParam) {
    RECT rect;
    INT i;

    GetClientRect (hWnd, &rect);
    i = CommandBar_Height (GetDlgItem (hWnd, IDC_CMDBAR));
    rect.top += i;

    SetWindowPos (GetDlgItem (hWnd, IDC_CLIENT), NULL, rect.left, rect.top,
                  rect.right - rect.left, rect.bottom - rect.top,
                  SWP_NOZORDER);
    return 0;
}
//----------------------------------------------------------------------
// DoDestroyFrame - Process WM_DESTROY message for window.
//
LRESULT DoDestroyFrame (HWND hWnd, UINT wMsg, WPARAM wParam,
                        LPARAM lParam) {
    PostQuitMessage (0);
    return 0;
}
```

Appendix B.

- Global_Var.cpp

```
#include "Global_Var.h"
//**************FontList.cpp ClientWnd.cpp*********************
HINSTANCE hInst;                    // Program instance handle

//********************ClientWnd.cpp**************************
BOOL fFirst = TRUE;
BOOL fReallyFirst = TRUE;
//PAINTFONTINFO global_pfi;
INT global_nFontHeight;
INT yCurrent;
BOOL SLPfirst = TRUE;
TCHAR global_szOut[10240];
INT offset = 0;
RECT global_rect;
HANDLE g_hThread;
DWORD   dwThreadID;
//SCROLLINFO si;
```

Appendix B.

- SLP_misc.cpp

```
#include "SLP_misc.h"

/*----------------------------------------------------------------------*/
void ToUINT16(unsigned char *charptr, unsigned int val)
/*----------------------------------------------------------------------*/
{
   charptr[0] = (val >> 8) & 0xff;
   charptr[1] = val & 0xff;
}

/*----------------------------------------------------------------------*/
void ToUINT24(unsigned char *charptr, unsigned int val)
/*----------------------------------------------------------------------*/
{
   charptr[0] = (val >> 16) & 0xff;
   charptr[1] = (val >> 8) & 0xff;
   charptr[2] = val & 0xff;
}


/*----------------------------------------------------------------------*/
unsigned short AsUINT16(const char *charptr)
/*----------------------------------------------------------------------*/
{
   unsigned char *ucp = (unsigned char *) charptr;
   return(ucp[0] << 8) | ucp[1];
}

/*----------------------------------------------------------------------*/
unsigned int AsUINT24(const char *charptr)
/*----------------------------------------------------------------------*/
{
   unsigned char *ucp = (unsigned char *) charptr;
   return(ucp[0] << 16) | (ucp[1] << 8) |  ucp[2];
}
```

Appendix B.

- SLPMain.cpp

```
#include "SLPMain.h"

int XID;
char REGbuf[256];
int regsrv;         // how many services have been read in REG file

int SLPMain(HWND hWnd)
/* problems about, XID, and socket connect, should be solved later */
/* remember to close socket at the end */
{
        int FuncID;
        int UDPsockfd;
        int TCPsockfd;
        TCHAR Message[128];

        //creat a UDP socket
        UDPsockfd = creatUDPsocket();
        //creat a TCP socket
        TCPsockfd = creatTCPsocket();
/*
        while(1)
        {
SEND_DA_REQ:
                SendSrvReq(hWnd,UDPsockfd);
                FuncID = receiveUDP(hWnd,UDPsockfd);
                if (FuncID == 8) //DA Advertisement
                {
                        wsprintf (Message, TEXT ("FuncID is %d\r\n"), FuncID);
                        printsz(hWnd, Message);
                        SendMessage(hWnd,WM_PAINT,0,0);
                }
                else
                {
                        wsprintf (Message, TEXT ("FuncID is %d\r\n"), FuncID);
                        printsz(hWnd, Message);
                        SendMessage(hWnd,WM_PAINT,0,0);

