Results of Comparing Bandwidth Usage and Latency: Service Location Protocol and Jini

Javier Govea and Michel Barbeau
School of Computer Science
Carleton University
1125 Colonel By Drive
Ottawa, ON K1S 5B6 Canada
{jgovea,barbeau}@scs.carleton.ca

May 30, 2001

Abstract

With resources such as printers, mail boxes or software available throughout the world and interconnected by fast networks, and with the increase in the number of mobile users, service discovery and pervasive computing are becoming important features of the network scenario. Service discovery is a essential element in the full deployment of pervasive computing. Service discovery allows devices to locate other devices or services avoiding the configuration of the services or the installation of drivers, a feature that is particularly important for roaming users. Roaming users are most likely to use wireless communications which operate under several constraints. This paper focuses on analyzing and evaluating two of these limitations, bandwidth usage and latency, of two approaches for dealing with service discovery: the Service Location Protocol (SLP) and Jini. In this paper we present the results of our bandwidth usage analysis and the results a comparison of latency of SLP and Jini.

Keywords - PERVASIVE COMPUTING, RESOURCE DISCOVERY, SERVICE LOCATION PROTOCOL, JINI, BANDWIDTH USAGE, LATENCY COMPARISON.

1 Introduction

The growth in the number of mobile users, due in part to devices such as PDAs, laptops, and cell phones which become more affordable daily, and the availability of services such as printers, memory, mail boxes or software in machines available world wide interconnected by fast networks, permits users to connect with all these resources remotely on demand. One of the main problems, in using all these resources, is the configuration of the services by users each time they move and for every single service they want to access. For example, users currently need to upload drivers for each service needed or to search for an IP address. By solving the configuration problem, it could be possible for roaming users, such as businessmen or passengers in a car [6], to access all these resources from their devices transparently. The same problem is faced in ad hoc networks where there is no central control, users move constantly, and there is a need of autoconfiguration.

Pervasive computing or ubiquitous computing, which roughly means computers anywhere at anytime, is a fairly recent concept in computer science. Several attempts have been made to define it. Birnbaum defined it as a computer hidden in so called information appliances such as washing machines and furnaces [8]. More recently Ciarletta and Dima define a layered model for pervasive computing [11]. The aim of all this work has been to provide a standard definition.

Pervasive computing implies the use of nomadic devices. The communication among these devices is by nature wireless. Nomadic devices and wireless networks work under several constraints, for example, limited power and bandwidth, ad hoc networking, and small displays [20].

Power consumption is seen in the literature as the most important problem for nomadic devices. A lot of research has been conducted in different areas to optimize power. For example, Chan and Tassiulas, in [9], present routing algorithms for optimizing power, Chen et al, in [10], focus on scheduling services in a low power MAC for wireless ATM networks. In a laptop, the hard disk and LCD screen consume most of the power while in a PDA most of the energy is consumed by the network interface. A network interface in idle mode can consume as much energy as a whole PDA [26]. Also, as bandwidth access increases, power consumption increases, thus reducing the battery life time.

Bandwidth is another important resource. Within a cell of current cellular systems, a set of users shares the same bandwidth and splits it among them. Because of its nature, the cellular architecture encourages the conservation of bandwidth per device in order to increase the number of users per cell.

Latency is also an important issue in wireless networks. The latency of a transmission depends on many factors such as the load of the network, error correction algorithms, physical layer, and routing algorithms [13]. It is well known that in a wireless transmission, the error rate is higher than in a wired network, and thus the latency is much higher. While in a wired network we can easily achieve speeds of up 100 Mbits/sec, in a wireless network it is difficult to go over 2 Mbits/sec.

This paper focuses on the bandwidth usage and latency of two important technologies for pervasive computing. These two issues are important for both pervasive computing and wireless networks in general, but to the best of our knowledge they have not been explored in previous publications. Currently there are several technologies available in the market that allow some degree of pervasiveness. They range from frameworks to protocols and hardware based.

Some examples of frameworks are the Salutation framework [12] and Home Audio/Video interoperability (HAVi) [24]. Some resource discovery protocols, which allow the devices to find each other, are the Service Location Protocol (SLP) [18], Dynamic Host Configuration Protocol (DHCP) [15], and Berkeley Secure Service Discovery Protocol [14]. Other technologies which provide more functionality in addition to a service discovery protocol are Universal Plug-and-Play (UPnP) [25], Bluetooth [16], and Jini [23].

