Bandwidth Usage Analysis of Service Location Protocol

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

Abstract

A resource discovery protocol allows networked embedded processors and handheld devices to discover each other's capabilities. Service Location Protocol (SLP) has been proposed by the IETF for that purpose. The usage of bandwidth, or the amount of traffic, that such a protocol makes is a characteristic that is particularly relevant when devices are mobile and wireless. Protocols that are less chatty are more desirable because wireless bandwidth is a scarce resource as well as mobile device power. The goal of this work is to characterize the usage of bandwidth SLP makes and to contrast it with the usage of bandwidth that JINI makes, which is another resource discovery protocol. We present equations characterizing the usage of bandwidth made by SLP and JINI.

1. Introduction

Connecting embedded processors and handheld devices to the net raises an important question. How can small processors become aware of each other and communicate with each other? Ideally, it would be nice if it could be done wirelessly in a plug-and-play fashion. Hence, wireless handheld devices would dynamically find, with little configuration, the processors embedded in home appliances, cars, etc. The answer to this question relies in the development and deployment of several hardware and software technologies among which a resource discovery mechanism plays a key role.

A number of resource discovery protocols has been proposed such as the JINI of Sun Microsystems [11], Service Discovery Protocol of Bluetooth [2], Service Location Protocol (SLP) of IETF [7] [8], and Universal Plug and Play of Microsoft [3].

The work presented in this paper aims at characterizing the usage of bandwidth made by resource discovery protocols. This exercise is particularly relevant in the context of wireless networks. Indeed, less chatty protocols are more desirable because i) wireless bandwidth is a scarce resource and ii) power of handheld devices need to be conserved by minimizing the number of transmissions.

We present work aiming at characterizing the usage of bandwidth made by SLP and contrasting it with the one made by JINI.

JINI/SLP proposes two alternative architectures. The first involves only Service Providers/Service Agents (SAs) and Clients/User Agents (UAs) communicating directly with each other. With the aim of reducing traffic, a second architecture involving Service Providers/SAs, Clients/UAs, and Lookup Services/Directory Agents (DAs) acting as central sources of information and to which Service Providers/SAs register services and Clients/UAs enquire about services. In this paper, we discuss the usage of bandwidth for each of these alternatives.

In Section 2, we review resource discovery, SLP, and JINI. In Section 3, we discus usage of bandwidth. We conclude with Section 4.

2 Resource Discovery

For the connection of a client process to a server process from machine to machine on the Internet, the client requires the IP address of the machine of the server and port number of the socket on which the server is listening. There is, therefore, the problem of mapping service names to machine names and port numbers on which services are offered.

The current practice in IP networks for mapping service names to machine names on which services are offered relies on naming conventions for machines offering services (e.g. mailhost.scs.carleton.ca, ftp.scs.carleton.ca, www.scs.carleton.ca) and registered associations with a DNS of standard machine name and real machine address or name (e.g. www.scs.carleton.ca is mapped to

0-7695-0771-9/00 $10.00 © 2000 IEEE

fusion.scs.carleton.ca). A drawback of this approach is that only one machine per DNS can offer a service under a standardized name. A solution to this problem has been proposed [5]. It is called DNS SRV Resource Records. It allows mapping a service name (as defined in file /etc/services, e.g. FTP) to names of machine offering the service.

Machine names have then to be mapped to machine addresses. It is based both on a local name server (DNS) and a local file listing associations of machine name and IP address (file /etc/hosts on Unix).

For mapping service names to port numbers on which services are offered, ports numbers are standardized. Associations of service name to port number and transport protocol name are stored in a local file (e.g. /etc/services on Unix).

On the one hand, the current practice in IP networks has certain advantages. There are no requirements for special infrastructure for service location. Indeed, service location is supported by in place naming services. On the other hand, it suffers from several drawbacks. There is the impossibility of advertising multiple machines offering the same service under the same name (unless DNS SRV Resource Records are used). UAs must be aware of naming conventions. Search by attributes is not supported. Machine names and port numbers are discovered using different mechanisms. Updates need to be performed at two different places. Introduction of new types of services requires standardization of new names. Information may not be up-to-date. Finally, there is the absence of a generic solution for all kinds of hardware and software.