                        Sleep(20000L);
                        goto SEND_DA_REQ;
                }

REG_SERVICE:
                SendSrvReg(hWnd,TCPsockfd);
                FuncID = receiveTCP(hWnd,TCPsockfd);
                wsprintf (Message, TEXT ("FuncID is %d\r\n"), FuncID);
                printsz(hWnd, Message);
                SendMessage(hWnd,WM_PAINT,0,0);

                goto REG_SERVICE;
        }
        */
        /******** Main algrithem **********
        while(1)
        {
SEND_DA_REQ:
                sendto(Req,UDP Bcast);
                receive(UDP);
                if (!got DA)
                {
                        Delay some time;
                        goto SEND_DA_REQ;
                }
                else
                {
                        if needed update DA list;
                }
REG_SERVICE:
                if (new DA)
                {
                        if (buf = assemble(REG) == NULL)
                        {
                                goto NOTHING_TO_DO;
                        }
                        sendto(Reg,buf,TCP);
                        receive(TCP);
                        if (ACK)
                        {
                                goto REG_SERVICE;
                        }
                        else
                        {
                                delay some time;
                                goto REG_SERVICE;
                        }
                }
NOTHING_TO_DO:
                delay some time;
        }
        ************************************/
        //closesocket(TCPsockfd);
        //closesocket(UDPsockfd);
        return 0;


}

/*====================================================================
==*/
int creatUDPsocket(void)
/* Creates a socket and provides a peeraddr to send to                  */
/*                                                                       */
/* peeraddr        (OUT) pointer to receive the connected DA's address   */
/*                                                                       */
/* Returns         Valid socket or -1 if no DA connection can be made    */
```

```
/*=====================================================================
==*/
{
   int     sockfd;
   BOOL    on = 1;                      //broadcast model, 0 is off
           int                   iMode = 0;         //blocking model, 0 is block

   /* setup broadcast */
   sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
   if(sockfd >= 0)
   {
      if(setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, (char*)&on, sizeof(on)))        //allow
broadcast
      {
         return -1;
      }
                     if(ioctlsocket(sockfd,FIONBIO, (u_long FAR*)&iMode))   //nonblocking
                     {
                               return -2;
                     }
   }

   return sockfd;
}

/*=====================================================================
==*/
int creatTCPsocket(void)
/* Creates a socket and provides a peeraddr to send to                  */
/*                                                      */
/* peeraddr         (OUT) pointer to receive the connected DA's address    */
/*                                                      */
/* Returns          Valid socket or -1 if no DA connection can be made     */
/*=====================================================================
==*/
{
   int     sockfd;
           int iSocketError;

   sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

   if(sockfd == INVALID_SOCKET)
           {
                     iSocketError = WSAGetLastError();
                     return -1;
           }


   return sockfd;
}

/*=====================================================================
==*/
```

Appendix B.

```
int SendSrvReq(HWND hWnd, int UDPsocketfd)
/* return socket handle for receive function                    */
/* input hWnd for print function                         */
/*======================================================================
==*/
{
        //Set the target device parameters
        SOCKADDR_IN sTargetDevice;
        memset(&sTargetDevice, 0, sizeof(SOCKADDR_IN));
        sTargetDevice.sin_family = AF_INET;
   sTargetDevice.sin_port = htons(SLP_RESERVED_PORT);
   sTargetDevice.sin_addr.s_addr = htonl(SLP_LOCAL_ADDRESS);

        //assemble a SrvReq package first
        int     result;
        PSLPBuffer pslpbuffer;
        pslpbuffer = (PSLPBuffer)xmalloc(sizeof(struct _SLPBuffer) + MTU + 1);

        /*************************
        // goto CLEANUP will skip the
        // initialization, so define it
        // before the goto
        *************************/
        char* srvtype = "service:directory-agent";
        char* langtag = "en";
        int bodysize=0;

        if (pslpbuffer)
        {
                pslpbuffer->start = (unsigned char*)(pslpbuffer + 1);
                pslpbuffer->curpos= pslpbuffer->start;
                pslpbuffer->end   = pslpbuffer->start + MTU;
        }
        else
        {
                result = -1;
                goto CLEANUP;
        }