Finally, there are other technologies, such as HP JetSend [27] that allows devices to talk to each other but does not provide a means of finding the resources or the MIT Intentional Naming Service [3] which is a good attempt for standardizing the attributes of services. These are some examples of different directions taken by the work in pervasive computing. Also, there are other technologies, hardware-based, that can work with the aforementioned as a physical layer such as Universal Serial Bus [19], FireWire (IEEE 1394) [1], and Home Phoneline Networking Alliance (HPNA) [4].

Bettstetter and Renner, in [7], give a good but broad comparison of several service discovery protocols. We will focus on specific elements, the bandwidth usage in bytes and latency, of SLP and Jini. This paper only presents our results of a bandwidth usage analysis and latency evaluation. A detail description of these analysis can be found in [17]. A brief description of these technologies is presented in Section 2, bandwidth usage comparison is presented in Section 3, and latency evaluation is presented in Section 4. Finally, some conclusions are given in Section 5.

2 Background

2.1 Service Location Protocol

The Service Location (SRVLOC) group, part of the Internet Engineering Task Force (IETF), published in 1997 the SLP Version 1 [28] and in 1999 the SLP Version 2 [18]. SLP is the IETF standard for service discovery.

SLPv2 is a protocol for service discovery. It has the notion of the scope, which is an unadministrative domain. For example, a scope can be a department within a company. SLP defines three entities:

- Service Agent (SA). Services are resources that can be used by a device, a user, a program or another service. Some examples are memory space, printers, and software. In general any resource available in the network is potentially a service.
- User Agent (UA). UAs are the clients or users of the services
- Directory Agent (DA). The primary function of DAs is to implement a repository of services where the clients, UAs, can look for particular services given particular attributes. SAs register their services with their attributes with the DA in their scope.

UAs, SAs, and DAs are members of scope. UAs and SAs can discover the DAs in their scope by different means, such as:

- Active discovery. SAs and UAs multicast SLP requests asking for the local DA.
- Passive discovery. DAs multicast advertisement messages on a periodic basis announcing their presence.
- UAs and SAs can learn the location of DAs by an external means, such as a configuration file.

UAs send a *Service Request* only to SAs and DAs supporting their scope. SLP addresses two modes of operation:

- Without DAs. UAs send *Service Request*s using IP multicast or broadcast. SAs, listening to a well-known port, find a match between a requested service and the service they offer, and reply to the UAs using unicast with their URL.
- With DAs. UAs and SAs discover the DA by any of the aforementioned means. Then SAs register their services with the local DA, whereas UAs can send queries to the DA inquiring for a service.

Figure 1 illustrates a common scenario in SLP. The operation mode is with DAs using active discovery. It is assumed that the SA, UA, and DA are members of the same scope. In this scenario: the SA and UA discover the DA by multicasting a *Service Request*. The *Service Request* message sets the parameter *Type* to *service:Directory-Agent*. Every DA that listens to this message responds with a unicast message, called *DA Advertisement*, that contains their URL.

Then the SA registers with the DA in its scope. The parameters of the *Service Registration* message are the *URL*, *Type* and *Attributes* of the service. The SA gets back a *Service Acknowledgment*, which is an error code. After the discovering process, the UA can look for a particular service by sending a *Service Request* to the DA specifying the *Type* and *Predicate* over attributes of the service needed. The DA responds with a *Service Reply* message, which includes a list of all the SAs' URLs that matches the request. If there is no match, the list is empty. Afterwards, the UA can establish communication with the SA. The communication between SAs and UAs is outside of the SLP architecture.