2.1 Service Location Protocol

SLP is a general protocol for advertisement and location of network services (e.g. hardware services such as a printer, a file system, a scanner, a projector, and software services such as a print spooler system, a Web server, a CORBA server, a data base) at the scale of an enterprise network. Advertisement and location of services is by name and by characteristics.

SLP defines a structure of service location information [6]. The goal is precise description of elements of service advertisements (i.e. a type of service, attributes, a service access point) and an extensibility allowing introduction of new types of service.

A service type is described by a structure called a *service type template*. Service type templates have a formal and standardized syntax that is both human and machine readable. A service type template defines the attributes associated with the service.

A *service access point* is information required to contact a service (e.g. an IP address and a port number). In SLP,

Figure 1. Interaction between a DA, an SA, and a UA.

a service access point is represented by a special type of URLs called URLs of scheme *service:*.

An announcement of a service consists of an URL of scheme service: or a generic URL [1].

Interactions between DAs, SAs, and UAs are based on the following basic messages: Acknowledgment (SrvAck), Service Reply (SrvRply), Service Request (SrvRqst), and Service Registration (SrvReg) (see Fig. 1). It contains a status code of the registration.

Message SrvReg is sent by an SA to a DA, using TCP or UDP unicast. The purpose of this message is registration of new services. It contains a URL, a type of service name, and descriptive attributes. Message SrvAck is sent by a DA to an SA, using TCP or UDP unicast. Message SrvRqst is sent by a UA to an SA, using UDP multicast or broadcast, or to a DA using UDP unicast or TCP unicast (when the reply can't stand in one UDP datagram). The purpose of this message is to lookup for services. It contains a type a service and a predicate which is query evaluated over the attributes of registrations in a DA. Message SrvRply is sent by an SA or a DA to a UA using UDP unicast or TCP unicast (when the reply can't stand in one UDP datagram). The purpose of this message is to respond to SrvRqst. It contains URLs of SAs matching the query.

SLP has the notion of scope which is an administrative domain. A scope is a group of UAs and SAs. DAs support scopes. UAs send SrvRqst only to SAs and DAs supporting their scope. SAs send SrvRegs only to all the DAs supporting their scope. The purpose of the concept of scope is to provide scalability limiting the network coverage of a request and the number of replies.

UAs and SAs can be member of several scopes. They can learn their scope name(s) by, for example, reading a configuration file.

There are two models of interaction: a model involving DAs (as in Fig. 1) and a model not involving DAs. Without DAs, UAs send using IP multicast or broadcast message SrvRqst to SAs. SAs are listening and when they find a match between a requested service and a service they offer they reply to the UAs using unicast.

With DAs, UAs, SAs, and DAs are members of scope. UAs and SAs communicate with DAs supporting their scope using unicast UDP or TCP. SAs register their offer

Figure 2. Interaction between a Client, a Lookup Service, and a Service Provider.

of service with DAs in their scope. UAs sends queries to the DAs supporting their scope. If a DA finds a match between a requested service and a registered service it replies to the UA.

There are four different ways SAs and UAs can obtain the service access point of their DA, through a configuration file, a DHCP server, active discovery (multicast of requests by SAs and UAs), and passive discovery (multicast of advertisements by DAs).

During the process of discovery of SAs by UAs, when sending SrvRqst using multicast or broadcast should stop? How to provide with reasonable probability a complete set of available services while not waiting for too long for SAs or DAs to respond?

To address this issue, SLP defines a multicast convergence algorithm. A SrvRqst is transmitted by a UA up to four times over a period of 15 seconds. Message SrvRqst contains a field called *previous responder list.* The list contains the IP addresses of the SAs that have returned SrvRplys so far in the execution of the multicast convergence algorithm. An SA listening and receiving a SrvRqst with its own IP address within the previous responder list of the message ignores the request and remains silent.

