Improving Quality of Experience by Adding Device Resource Reservation to Service Discovery Protocols

A. Delphinanto, A.M.J. Koonen
Technische Universiteit Eindhoven
Eindhoven, The Netherlands
A.Delphinanto@tue.nl

F.T.H. den Hartog
TNO
Delft, The Netherlands
Frank.denHartog@tno.nl

Abstract— Current service discovery protocols (SDP) hardly provide information on the actual availability of resources in the network or a mechanism for (device) resource reservation. When the resources cannot serve all multiple client requests at the same time, conflicts happen, often involving heavy and frequent reconfiguration traffic. This paper presents a generic resource reservation scheme. Its properties are derived from common SDP operations. We built a prototype, measured the extra overhead such a reservation manager introduces and simulated the gain in network scalability. The main conclusion is that our solution improves the scalability and the sustainability of the service access significantly, and at a minor cost.

Resource Reservation, UPnP, Service Discovery

I. INTRODUCTION

Rapid developments in networking and digital processing technologies and the decreasing of manufacturing cost allow consumers to use their electronic equipment for ever more purposes in an intuitive and convenient way. Consequently, multiple digital devices appear in the various private domains of the user (home, car, office, workplace, the personal operating space, etc.) forming small to medium size private local networks. These private networks can often be characterized as dynamic and technologically heterogeneous. The increasing size and number of these networks, as well as the growing need to interconnect them, leads to significant management complexity for the user. This is where auto-configuration protocols and device- and service-discovery protocols (SDP) come into play. These protocols (re)configure a private network dynamically, discover the device- and service properties of the devices present, and communicate these properties to the other devices in the network. Current protocols are still relatively immature and suffer a number of limitations. The one relevant to this paper is the lack of an acceptable mechanism for device resource reservation.

Device resource reservation is required whenever a device acts as a server in the network and shares its services to multiple clients but, because of its limited resources, can only serve a limited number of clients simultaneously. For example, most wireless music receivers that are on the market today can only provide audio streams to one client at a time. They do not report their current state as being occupied and, often, they still accept service access requests by other clients. This leads to overriding of the first client, to a strong increase of unnecessary

configuration traffic because, for instance, the first client may try to regain the service access, or to other erroneous behavior. This problem has been mentioned in the literature before [1, 2]. What is needed is service access management to solve the conflict, or more generic, resource reservation management.

This paper presents and discusses a concept of a resource reservation manager. It enables a client to reserve relevant service resources for sustainable service access and protect this reservation from other clients. Three important design requirements were applied. The reservation manager should be:

- 1) based on a centralized architecture. Hence, it can gracefully be implemented on the existing SDPs and will avoid extra requirements (and therefore costs) on the current server specifications.
- 2) independent of specific SDP properties that are not shared by other SDPs. This makes bridging easier to private networks that are governed by other SDPs.
- 3) manageable, possibly remotely, so it can be updated with, for instance, more advanced policies.

We built an example of such a reservation manager using Universal Plug and Play (UPnP) [3], simulated the gain in network scalability, and measured the overhead it introduces.

The paper is structured as follows. In section II, we discuss the relevant properties of SDPs and the state of the art on resource reservation management. In the next section we show how much unnecessary traffic may be generated by a conflict. We present our reservation manager design in section IV, followed by a description of our reservation manager implementation. The performance of the implementation is evaluated in section VI, followed by our conclusions.

II. SDPs AND RESOURCE RESERVATION

Service discovery technologies have been developed to reduce the complexity involving the manual configuration of multiple devices to form a local network, even if the network is still relatively small. These technologies enable users to automatically discover network services, configure their properties, and access their functionalities. Results of various surveys on these technologies [4-6], conclude that these protocols contain mostly the same elements and functionalities but have different approaches, and that none of these technologies is superior. The elements and functionalities of the SDPs that are similar and useful to this paper are:

The work presented in this article has been carried out in the collaborative Freeband Communication technology program which is supported by the Dutch Ministry of Economic Affairs.

- SDP elements always contain clients (service users), servers (service containers) and a service-repository. In distributed architectures, such as UPnP, the repository's job resides in the server.
- The SDPs contain a uniform grammar and semantics of service and device descriptions that enable a client to learn a newly deployed server without prior knowledge of the server, and then to use its services.
- Automated service discovery is done by *announcement* (except Bluetooth SDP [7]) and *query* [5]. An announcement lets the server advertise its services to the network and a query lets the client actively discover a desired server.
- Dynamic server information (state or variable change) is updated to any clients with event subscription and notification.
- The server lifetime is leased and should be renewed periodically (not available in Salutation [4]). This prevents deadlocks and inconsistent server states whenever the server has been unintentionally out of service.