        /*************************
        // make body for SrvReq
        *************************/
        // I only implement a simple model, here you need to
        // re-calculate the length of the buf, and the fulfill
        // the content of the SrvReq package.
        unsigned char* body;
        //int bufsize=0;   //define upper
        bodysize += 2;    //PRList len = 0
        bodysize += 2;    //srv type len = 23;
        //char* srvtype = "service:directory-agent";  //define upper
        int srvtypelen;
        srvtypelen = strlen(srvtype);
        bodysize += 6;   //Scopelist len = 0; Predicate len = 0; SLP SPI len = 0;
        bodysize += srvtypelen;

        body = (unsigned char*)xmalloc(bodysize);
        if (body<=0)
        {
                result = -2;
                goto CLEANUP;
        }
        memset(body,0,bodysize);
        /*PRList len*/
        *(body)                     = 0;
        /*srv type len*/
        ToUINT16(body+2, 23);
        /*srv type*/
        memcpy(body+4,srvtype,srvtypelen);
        /*Scopelist len*/
        ToUINT16(body+4+srvtypelen, 0);
        /*Predicate len*/
        ToUINT16(body+6+srvtypelen, 0);
        /*SLP SPI len*/
        ToUINT16(body+8+srvtypelen, 0);


        /*************************
        // make header for SrvReq
        *************************/
        //char* langtag = "en"; //define upper
        int langtaglen;
        langtaglen = strlen(langtag);
        int headsize;
        headsize = langtaglen + 14;

        /*version*/
        *(pslpbuffer->start)      = 2;
        /*function id*/
  *(pslpbuffer->start + 1)   = 1;
        /*length*/
        ToUINT24(pslpbuffer->start + 2, bodysize+headsize);
        /*flags*/
        ToUINT16(pslpbuffer->start + 5, SLP_FLAG_MCAST);
        /*ext offset*/
        ToUINT24(pslpbuffer->start + 7, 0);
        /*xid*/
        ToUINT16(pslpbuffer->start + 10, (XID+1)%65535);
        /*lang tag len*/
  ToUINT16(pslpbuffer->start + 12, 2);
  /*lang tag*/
  memcpy(pslpbuffer->start + 14, langtag, langtaglen);

        pslpbuffer->curpos = pslpbuffer->start + langtaglen + 14 ;   //curpos is point to the start of the
body
```

```
	/************************
	// combine header and body
	************************/
	memcpy(pslpbuffer->curpos, body, bodysize);
	pslpbuffer->curpos += bodysize;

	/************************
	// send it by UDP BroadCast
	************************/
	//SOCKADDR_IN sTargetDevice;
	//memset(&sTargetDevice, 0, sizeof(SOCKADDR_IN));
	//int UDPsocketfd;
	//UDPsocketfd = SLPNetworkConnectToBroadcast(&sTargetDevice);
	int nBytesSent;
	char* UDPbuf;
	int UDPbufsize;
	TCHAR Message[256];

	UDPbufsize = bodysize+headsize;
	UDPbuf = (char*)xmalloc(UDPbufsize);
	if (UDPbuf<=0)
	{
		result = -3;
		goto CLEANUP;
	}
	memset(UDPbuf,0,UDPbufsize);
	memcpy(UDPbuf, pslpbuffer->start, UDPbufsize);


#ifdef __DEBUG__			//just comment off the while(1) and {}
	while(1)
	{						//when debuging, we loop the package
#endif
		nBytesSent = sendto(UDPsocketfd, UDPbuf, UDPbufsize, 0,
			(SOCKADDR*)&sTargetDevice,
			sizeof(SOCKADDR_IN));
		wsprintf (Message, TEXT ("Send Package, Len is %d\r\n"), nBytesSent);
		printsz(hWnd, Message);
		SendMessage(hWnd,WM_PAINT,0,0);
#ifdef __DEBUG__
		Sleep(10000L);
	}
#endif

	result = 0; // everything is ok
CLEANUP:
	/*Free memory here*/
	if (pslpbuffer)
	{
		xfree(pslpbuffer);
	}
	else if (body)
```