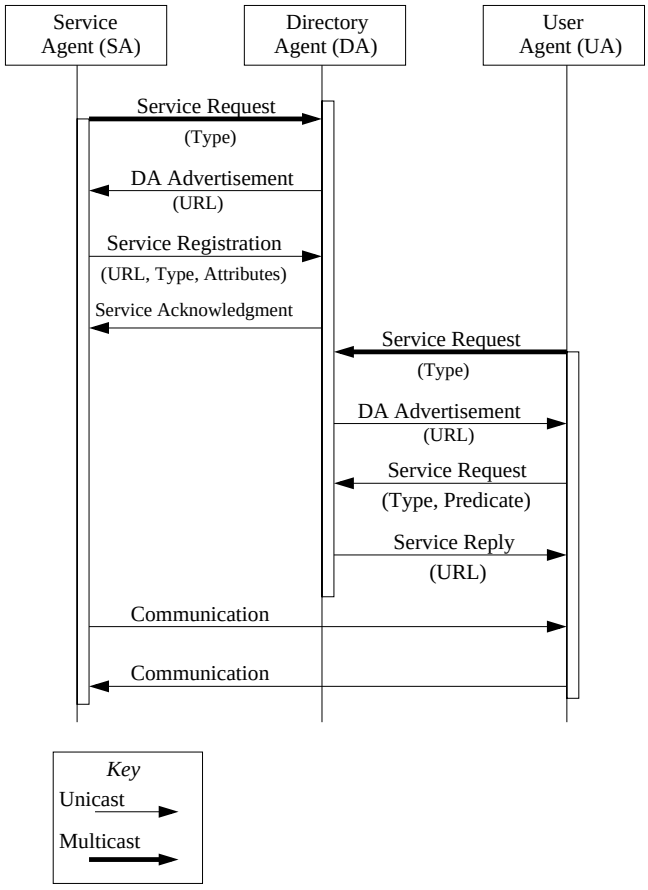

Figure 1: SLP Operation.

2.2 Jini Technology

Jini [23] is a coordination framework written in Java and developed by Sun Microsystems. In the development of Jini, many people from the industry and academia were involved. However, it is important to mention the work done by David Gelernter of Yale University and Nick Carriero who built the Linda coordination model using Tuple Spaces [2]. The evolution of Linda led to the JavaSpaces technology that eventually evolved into Jini.

Jini defines five key concepts: discovery, lookup, leasing, remote events, and transactions [22]. These concepts are also present in the SLP architecture, except the remote events and transactions. However, these two ideas can be implemented

on the top of SLP by other protocols or at a higher layer by some applications.

Jini defines three different entities: Service providers, Lookup services, and Clients. These entities are analogous to the SAs, DAs, and UAs respectively. It also defines the concept of group, which is analogous to the SLP scope.

Jini bases its operation on three protocols called discovery, join, and lookup protocol [23]. The last two protocols are not communication protocols. They only define a set of rules on how well behaved Jini entities should join a Jini community and how to look for a particular entity. The discovery protocol is used for discovering the Lookup services by the Service providers and Clients (discoverers). It consists of three specific protocols:

- Multicast request protocol. It is analogous to the active discovery in SLP. The lookup service listens to a well-known port. The discoverers multicast a request and get back a *Service Registrar* message.

- Announcement request protocol. It is similar to the passive discovery in SLP. The lookup service multicasts on a periodic basis a message advertising its presence. Interested discoverers contact the lookup service by sending a unicast request message and they get back a *Service Registrar* message.

- Unicast discovery protocol. This protocol is used by the discoverers when they know the address of the lookup service, and is similar to the last part of the announcement request protocol.