2.2 JINI

The architecture of a JINI system consists of Clients, Lookup Services, and Service Providers which are analogous to the concepts of UAs, DAs, and SAs of SLP.

In JINI, a discovery protocol is used by a Service Provider or a Client to discover a Lookup Service. Thereafter, a Service Provider may register with the Lookup Service with a protocol called *Join.* A service may be located on the Lookup Service by a Client using a protocol called *Lookup.*

The discovery protocol goes as follows. A Service Provider or a Client sends a discovery request on the local network using multicast. Listening Lookup Services reply to the Service Provider or Client using unicast. Each Lookup Service returns a proxy object which methods are used by the Service Provider or Client to contact the Lookup Service.

Following the discovery, protocol Join is used. The Service Provider calls method *register()* to load a service object (also called a proxy) into the Lookup Service (see Fig. 2). The service object consists of a Java interface to the service (i.e. signature of methods and descriptive attributes). The Lookup Service returns a service registration object that will be used by the Service Provider to maintain its offer of service.

Protocol Lookup is used by a Client to enquire for Service Providers with a Lookup Service. Location is by type of Java interface or by values of attributes of the service. The requirements of the Client are specified with a service template.

When a Service Provider is located, the corresponding service object is taken from the Lookup Service and loaded into the Client. Afterwards, the Client communicates through the service object (which acts as a proxy) with the Service Provider. The communication protocol used between the service object and service provider is not in the scope of JINI and is said private. RMI can be used for that purpose.

In contrast to SLP, JINI needs a Java virtual machine at the Client as well as the Service Provider nodes.

As SLP, presence of a Lookup Service is not mandatory. For location of a service when there are no Lookup Services, a Client can apply a technique called *peer lookup.*

The Client sends a message called identification to request registration messages from Service Providers (the identification message is normally sent by a Lookup Service). The Client then receives registration messages from Service Providers among which one can be selected.

In addition to components Client, Lookup Service, and Service Provider, JINI requires some elements of infrastructure. When a proxy or a service object is down loaded. Only the data member are down loaded. The implementation class must be down loaded. A Web server is required for down loading of code.

JINI is based on Remote Method Invocation (RMI) [10]. A RMI activation daemon is required to start Java server object on demand such as the JINI Lookup Service.

3 Bandwidth usage analysis

3.1 The SLP Case

We compute an upper bound on bandwidth usage, in bytes, by SLP.

In the TCP/IP stack, SLP is placed over UDP[1]. SLP messages share a common header and are encapsulated within a UDP header, a IP header, and a network header (e.g. Ethernet header).

[1] We consider SLP over UDP only.

In the sequel, variables $ESRVACK$, $ESRVREG$, $ESRVRQST$, $ESRVRPLY$, URL, and PR respectively represent the size of a encapsulated SrvAck, a encapsulated SrvReg, a encapsulated SrvRqst, a encapsulated SrvRply, a URL, and a previous responder address.

Let N be the number of SAs, M be the number of UAs, P the service registration period, and L the service request period. Over a duration of T seconds, in the presence of a single DA, the amount of traffic, in bytes, generated by registrations is given by the following expression:

$$Reg = N \times T/P \times (ESRVREG + ESRVACK)$$

The amount of traffic generated by registration is the number of SAs times the frequency of registration times the size of an encapsulated SrvReg message times the size of an encapsulated SrvAck message.

In a configuration without DAs, the multicast convergence algorithm is applied. Let us suppose that $x\%$ of the SAs reply after a first request, $y\%$ after a second request, and $z\%$ after a third ($x \leq y \leq z \leq 100$). The amount of bandwidth generated by SrvRqst messages is modeled by the following formula:

$$Req_1 = M \times T/L \times$$
$$[4 \times ESRVRQST + PR \times N \times (x + y + z)/100] \quad (1)$$

The amount of bandwidth generated by SrvRqst messages is the number of UAs times the service request frequency times the size of four encapsulated SrvRqst messages. If the first SrvReqst message, the previous responder list is empty, in the second it contains the addresses of $w\%$ of the SAs, in the third $x\%$ of the addresses, and in the fourth $y\%$ of the addresses.