Among the SDPs mentioned, Home Audio Video Interoperability (HAVi) [8] is the only one that provides a form of resource management. HAVi is a relatively advanced SDP and designed especially for audio and video related devices. The resource manager lets clients reserve resources, release resources and arrange scheduled actions. These scheduled actions identify a time when the resource is reserved, and can be appointed to the interested clients to prevent potential conflicts. Reserved resources may be preempted by a negotiation with the resource owner, which may be a person. Resource reservation by human intervention has always priority over automatic reservation by the system.

Unfortunately, the HAVi resource reservation concept cannot be easily adapted to other SDPs. Other SDPs mostly rely on the Internet Protocol (IP), whereas HAVi is built on the IEEE 1394 bus standard. Consequently, services based on other SDPs cannot participate in HAVi resource reservation. Furthermore, HAVi resource management works in a distributed fashion. It requires an extra resource management software element on all participating devices in the network, thus introducing problems in terms of backward compatibility and upgradeability, and additional costs to every single device.

III. TRAFFIC GENERATION BY THE CONFLICT

A significant consequence of resource conflicts is the generation of unnecessary traffic. To quantify this effect, we made an analysis of the conflict assuming a network that consists of a server and multiple clients. A conflict arises when the clients compete for access to the server. To ease the analysis, some assumptions are made:

- All clients which access the server will generate configuration packets that consist of controlling and response messages.
- The clients' requests arrive at the server at random times ($t_1, t_2, \dots, t_i$), represented as random time functions T_i .

Figure 1. Basic operation of the reservation manager

- The server does not have a resource manager. Therefore, server access granted to a particular client can easily be interrupted by any other client.
- Without awareness of the conflict, each client will always retry to regain the server session after a timeout T_o .

The first conflict happens when a second client sends an access request at t_2 while the first client has not finished using the server. As a result, the first client will lose the server session and try to re-access the server by sending another access request after T_o . Later, before the first client re-accesses the server, the third client has sent an access-request and replaced the second client's server session. Similarly to the first client, the second client will also retry to re-access the server after T_o . The conflict will continue and become worse with increasing number of clients.

The total traffic generation because of the conflict during a server lifetime T_s is given in equation (1), which can be obtained from the total number of incidents at which the clients send configuration packets, multiplied by the configuration packet size (CP). The first term shows the initial configuration traffic because each client sends a configuration packet for the first time. It corresponds with the number of clients n . The following term shows the unnecessary overhead traffic because each client will try to get server access by sending a configuration packet periodically at an interval of T_o until the end of T_s . Equation (1) assumes that there is more than one client in the conflict, that all the clients have the same T_o , and that CP is the same for each client and request.

$$Traffic = \left[n + \sum_{i=2}^n \text{int} \frac{T_s - T_i}{T_o} \right] \cdot CP \quad (1)$$

IV. RESOURCE RESERVATION MANAGER DESIGN

A. Concept

The basic operation of the reservation manager is shown in Figure 1. The network consists of a client, a server and the reservation manager. The solid lines represent the physical connection to the network and the dashed lines show the exchange of messages. First, the server registers its services to the reservation manager and the clients, which is done using the SDP's native advertisement mechanism. Upon receiving the information, the reservation manager registers/updates the (new) services in its local service registry database. Each service registry includes, amongst others, universally unique identification, service state (explained in section IV.B), service

user, service lifetime, and user reservation time (i.e. how long the service will be engaged by the service user).

Then, the client will query the reservation manager to reserve access to the service for a given duration. The reservation manager will check the availability of the requested service and whether it is possible to be reserved, which depends on the reservation policy present on the reservation manager. In case the service can actually be reserved, the reservation manager will respond the request with a “successful reservation” message which also contains the duration of the reservation. Otherwise it will return a “failed reservation” message which may contain a duration after which access can be retried. Additionally, the client may want to extend the reservation. For this, the client should reserve the service again before the reservation expires. This is the same mechanism as the leasing concept mentioned in section II.

Finally, the client may stop using the service before the reservation ends. The client then must release its service reservation. There are two ways for this. First, the client sends “release reservation” to the reservation manager. The manager then changes the reservation state and informs the other potential clients. Thus the other potential clients do not need to wait until the suggested retrial time for reserving the service. Second, the service reservation of the client will finish by itself after the reservation duration has ended. Because the client does not renew the reservation, the service state automatically becomes available.

B. Service State Diagram

To provide the clients with useful service state information, a service state diagram is proposed as depicted in Figure 2. The state diagram shows different service states and relevant transitions. A service is *Unavailable* when it is known by the reservation manager but not ready for accepting any jobs. A service becomes *Ready* when it is known by the reservation manager and all clients in the network and is ready for accepting any jobs. Finally, a service is *Reserved* when it is known by the reservation manager and all clients in the network, but its access is only granted to the client who has reserved it.