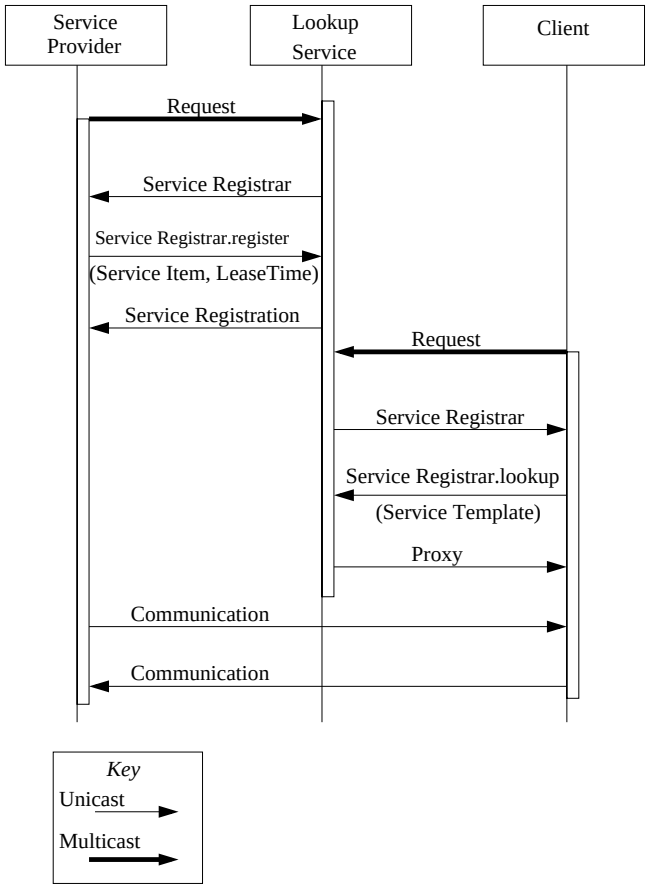

Figure 2: Jini Operation.

In these three protocols, the last message is always from the lookup service to the discoverers. This message contains the proxy of the lookup service, which is serialized according to the Java Object Serialization specification [21]. The proxy is an object that implements the *Service Registrar* interface. This interface defines the methods needed by the discoverers to talk to the lookup service.

One representative scenario demonstrating how Jini works is shown in Figure 2. In the Figure, the multicast request protocol is used and it is assumed that the three entities are part of the same group. Note that this figure is very similar to Figure 1 which illustrates the operation of SLP.

After discovering the lookup service, the service provider registers its service by calling the *register* method on the lookup service proxy. By doing this, the service provider uploads a proxy of the service. One of the parameters of the *register* method is a lease period, which states how long the service will be available. The lookup services respond with a *Service Registration* object, which is more than a simple acknowledgement. For example, it is possible to ask this object whether the lease period has been granted.

Clients find services by calling the *lookup* method on the lookup service proxy. The parameter of this method is a template specifying a set of attributes desirable in the service that is looked for. In response, the lookup service delivers the proxy of the service requested, if it is known; otherwise the lookup service responds with a *Null* value. This proxy is the

same that was uploaded by the service provider. After a client downloads a service proxy, it establishes communication, via RMI, with the service provider.

3 Bandwidth Usage Analysis

This section describes a bandwidth usage analysis performed on SLP and Jini. The analysis and the results are based on the SLP and Jini specification and in previous work [5].

3.1 SLP Analysis

We focus on the scenario illustrated in Figure 1, which shows a representative situation in SLP. It uses active discovery and the presence of one DA will be assumed.

Three processes are considered: discovery, registration, and lookup. The discovery process, executed by SAs and UAs, is represented, in Figure 1, by two messages: the *Service Request(Type)* and *DA Advertisement(URL)*. The registration, executed by the SAs, uses the messages *Service Registration(URL, Types, Attributes)* and *Service Acknowledgement*. The lookup process, executed by the UAs, uses the messages: *Service Request(Type, Predicate)* and *Service Reply(URL)*.

We compute an upper bound on bandwidth usage, in bytes, in SLP for each of the processes, and then we calculate the overall bandwidth using that information for a full-featured printer service implemented in C.

In the TCP/IP stack, SLP is placed over UDP. SLP messages share a common header and are encapsulated within a UDP header, IP header, and network header, i.e. an Ethernet header.

In the sequel, let N be the number of SAs, M be the number of UAs, T_R the service registration period, and T_L the service request period.

3.1.1 Discovery Process

Since one DA is assumed to be present in the scope and active discovery is used, the UAs and SAs broadcast a *Service Request* looking for a DA in their scope. For discovering DAs, the parameter *Type* of the request is set to *service:Directory-Agent*. According to the SLP specification, the *Service Request* message is always sent a constant number of times, C, even if the agents get a response from the DA after the first request. This is because the agents try to discover all the DAs available in the scope. To make a fair comparison with Jini, we will use the same constant in the Jini discovery process. With one DA in the scope, after the first *Service Request* message sent by the SAs and UAs, they get back a *Service Reply* message. The remaining $C - 1$ *Service Request* messages are discarded by the DA.

3.1.2 Registration Process

In the presence of a single DA, over a duration of T seconds, the amount of traffic, in bytes, generated by the registration process amounts to the size of an encapsulated *Service Registration* and the size of the encapsulated *Service Acknowledgement* message times the frequency of registration multiplied by the number of SAs.