In a configuration comprising a DA, the multicast convergence algorithm is not applied. The amount of bandwidth due to SrvRqst messages amounts to:

$$Req_2 = M \times T/L \times ESRVRQST$$

In a configuration without DAs, every reply contains a single URL. We assume that $v\%$ of the SAs return a reply ($z \leq v \leq 100$). The amount of bandwidth incurred to SrvRply messages is given by the following formula:

$$Rep_1 = M \times T/L \times v/100 \times N \times (ESRVRPLY + URL)$$

The amount of bandwidth incurred to SrvRply messages is the number of UAs that return a reply times the service request frequency times the number of SAs times the size of an encapsulated SrvRply along with a URL (we assume an average size for URLs).

In a configuration with a DA, every SrvRqst message is sent using unicast to the DA and all URLs matching the required service are packed in a single SrvRply message.

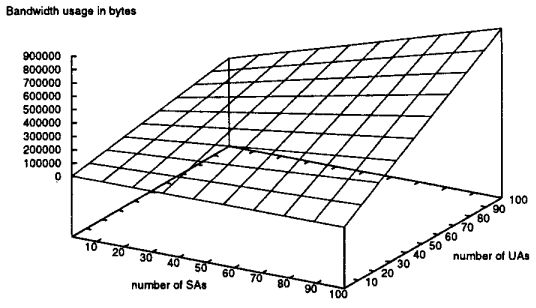

Figure 3. Bandwidth usage of a configuration with SAs and UAs only.

We assume that $v\%$ of the SAs offer a matching service. The amount of bandwidth therefore corresponds to:

$$Rep_2 = M \times T/L \times (ESRVRPLY + v/100 \times N \times URL)$$

The amount of bandwidth due to SrvRply messages is the number of matches times the service request frequency times the size of an encapsulated SrvRply message along with N URLs.

In a configuration without DAs, with N SAs and M UAs, the overall amount of bandwidth generated over a period of T seconds is expressed by the following equation:

$$BU_1 = Req_1 + Rep_1 \quad (2)$$

It is the traffic due to SrvReg messages, sent with the application of the multicast convergence algorithm, and the traffic due to SrvRply messages returned by the SAs. Figure 3 plots the usage of bandwidth if T is fixed to one hour and L to 15 minutes (other fields in SLP messages such as *service-type*, *scope-list*, and *predicate* are assigned representative values). Percentages x, y, and z are respectively set to 5, 8, and 10.

In this representative situation, usage of bandwidth is more sensitive to the number of UAs than to the number of SAs. In a configuration with a DA, with N SAs, and M UAs, the overall amount of bandwidth generated over a period of T seconds is expressed as the following formula:

$$BU_2 = Reg + Req_2 + Rep_2 \quad (3)$$

It is the the traffic due to SrvRegs, SrvRqsts sent using unicast, and SrvRplys that pack matching URLs. Figure 4 plots the usage of bandwidth if T is fixed to one hour, L to 15 minutes, and P to five minutes. Percentages v is set to 10.

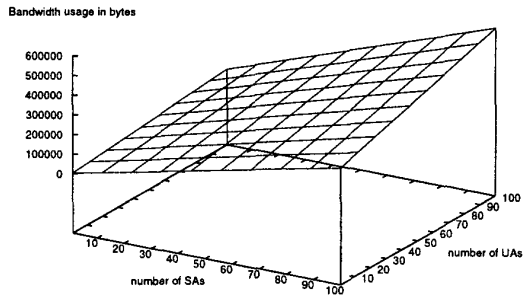

Figure 4. Bandwidth usage of a configuration with a DA, SAs, and UAs.