A new service is *Unavailable* from the start (1). The service will become *Ready* (2), if its server has advertized it to the network. This will change to *Reserved* (3), if there is a successful reservation by any client. The service returns to *Ready*, if the client ends the reservation (4) or it does not renew the reservation (5). Alternatively, the reserved or ready service can become *Unavailable* (6 and 7), if the server becomes inactive intentionally or accidentally.

C. Reservation Management

Let's first assume that the reservation policy is based on a fair context, namely “first come first served”. The earlier a client requests a reservation, the more priority it gets. Consequently, the later incoming clients will be put in a queue and are suggested a retrial time by the reservation manager. The retrial time T_{ret} of a client is given in (2) as a function of the client's position in the queue (i), and is an accumulation of earlier clients' reservation times T_{res} .

$$T_{ret}(i) = \sum_{k=1}^{i-1} T_{res}(k) \quad (2)$$

As each client i may reserve the service for a different duration, the reservation manager should keep these individual durations with the clients' positions in the queue. Furthermore, it is likely that any client may want to continue its service reservation before the current reservation ends. Then the client should renew the reservation before the reservation gets expired. Alternatively, the client may release the reservation. For those two conditions, the reservation manager must update the queue table and disseminate the new information to the other clients.

Including this reservation management, the total configuration traffic until the last client accesses the server can be predicted by (3). The total configuration traffic is now given by the number of clients n each generating one configuration packet (CP) to access the server and one reservation packet (RP) to reserve the service (including the response), and additionally m announcement packets (AP) that are generated if any client stops or extends the reservation before the reservation gets expired.

$$Traffic = n.(RP + CP) + m.(AP) \quad (3)$$

The reservation policy can be easily extended with any other more advanced contexts such as bandwidth availability [9] and identity of the client. Of course this will then affect equations (2) and (3).

D. Requirements

A reservation manager as defined above does fulfill the requirements as given in Section I. Requirement 1) is fulfilled when the reservation manager as depicted in Figure 1 is based on a centralized architecture and makes use of native SDP messages only. This is the most innovative part of this work. We do not impose modification of any standard servers (devices). The reservation manager can be added to an existing SDP network as a new service without any additional software or configuration of the current server specifications. However, we do require a modification in the client behavior. To have effective and consistent service-access reservation, every client in the network needs to request a service reservation from the reservation manager prior to accessing the service. The client must refrain itself from accessing the service whenever the requested service is reserved by another client. To eliminate this client requirement, the reservation manager may be combined with the authentication mechanism provided by the SDP, like the UPnP security ceremony [10]. Only clients that have been granted permission can then access the reserved

Figure 2. A simple service state diagram

service. This permission should then not be based on identity only, but also on other client properties such as the ability to request and release reservations.

Requirement 2) is fulfilled for all SDPs that support the functions as bulleted in Section II. As mentioned there, Bluetooth SDP does not define any advertisement protocol. To monitor service availability anyway, the reservation manager should then periodically invoke the server. This is a fairly complicated and bandwidth consuming work around though.

The third requirement is fulfilled if the reservation manager is implemented in a remotely manageable device, such as a residential gateway [11, 12] or a mobile phone.

V. IMPLEMENTATION ON UPnP

A. UPnP basics

UPnP [3] is an interoperability framework for devices and services in a relatively small-scale IP network. It is based on the client/server model and distinguishes three logical entities in the network: UPnP Services, which represent the service functionality of a device, UPnP Devices, which act as services servers, and UPnP Control Points (CPTs), which act as clients for controlling the services. UPnP defines the use of Dynamic Host Configuration Protocol (DHCP), Simple Service Discovery Protocol (SSDP), Simple Object Access Protocol (SOAP), and General Event Notification Architecture (GENA) for addressing, discovery, control, and eventing, respectively. Device and service descriptions are standardized eXtended Markup Language (XML) templates.

B. Architecture

Figure 3 shows the architecture of our reservation manager implementation on UPnP. The architecture is composed by four modules, namely a Service Discovery module, a Database, a Decision Engine, and a User Interface module. The modules function as follows:

The Service Discovery module is responsible for monitoring service advertisements and for listening to any client request for service reservation. This module contains all basic UPnP elements, i.e. a CPT and a server incorporating a reservation manager Device with a corresponding Service. In our implementation, this module is developed using a Java UPnP toolkit from Cyberlink [13]. The CPT will discover any server in the network using standard service discovery operations (i.e. *search* and *listen*) and then registers it to the database. The Device periodically advertises the reservation manager Service to the network, and it listens to any service invocation. The reservation manager Service describes methods, arguments and variables to reserve any registered services.