3.1.3 Lookup Process

This process involves two types of messages: *Service Request* and *Service Reply*s. The amount of bandwidth during T seconds, leased for T_L seconds, due to *Service Request* messages amounts to the size of the encapsulated *Service Request* times the number of renews needed, multiplied by the number of SAs. Since the service requested is leased, there is a need to renew the lease before it expires.

Every *Service Request* message is sent using unicast to the DA and all URLs matching the required service are packed in a single *Service Reply* message in a field called URL. We assume that v% of the SAs available in the scope offer a matching service. The amount of bandwidth therefore corresponds to the size of an encapsulated *Service Reply* message, along with the number of URLs that matches the request times the service request frequency, due to the lease, multiplied by the number of UAs requesting a service.

3.1.4 Results

The overall amount of bandwidth, BU_{SLP}, can be calculated by adding the bandwidth usage of the discovery, registration and lookup processes.

Figure 3 plots the usage of bandwidth if T is fixed to one hour, T_L to 15 minutes, and T_R to five minutes. Percentage v is set to 10. The other values are set according to our implementation of the printer service. A printer service was implemented, in C, according to the RFC 2608 [18]. A UA for

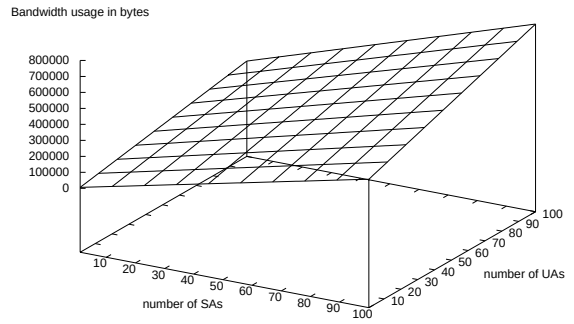

Figure 3: Bandwidth usage of a configuration with a DA, SAs, and UAs.

the printer and a DA were also implemented. All the agents were deployed in 100 MBits Ethernet network. These agents were used to obtain the values for the simulation presented in this paper. For this situation, that can be considered representative, usage of bandwidth is more sensitive to the number of UAs than the number of SAs.

Another example we analyzed is a simple *Hello World* service. This service was implemented and analyzed in the same way we did with the printer service, i.e. network conditions, language, RFC specifications. The results of analyzing the hello world service are similar to the printer service. The bandwidth consumption is only about 1% less. This is mainly because SLP is only delivering URLs and there is only a light difference, in bytes, between sending a URL of a full-featured printer service or a simple hello world service.

In [5], an SLP architecture with a DA is compared with one without DAs. It is shown that the architecture with a DA consumes less bandwidth, about 50% less. This is true for the most of the cases (a conditions showing for which cases is true is presented in the same paper).

3.2 Jini Analysis

We will follow the same idea of the previous subsection in the Jini bandwidth usage analysis. We refer to Figure 2 for our analysis of calculating an upper bound of a full-featured printer service using Jini. The lookup service used for the analysis is the one that comes with the Jini distribution, which is called *reggie*.

Jini makes use of RMI to achieve the communication among the entities. RMI uses in turn a TCP socket for communication. RMI follows a simple protocol for the communication. It uses two types of messages: request and reply. The object invoking a remote method sends a request over a TCP connection and the remote method returns a value or an exception over the same TCP connection.

RMI defines three different protocols for the communication: *StreamProtocol*, *MultiplexProtocol*, and *SingleOpProtocol*. The protocol is specified in the RMI header. The *StreamProtocol* only transmits the RMI requests and replies over a TCP connection. The *MultiplexProtocol* allows multiple sessions over the same TCP connection. The *SingleOpProtocol* uses HTTP protocol on the top of the TCP protocol. It uses Post and it is intended to be used when there are firewalls between the remote objects. The payload of the request contains the parameters of the method invoked remotely. Each parameter is serialized according to the Java Serialization Object specification.

The reply message has an acknowledgment in the header and the payload contains the object, primitive data type or exception returned by the method invoked, which is serialized according to the Java Serialization Object specification. We will assume the use of the *StreamProtocol*. We do not consider RMI running over another protocol such as IIOP, since doing this clearly will lead us to higher bandwidth consumption.

As in the SLP case, we consider the three processes involved in the Jini operation: discovery, registration, and lookup. Some of these processes use RMI, whereas others use only UDP or TCP connections.