```
        {
                xfree(body);
        }
        else if (UDPbuf)
        {
                xfree(UDPbuf);
        }

        if (result!=0)
        {
                wsprintf (Message, TEXT ("Error Code is %d\r\n"), result);
                printsz(hWnd, Message);
                SendMessage(hWnd,WM_PAINT,0,0);
        }

        return result;
}
/*=====================================================================
==*/
int receiveUDP(HWND hWnd, int sockfd)
/* return Function ID of the received package                    */
/*=====================================================================
==*/
{
        int result=0;
        // set the receive end parameters
        SOCKADDR_IN sReceiveFromAddr;
        memset(&sReceiveFromAddr, 0, sizeof(SOCKADDR_IN));

        sReceiveFromAddr.sin_family = AF_INET;
        sReceiveFromAddr.sin_port = htons(SLP_RESERVED_PORT);
        sReceiveFromAddr.sin_addr.s_addr = htonl(SLP_LOCAL_ADDRESS);

        /******************************
        * Read the body of the package
        ******************************/
        char* head;
        head = (char*)xmalloc(16);        //read the header of the package
        memset(head,0,16);
        int nBytesRecv = 0;
        int nReceiveAddrSize = 0;
        int FuncID = 0;

        if (head)
        {
                nBytesRecv = recvfrom(sockfd, head, 16, 0,              //need to set non block
model
                                        (SOCKADDR *)&sReceiveFromAddr,     //when
creat the socket
                                        &nReceiveAddrSize);
        }
        if (nBytesRecv<=0)
        {
                result = GetLastError();
                goto CLEANUP;
        }
        FuncID = *(head + 1);

        /******************************
        * Read the body of the package
        ******************************/
        int bodylen;
        PSLPBuffer body;                //need to use end, and curpos
                                                        //no useful here, maybe used later
        /* check the version */
  if(nBytesRecv >= 5 && *head == 2)
  {
     /* allocate the recvmsg big enough for the whole message */
     bodylen = AsUINT24(head + 2);
     /* one byte is minimum */
     if (bodylen <= 0)
        bodylen = 1;
     body = (PSLPBuffer)xmalloc(sizeof(struct _SLPBuffer) + bodylen + 1);
                memset(body,0,sizeof(struct _SLPBuffer) + bodylen + 1);
     if(body)
     {
                        body->start = (unsigned char*)body + 1;
                        body->curpos = body->start;
                        body->end = body->start + bodylen;

        while(body->curpos < body->end)
        {
           if(nBytesRecv > 0)
           {
              nBytesRecv = recv(sockfd,
                        (char*)body->curpos,
                        body->end - body->curpos,
                        0);
              if(nBytesRecv > 0)
              {
                 body->curpos = body->curpos + nBytesRecv;
              }
              else
              {
                 //errno = ENOTCONN;
                                                result = -2;
                 goto CLEANUP;
              }
           }
           else if(nBytesRecv == 0)
           {
              //errno = ETIMEDOUT;
                                        result = -3;
              goto CLEANUP;
           }
           else
```

```
            {
              //errno =  ENOTCONN;
                                                result = -4;
              goto CLEANUP;
            }
          } /* end of main read while. */
        }
        else
        {
          //errno = ENOMEM;
                                result = -5;
          goto CLEANUP;
        }
    }
    else
    {
        //errno = EINVAL;
                        result = -6;
        goto CLEANUP;
    }
#ifdef __DEBUG__
        if (result>=0)
        {
                TCHAR Message[128];
                wsprintf (Message, TEXT ("Function ID is %d\r\n"), result);
                printsz(hWnd, Message);
                SendMessage(hWnd,WM_PAINT,0,0);
        }
#endif
        // everything is ok, return the Function ID
        result = FuncID;

CLEANUP:
        if (head)
        {
                xfree(head);
        }
        if (body)
        {
                xfree(body);
        }

        return result;
}

/*=====================================================================
==*/
int SendSrvReg(HWND hWnd, int TCPsocketfd)
/* return socket handle for receive function                    */
/* input hWnd for print function                               */
/*=====================================================================
==*/
{
        int iSocketError;
        int     result;
        int bodysize;
        int entrysize;
        int nBytesSent;
        int nBytesIndex;
        int nBytesLeft;
        int lowat;