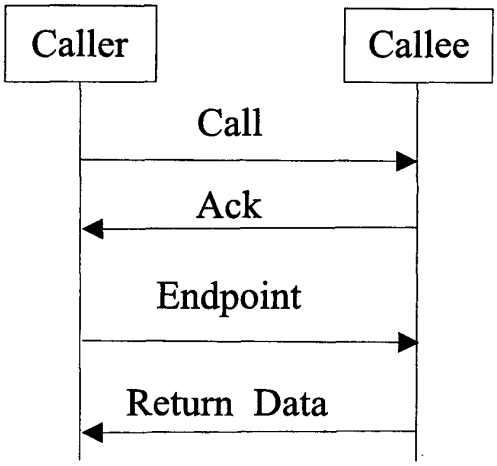

Figure 5. An RMI interaction.

For this particular situation, that may be considered representative, usage of bandwidth with a DA is twice the one without DAs (Figure 3) and more sensitive to the number of UAs than the number of SAs.

It is interesting to isolate the condition under which bandwidth usage with a DA is less than bandwidth usage without a DA, that is, the condition such that $BU_2 < BU_1$. Simple algebraic manipulations and simplifications of Equations 2 and 3 lead to the following conclusion, $BU_2 < BU_1$ if and only if

$$N/P \times (ESRVREG + ESRVACK) < M/L \\ (3 \times ESRVRQST + \\ (N-1)ESRVRPLY + N(PR(x+y+z))/100) \quad (4)$$

With the addition of a DA, bandwidth usage is less as long as the amount of traffic generated by SrvReg messages is less than the amount of traffic generated by the three retransmissions of the SrvRply message by the multicast convergence algorithm. Nevertheless, it tells that if the number of optional retransmissions by the multicast convergence algorithm can be limited, the bandwidth usage of a configuration without DAs approaches the one with a DA while not having the overhead of a DA.

3.2 The JINI Case

JINI uses RMI which in turn uses TCP. We discuss traffic generated by TCP, RMI, and JINI protocols Join and Lookup.

The amount of traffic generated by TCP is modeled as the equation:

$$T_{TCP}(X) = \alpha_{TCP} + \beta_{TCP}(X)X \quad (5)$$

In the above, α_{TCP} represents the overhead of TCP due to connection establishment and release, five packets of 40 bytes each (TCP header 20 bytes, IP header 20 bytes) for a total of 200 bytes.[2] $\beta_{TCP}(X)$ represents the overhead due to segmentation and acknowledgments of each individual data unit. It is defined as:

$$\beta_{TCP}(X) = \frac{2H}{X}\left\lceil \frac{X}{MSS} \right\rceil + 1 \quad (6)$$

where H and MSS respectively correspond to the size of headers of TCP and IP encapsulation and maximum segment size (1460 bytes on most systems).

The RMI protocol is defined in Ref. [10]. An RMI call essentially consists of four interactions: a call message, an acknowledgment, a endpoint identifier, and a return value which sizes are modeled as variables C, A, E, and R (Figure 5). The traffic generated by a RMI call is modeled as the following equation.

$$RMI(C,A,E,R) = \alpha_{TCP} + \beta_{TCP}(C)C \\ +\beta_{TCP}(A)A + \beta_{TCP}(E)E + \beta_{TCP}(R)R \quad (7)$$

The Join and Lookup protocols are defined in Ref. [12].

To register a service, a service provider down loads a proxy which provides an interface to a lookup service and hides the actual protocol used to communicate with the remote lookup service. Registration actually translates to a call to remote operation *register()* on a lookup service (see interface *Registrar in package* com.sun.jini.reggie [9]). The method has two parameters: a service item and a lease duration. The service item consists of a serialized service object. The lease duration is the lifetime of the registration. Operation *register()* returns a unique service ID.

[2]For details about TCP/IP traffic see Ref. [13].

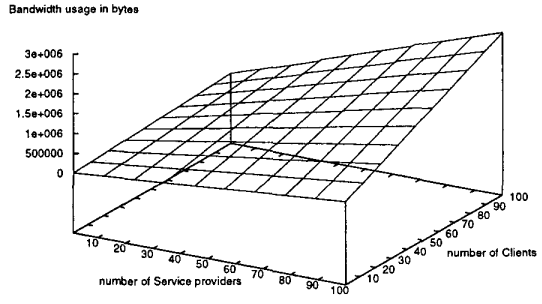

Figure 6. Bandwidth usage of JINI.