3.2.1 Discovery Process

In Figure 2, the discovery process is represented by the *Multicast Request* message and the *Service Registrar* object returned. The *Multicast Request* message is a UDP datagram, whereas the *ServiceRegistrar* object goes in a TCP communication.

The bandwidth usage in Jini due to the discovery process by N service providers and M clients is the sum of these two messages multiply by N and M. When the service provider or the clients do not have the Jini classes already installed,

they can download them from the lookup service host. This process is done in the discovery stage.

As in SLP, the number of requests sent by the discoverers is always constant, even if they receive a response after the first request message sent. Jini specifies seven as the value for this constant, C. Jini assumes that after seven requests, the discoverers will get answers from all the lookup services available in the group.

3.2.2 Registration Process

In Figure 2, the registration process is represented by the invocation of the method *register* on the *Service Registrar* object with the item to be registered and a lease time as parameters. The response is the *Service Registration* object. This communication is done via RMI.

The service providers register with the lookup service for a lease period of time T_R and then the lease is renewed. The process of renewing the lease is also done via RMI. The renew process does not return any value. The service provider can call the method *getExpiration()* to know if the lease was granted. We assume that the lease is always granted.

Therefore, the total amount of bandwidth consumed in the registration process during a period of T seconds, leasing the service for T_R seconds, is defined as the size of the registration message plus the size of the renew lease message times the number of lease renews multiplied by the number of service providers.

3.2.3 Lookup Process

In Figure 2, the lookup process is represented by the invocation of the method *lookup* on the *Service Registrar* object. The response is the proxy of the service requested. With M clients discovering at a ratio of T/T_L independent services, the lookup process can be defined as the amount of traffic generated by the lookup queries times the ratio of the discovering multiply by the number of clients. The lookup queries are done via RMI. The queries include a parameter that specifies the attributes and type of the service needed. The object returned is the proxy of the service requested. If there are many services known by the lookup service that match the lookup request, it returns only one, which is chosen randomly. If there is no matching, it returns an object with a *Null* value. We assume that a service is always found.

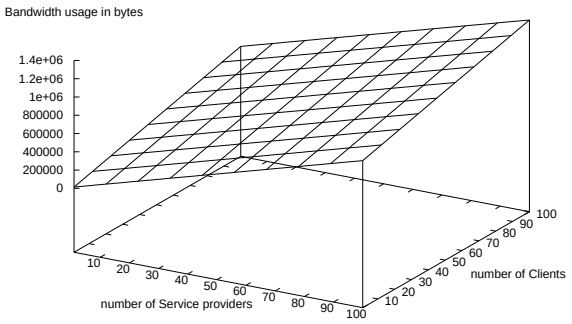

Figure 4: Bandwidth usage of JINI without downloading code.

3.2.4 Results

The total amount of bandwidth usage can be calculated by adding the bandwidth usage of each process. Figure 4 plots the usage of bandwidth of Jini based on analysis performed in the last subsections. Variables N, M, T, T_R, and T_L are set as in Subsection 3.1. The other parameters are set according to a full-featured printer service implemented using Jini 1.1. We deployed the printer service, client, and lookup service in the same network and conditions as the SLP agents. The Jini printer service has the same features of SLP printer service, i.e. attributes. In particular, the amount of code downloaded by the discoverers is set to 0, which means that the service providers and clients do not need to download code for running the lookup service proxy.

The Jini architecture is similar to the SLP architecture with a DA. However, Figure 4 shows a higher use of bandwidth, even though no code was downloaded, i.e. classes, only objects. Figure 5 plots the Jini bandwidth usage of the same printer service when we take into account the code downloaded. In our experiments, the size of the code was approximately 60 Kbytes.

Using Jini, it is possible to have communication between the service providers and the clients through RMI. However in order to make a fair comparison with SLP we did not include this process as part of our analysis since SLP does not address the communication between SAs and UAs.

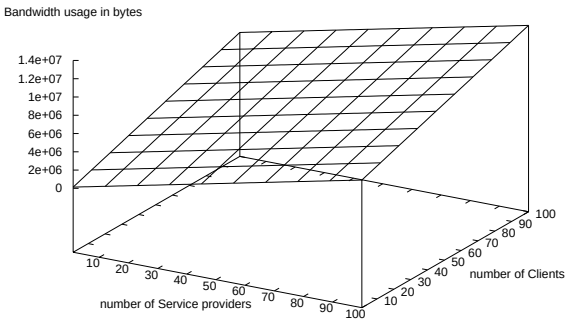

Figure 5: Bandwidth usage of JINI downloading code.