        //Set the target device parameters
        SOCKADDR_IN sTargetDevice;
        memset(&sTargetDevice, 0, sizeof(SOCKADDR_IN));
        sTargetDevice.sin_family = AF_INET;
  sTargetDevice.sin_port = htons(SLP_RESERVED_PORT);
  sTargetDevice.sin_addr.s_addr = htonl(SLP_DA_ADDRESS);    //need to get from last step

        // Connect with a valid socket
        // only can connect once!! need to add a sign to guarantee that !!!
        if(connect(TCPsocketfd, (SOCKADDR *)&sTargetDevice, sizeof(sTargetDevice)) == 0)
        {
                /* set the receive and send buffer low water mark to 18 bytes
       (the length of the smallest slpv2 message) */
                lowat = 18;

        setsockopt(TCPsocketfd,SOL_SOCKET,SO_RCVLOWAT,(char*)&lowat,sizeof(lowat));

        setsockopt(TCPsocketfd,SOL_SOCKET,SO_SNDLOWAT,(char*)&lowat,sizeof(lowat));
        }
        else
        {
                result = WSAGetLastError();
                return result;
        }

        //assemble a SrvReq package first

        PSLPBuffer pslpbuffer;
        pslpbuffer = (PSLPBuffer)xmalloc(sizeof(struct _SLPBuffer) + MTU + 1);

        /*************************
        // goto CLEANUP will skip the
        // initialization, so define it
        // before the goto
        *************************/
        //because of goto, need to define the char* varaible carefully (the location of the code)
        //need to read from REG file
        //i.e. readattr("attribute")
        //and in REG file, we can write like this
        //attribute:(description = OpenSLP Testing Service);
        //use : and ; to divide
        //or just do what they have done in the REG file, I mean the same way.
        char* url = "service:test.openslp://192.168.1.215";
        char* srvtype = "service:test.openslp";
```

```
char* scope       = "default";
char* attribute = "(description = OpenSLP Testing Service)";
char* langtag = "en";


if (pslpbuffer)
{
        pslpbuffer->start = (unsigned char*)(pslpbuffer + 1);
        pslpbuffer->curpos= pslpbuffer->start;
        pslpbuffer->end   = pslpbuffer->start + MTU;
}
else
{
        result = -1;
        goto CLEANUP;
}


/*************************
// make body for SrvReq
*************************/
// I only implement a simple model, here you need to
// re-calculate the length of the buf, and the fulfill
// the content of the SrvReq package.

//creat a URL entry first
unsigned char* URLentry;
entrysize = 0;
entrysize += 1; //reserved
entrysize += 2; //lifetime
entrysize += 2; //URL len
int urllen;
urllen = strlen(url);
entrysize += urllen;
entrysize += 1;   //Auths

URLentry = (unsigned char*)xmalloc(entrysize);
if (URLentry<=0)
{
        result = -2;
        goto CLEANUP;
}
memset(URLentry,0,entrysize);

/*reserved*/
*(URLentry)      = 0;
/*URL life time*/
ToUINT16(URLentry+1,65535);
/*URL length*/
ToUINT16(URLentry+3,36);
memcpy(URLentry+5,url,urllen);
/*Auths*/
*(URLentry+5+urllen) = 0;
```

```
//creat a body, and add URLentry at the beginning of the body
//the size of the body is dependent on the contents of the REG file
//need to write a REG file reading function
unsigned char* body;
bodysize = 0;
bodysize += 2;    //Srv len
int srvtypelen;
srvtypelen = strlen(srvtype);
bodysize += srvtypelen;
int scopelen;
scopelen = strlen(scope);
bodysize += scopelen;
int attrilen;
attrilen = strlen(attribute);
bodysize += attrilen;
bodysize += 1;    //Auths

bodysize += entrysize;


body = (unsigned char*)xmalloc(bodysize);
if (body<=0)
{
        result = -3;
        goto CLEANUP;
}
memset(body,0,bodysize);

memcpy(body,URLentry,entrysize);