To find a service, a client down loads as well proxy providing an interface to a lookup service. A lookup actually translates to a call to remote operation *lookup* on a lookup service (see interface *Registrar in package* com.sun.jini.reggie [9]). The method has one parameter: a service template. It is a specification of the service the client is looking for. If there are matches, operation *lookup* returns serialized service objects providing the service.

Figure 6 plots the usage of bandwidth of JINI. Parameters of variables N, M, T, P, L, and v are set as in Subsection 3.1. To set the other parameters, we consider a simple service object that returns the string `Hello, World!` [4]. In particular, the size of a serialized service item including the service object is 219 bytes. A lease duration is a long integer (i.e. two bytes). A unique service ID is a 128-bit value. The size of a serialized service template is 279 bytes. A serialized representation of a service object that returns the string `Hello, World!` takes 43 bytes.

Method *lookup()* returns the member data of the service object (by definition of Java serialization). It doesn't return the class implementations. Which is actually required to use the service object. If the client doesn't have that class locally, it must down load it from a location (i.e. a HTTP server). tagged on the serialization of the service object (this tag is called the *Code Base*).

To make a fair comparison of JINI with SLP, we make abstraction of traffic required to down load, execute, and call the service object. To reduce traffic, Clients may stored locally these classes. Similarly, a SLP UA may contain all the protocols required to communicate with the SAs.

There is also some traffic due to the interactions with RMI activation daemons. These interactions are required to start on demand JINI Lookup Services and Service Providers. A RMI daemon normally resides on each node on which a Lookup Service an Service Provider is running. Hence, the RMI activation daemon overhead doesn't necessarily produce network traffic, just local traffic.

4 Conclusion

Resource discovery, which is required for auto configuration, and wireless networks are important tools of pervasive computing. On wireless networks, bandwidth is a scare resource and must be used efficiently. In this paper, we have developed two equations characterizing the usage of bandwidth of SLP for two main configurations. Furthermore, we have identified a condition which may be helpful to select a configuration instead of an other.

JINI makes a substantially higher usage of bandwidth than SLP. The analysis we presented in Subsection 3.2 makes abstraction of several details. The example we considered, `Hello, world`, is simple. The actual traffic may be higher. The infrastructure required by JINI is more complex than the one required by SLP. JINI is, however, a more sophisticated system.

References

[1] T. Berners-Lee, R. Fielding, U. Irvine, and L. Masinter. Uniform Resource Identifiers (URI): Generic syntax. IETF Request for Comments: 2396, August 1998.

[2] Bluetooth. Specification of the bluetooth system. http://www.bluetooth.com, December 1999.

[3] M. Corporation. Universal plug and play: Background. http://www.upnp.org/resources/UPnPbkgnd.htm, 1999.

[4] H. Edwards. *Core JINI*. Prentice Hall PTR, Upper Saddle River, NJ, 1999.

[5] A. Gulbrandsen and P. Vixie. A DNS RR for specifying the location of services (dns srv). IETF Request for Comments: 2052, October 1996.

[6] E. Guttman, C. Perkins, and J. Kempf. Service templates and service: schemes. IETF Request for Comments: 2609, June 1999.

[7] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. IETF Request for Comments: 2608, June 1999.

[8] J. Kempf and P. St. Pierre. *Service Location Protocol for Enterprise Networks*. Wiley, 1999.

[9] S. Microsystems. Jini 1.0.1. http://www.javasoft.com/jini.

[10] S. Microsystems. Java remote method invocation specification, December 1999.

[11] S. Microsystems. Jini architecture specification, November 1999.

[12] S. Microsystems. Jini lookup service specification, November 1999.

[13] W. Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley Publishing Company, Inc., professional computing series edition, Reading, Massachusetts, 1994.