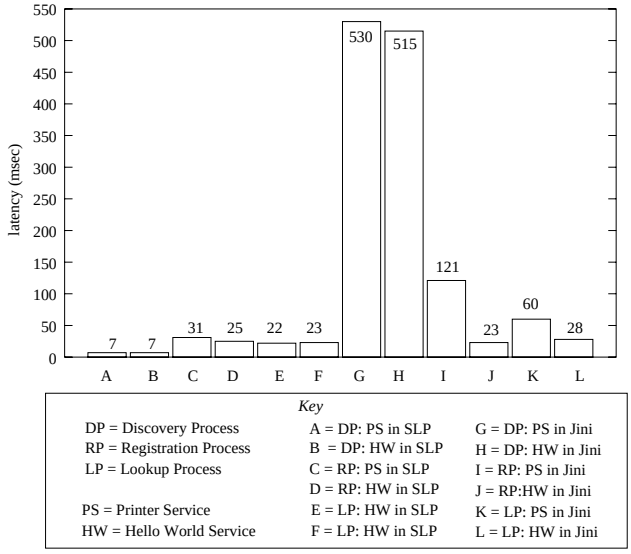

Figure 6: Latency comparison.

Another example we analyzed is a simple hello world service. The analysis shows that the bandwidth consumption of the full-featured printer service is higher by about 14% when there is no code to be downloaded and less than 1% when the code to be downloaded is 60 Kbytes. This number is not as high as could be expected mainly because the only difference between these two services is the size of their proxies. Whereas the hello world proxy is only 43 bytes, the printer proxy is 177 bytes long. Therefore, it can be expected that for any kind of service, the bandwidth usage in Jini would be similar to the one presented here with slight variations depending on the size of proxies transmitted.

4 Latency comparison

For testing the latency of the SLP and Jini service discovery protocols, a set of experiments was conducted. We tested two different applications separately in both architectures: a hello world service and a full-featured printer service.

For the SLP case, an SA, a printer service and a hello world provider, a UA of both services, and a DA were implemented in C. In the Jini case, the same service provider, client, and lookup service were implemented in Java. The lookup service utilized in our experiments was *reggie*. For each experiment, the SA first discovers the DA and then it registers with it. Then the UA discovers the DA and looks for the service. In the Jini case, the same process is executed by the service provider, client, and lookup service.

All the experiments, a set of 20, were conducted in 100 MBit Ethernet network of Pentium III, 500 MHz, and 128 MB RAM workstations. The results of the experiments are depicted in Figure 6.

From Figure 6, we can note clearly that Jini latency is very sensitive to the size of the proxies transmitted. Jini's latency is higher depending on the size of the proxies. If the proxy is small enough, such as the hello world proxy, then the difference with SLP is minimum. We can say that SLP latency is lower mainly because it transmits URLs instead of proxies and its relative light weightedness, i.e. fewer interactions, smaller messages.

5 Conclusions

Service discovery protocols are key elements in the full deployment of pervasive computing. Many devices in pervasive computing are nomadic, and therefore many of them use wireless communication where the bandwidth is an important resource that must be used efficiently, and latency is a key factor in the overall performance of the protocols. SLP and Jini are two important technologies for service discovery that

were discussed in this paper. We present results on bandwidth usage by SLP and Jini and their latency for a full-featured printer service.

The Jini configuration is similar to the SLP configuration with a DA, but Jini makes higher use of bandwidth mainly because of the transmission of proxies across the network. It is important to mention that Jini might need more bandwidth when the service providers or clients need to download code for running the proxies or the Jini classes. In this case, we showed that the bandwidth usage is more than 10 times higher than when there is no need of downloading code.

Although the two protocols behave differently in bandwidth usage and latency if a code migration protocol is paired with SLP, the SLP behavior would be very similar to the Jini's. Generally, Jini is a more complicated protocol and it is expected to have a worse timing performance compared with SLP, but it offers many more capabilities than SLP. The selection of SLP or Jini should not be based only on the analysis presented in this paper. SLP and Jini have other advantages and disadvantages that have to be taken in consideration. For example, SLP is simple to implement and it is very lightweight and thus suitable for wireless communications. However, it is only a string-based protocol and it does not address communication among the agents. Jini is flexible for implementing any service and it is OS independent because of the JVM. Jini has as its main strength the ability to move code, although this ability can be regarded also as a drawback since moving even a small piece of code can involve a lot of traffic in the network. Jini has heavy requirements, and the bandwidth usage is higher than SLP, although it is a more sophisticated system.