/*Srv type len*/
ToUINT16(body+entrysize,srvtypelen);
/*srv type*/
memcpy(body+entrysize+2,srvtype,srvtypelen);
/*scope len*/
ToUINT16(body+entrysize+2+srvtypelen, scopelen);
/*scope*/
memcpy(body+entrysize+2+srvtypelen+2,scope,scopelen);
/*attribute len*/
ToUINT16(body+entrysize+2+srvtypelen+2+scopelen, attrilen);
/*attribute*/
memcpy(body+entrysize+2+srvtypelen+2+scopelen+2,attribute,attrilen);
/*Auths*/
*(body+entrysize+2+srvtypelen+2+scopelen+2+attrilen+1) = 0;

/*************************
// make header for SrvReq
*************************/
int langtaglen;
langtaglen = strlen(langtag);
int headsize;
headsize = langtaglen + 14;
```

```
        /*version*/
        *(pslpbuffer->start)       = 2;
        /*function id*/
  *(pslpbuffer->start + 1)   = 3;
        /*length*/
        ToUINT24(pslpbuffer->start + 2, bodysize+headsize);
        /*flags*/
        ToUINT16(pslpbuffer->start + 5, SLP_FLAG_FRESH); //dependent on something ????
        /*ext offset*/
        ToUINT24(pslpbuffer->start + 7, 0);
        /*xid*/
        ToUINT16(pslpbuffer->start + 10, (XID+1)%65535); //need to be careful here, may assign
XID outside ????
        /*lang tag len*/
  ToUINT16(pslpbuffer->start + 12, 2);
  /*lang tag*/
  memcpy(pslpbuffer->start + 14, langtag, langtaglen);

        pslpbuffer->curpos = pslpbuffer->start + langtaglen + 14 ;   //curpos is point to the start of the
body

        /*************************
        // combine header and body
        *************************/
        memcpy(pslpbuffer->curpos, body, bodysize);
        pslpbuffer->curpos += bodysize;

        /*************************
        // send it by TCP
        *************************/
        char* TCPbuf;
        int TCPbufsize;
        TCHAR Message[128];

        TCPbufsize = bodysize+headsize;
        TCPbuf = (char*)xmalloc(TCPbufsize);
        if (TCPbuf<=0)
        {
                result = -4;
                goto CLEANUP;
        }
        memset(TCPbuf,0,TCPbufsize);
        memcpy(TCPbuf, pslpbuffer->start, TCPbufsize);


#ifdef __DEBUG__                    //just comment off the while(1) and {}
        while(1)
        {                                               //when debuging, we loop the package
#endif
                nBytesSent = 0;
                nBytesIndex = 0;
                nBytesLeft = TCPbufsize;
```

```
				// Send the entire buffer
				while(nBytesLeft > 0)
				{
					nBytesSent = send(TCPsocketfd, TCPbuf+nBytesIndex, nBytesLeft, 0);
					if(nBytesSent == SOCKET_ERROR)
					{
						result = -5;
						goto CLEANUP;
					}

					// See how many bytes are left. If we still need to send, loop
					nBytesLeft -= nBytesSent;
					nBytesIndex += nBytesSent;
				}

				wsprintf (Message, TEXT ("Send Package, Len is %d\r\n"), nBytesSent);
				printsz(hWnd, Message);
				SendMessage(hWnd,WM_PAINT,0,0);
#ifdef __DEBUG__
				Sleep(10000L);
		}
#endif

		result = 0; // everything is ok
CLEANUP:
		/*Free memory here*/
		if (pslpbuffer)
		{
				xfree(pslpbuffer);
		}
		else if (body)
		{
				xfree(body);
		}
		else if (TCPbuf)
		{
				xfree(TCPbuf);
		}
		else if (URLentry)
		{
				xfree(URLentry);
		}

		if (result!=0)
		{
				wsprintf (Message, TEXT ("Error Code is %d\r\n"), result);
				printsz(hWnd, Message);
				SendMessage(hWnd,WM_PAINT,0,0);
		}

		return result;
}
```

Appendix B.