Finally, two other important commercial technologies available in the market are the Bluetooth and Universal Plug-and-Play. The first one uses the Simple Discovery protocol for service discovery, whereas the second one uses the Simple Service Discovery Protocol. Further work might include these two technologies in bandwidth usage and latency analysis.

References

[1] IEEE Std. 1394a-2000, Standard for a High Performance Serial Bus, Amendment 1.

[2] Linda Group. http://www.cs.yale.edu/Linda/ linda.html.

[3] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. In *Proceedings of the 17th ACM SOSP*, December 1999.

[4] The Home Network Phone Alliance. Simple, high-speed ethernet technology for the home. White Paper, June 1998.

[5] M. Barbeau. Bandwidth usage analysis of service location protocol. In *Proceedings of the 2000 International Conference on Parallel Processing Workshops*, pages 51–56, Toronto, August 2000.

[6] C. Bettstetter. Toward internet-based car communications: On some system architecture and protocol aspects. In *Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School: Innovative Internet Applications*, Twente, Netherlands, September 2000.

[7] C. Bettstetter and C. Renner. A comparison of service discovery protocols and implementation of the service location protocol. In *Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School: Innovative Internet Applications*, Twente, Netherlands, September 2000.

[8] J. Birnbaum. Pervasive infomation systems. *Communications of the ACM*, 40(2):40–41, February 1997.

[9] J. Chan and L. Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *Proceedings of IEEE INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 22–31, 2000.

[10] J. Chen, K. M. Sivalingam, P. Agrawa, and R. Acharya. Scheduling multimedia services in a low-power MAC for wireless and mobile ATM networks. *IEEE Transactions on Multimedia*, 1(2):187–201, June 1999.

[11] L. Ciarletta and A. Dima. A conceptual model for pervasive computing. In *Proceedings of the 2000 International Conference on Parallel Processing Workshops*, Toronto, August 2000.

[12] Salutation Consortium. Salutation Architecture Version 2.0c. ftp://ftp.salutation.org/salute/Sa20e1a21.pdf, June 1999.

[13] WAP Forum W3C Cooperation. W3C Note Version 1.1. White Paper, September 1998.

[14] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *Proceedings of the 5th ACM/IEEE International Conference on Mobile Computing MobiCom'99*, pages 24–35, 1999.

[15] R. Droms. Dynamic Host Configuration Protocol. IETF RFC 2131, March 1997.

[16] D. Farnsworth and B. Miller. Bluetooth Specification Version 1.0b, Service Discovery Protocol (SDP). http://www.bluetooth.com/link/spec/bluetooth_e.pdf, June 1999.

[17] J. Govea. Bandwidth usage comparison: Service Location Protocol and Jini. Master's thesis, Carleton University, February 2001.

[18] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. IETF RFC 2608. June 1998.

[19] USB Implementers Forum Inc. USB 2.0 Backgrounder. http://www.usb.org/developers/usb20/ backgrounder.html.

[20] H. Little. Why wireless is different. The 4th CACR Information Security Workshop, April 2000.

[21] Sun Microsystems. Java Object Serialization Specification, Revision 1.4. http://www.sun.com/research/forest/opj/docs/ guide/serialization/, November 1998.

[22] Sun Microsystems. Jini Architectural Overview. Technical White Paper, January 1999.

[23] Sun Microsystems. Jini Specification 1.0.1. http://www.sun.com/jini/specs/index.html, November 1999.

[24] The HAVi Organization. The HAVi Specification v1.0. http://www.havi.org/techinfo/index.html, January 2000.

[25] Universal Plug and Play Forum. Universal Plug and Play Device Architecture, Version 0.91, March 2000.

[26] M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications*, E80-B(8):1125–31, 1997.

[27] HP JetSend Communication Technology. Protocol Specification, Document Version 1.5. http://www.jetsend.hp.com/servlet/js/developer/ registered/disclaimer.shtml, May 1999.

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