```
int receiveTCP(HWND hWnd, int sockfd)
{
        int result;
        /******************************
        * Read the body of the package
        ******************************/
        char* head;
        head = (char*)xmalloc(16);          //read the header of the package
        memset(head,0,16);
        int nBytesRecv = 0;
        int nReceiveAddrSize = 0;
        int FuncID = 0;

        if (head)
        {
                nBytesRecv = recv(sockfd, head, 16, 0);
        }
        if (nBytesRecv<=0)
        {
                result = GetLastError();
                goto CLEANUP;
        }
        FuncID = *(head + 1);

        /******************************
        * Read the body of the package
        ******************************/
        int bodylen;
        PSLPBuffer body;                    //need to use end, and curpos
                                                    //no useful here, maybe used later
        /* check the version */
  if(nBytesRecv >= 5 && *head == 2)
  {
    /* allocate the recvmsg big enough for the whole message */
    bodylen = AsUINT24(head + 2);
    /* one byte is minimum */
    if (bodylen <= 0)
      bodylen = 1;
    body = (PSLPBuffer)xmalloc(sizeof(struct _SLPBuffer) + bodylen + 1);
                memset(body,0,sizeof(struct _SLPBuffer) + bodylen + 1);
    if(body)
    {
                        body->start = (unsigned char*)body + 1;
                        body->curpos = body->start;
                        body->end = body->start + bodylen;

      while(body->curpos < body->end)
      {
        if(nBytesRecv > 0)
        {
          nBytesRecv = recv(sockfd,
                  (char*)body->curpos,
                  body->end - body->curpos,
                              0);
                if(nBytesRecv > 0)
                {
                   body->curpos = body->curpos + nBytesRecv;
                }
                else
                {
                   //errno = ENOTCONN;
                                                    result = -2;
                   goto CLEANUP;
                }
             }
             else if(nBytesRecv == 0)
             {
                //errno = ETIMEDOUT;
                                              result = -3;
                goto CLEANUP;
             }
             else
             {
                //errno =  ENOTCONN;
                                              result = -4;
                goto CLEANUP;
             }
          } /* end of main read while. */
       }
       else
       {
          //errno = ENOMEM;
                        result = -5;
          goto CLEANUP;
       }
    }
    else
    {
       //errno = EINVAL;
                  result = -6;
       goto CLEANUP;
    }
#ifdef __DEBUG__
         if (result>=0)
         {
                  TCHAR Message[128];
                  wsprintf (Message, TEXT ("Function ID is %d\r\n"), result);
                  printsz(hWnd, Message);
                  SendMessage(hWnd,WM_PAINT,0,0);
         }
#endif
         // everything is ok, return the Function ID
         result = FuncID;

CLEANUP:
         if (head)
```

```
        {
                xfree(head);
        }
        if (body)
        {
                xfree(body);
        }

        return result;

}

/*
int REGreader(void)
{
        FILE *regfd;
        char line[128];
        int result;
        int REGlen;
        int subregsrv = 0;      //when find one service section, check the index of this section,
                                                //if it was already read by this function
(subregsrv<=regerv), then goto next section
        int starttowrite = 0; //if get a new section, set this to 1, and this function will start to write the
buffer

        if (regfd = fopen("F:\\Thesis\\Shasha\\REG.txt","r") == NULL)
        {
                result = -1;
                return result;
        }

        while (!feof(regfd))
        {
                if (fgets(line,sizeof(line),regfd)==NULL)  //read one line from file
                {
                        result = -2;
                        return result;
                }
                if (line[0]=='#' || line[0]==';')       //commment
                {
                        continue;
                }
                if (mystrcmp(&line,"services"))
                {
                        subregsrv += 1;  // account this service section
                        if (subregsrv<=regsrv)
                        {
                                continue; //read next line
                        }
                        else
                        {
                                starttowrite = 1;
                        }
```

```
		}
		else if ()
		else
		{
			continue;
		}
	}

	result = REGlen;
	return result;
}

int mystrcmp(char* source, char* keyword)
// check if the keyword in the beginning of the source
{
	int keywordlen=strlen(keyword);
	int ifequ = 1;

	for (int i=0; i<keywordlen; i++)
	{
		if (source+i == keyword+i)
		{
			continue;
		}
		else
		{
			ifequ = 0;
			break;
		}
	}

	return ifequ;
}*/
```

COS/CCS 2008-23