The Database contains the reservation information and provides an interface to manage it. The information management includes verifying and storing new service information, querying reservation information, and updating the service states. Each service advertisement being received is related to one service registry that contains the following fields: server device identity (universally unique), service identity

(locally unique within the server), service state, service lifetime, reservation time, and service user. The device identity and service identity are used to uniquely distinguish services including the ones from the same server. The service user describes the client identity that reserves the service.

The Decision engine defines the policy for handling reservation request.

The User Interface is a module that displays the current configuration and enables the remote configurability of the reservation manager server. The configuration may include the deployment of new decision policies, the modification of server leasing times, etc.

VI. PERFORMANCE ANALYSIS

A. Reducing overhead traffic

By preventing service access conflicts, unnecessary traffic generation evolving from this conflict can also be prevented, therefore reducing the total average network bandwidth occupation by control traffic. To quantify this, we created a scenario, calculated the amount of traffic created by incidents without a reservation manager present in the network, using (1), and compare it with the same situation but now including a reservation manager, using (3).

The scenario is the following. There is one desired server, which is accessed by $n = 2-10$ clients that randomly request the server with an average arrival rate of $\lambda = 5, 10$ and 20 clients per hour. Each client will reserve the server averagely for $T_{res} = 2$ minutes, and the server life time T_s is set to one hour. Each client will need $T_o = 5$ s to retry to access the server upon interruption. Additionally we assume that each client changes its reservation once during the given duration. We also assume that data packets for configuring the server, reserving the server and announcing the possible reservation changes (*CP*, *RP*, and *AP*) have the same size.

Figure 4 depicts the results of the two calculations. The figure shows the number of incidents as a function of the number of clients in the network. The number of incidents of the system is defined as $Traffic/CP$. Without reservation manager it is given by the left hand part of (1), with the client random arrival T_i , assumed to follow a Poisson process (4) with

Figure 3. The architecture of our reservation manager implementation based on UPnP.

Figure 4. Number of incidents in a system with and without reservation manager upon conflict situation

interval duration T_s and arrival rate λ .

$$T_i = \sum_{j=2}^i \frac{(\lambda T_s)^j \cdot e^{-(\lambda T_s)}}{j!} \cdot T_s \quad (4)$$

Without reservation manager, the number of incidents increases significantly with increasing client number and increasing client arrival rate. With reservation manager, the number of incidents increases only slightly with client addition (in our case it equals $3n$), and is many orders of magnitude smaller than without reservation management.

B. Application overhead

The reservation manager is an active device in the network, and will introduce extra application overhead to the existing system. This overhead mainly consists of extra control traffic generated by reservation manager advertisements and latency for the client application as each client needs to reserve a

service before using it. To understand the size of this overhead, we performed some measurements with our UPnP based reservation management system.

We measured the UPnP server advertisement packet for the reservation manager to be 360 bytes. Furthermore, as UPnP recommends, to minimize packet loss effects, the server should repeat the same advertisement three times. Thus, every advertisement generates 1080 bytes of traffic. If the lease time is set to 30 minutes, then the traffic overhead is fairly small (5 bps). However, as the number of clients in the network increases, they will invoke service discovery, and the number of advertisement activities by the reservation manager increases linearly, as shown in Figure 4.

To get an idea of the client application latency introduced by the reservation manager, we measured the added delay for finding the reservation manager service (i.e. discovery delay) and for reserving the service (i.e. reservation delay). We have measured the discovery delay using two UPnP methods i.e. *Search:All* and *Search:uuid*. With *Search:All*, the client will trigger all available servers to multicast their services randomly within a given maximum time (i.e. MX time). The *Search:uuid* is a discovery method where the client only triggers the identified server (by a unique identity) to unicast its services within a given MX time. In our implementation, the servers and clients are connected by 100 Mbps full duplex Ethernet. We assume up to 100 UPnP devices in the network. For minimizing the sampling error, we measured the discovery delay 100 times for each UPnP device. We set the MX time to 3 seconds for both discovery methods. Figure 5a shows the average discovery delay using the two methods as a function of the number of devices. Both searching methods give responses within 3 seconds, independent of the number of devices, and the average discovery delay of both service discovery methods is 1.6 seconds. This means that the chosen MX time (i.e. 3 seconds) is sufficient for avoiding scalability problems.

For the reservation delay, we measured the time of the reservation manager to respond to the client's reservation request. Figure 5b shows the reservation delay as a function of the number of UPnP devices. Each measurement was repeated 100 times. The reservation delay is found to increase slightly with the number of devices in the network. This can be

a)

b)

Figure 5. The overheads of the reservation manager: a) search delay, and b) reservation delay

explained as follows. The total reservation delay consists of the network transmission delay (roundtrip) and the application delay introduced by the reservation manager application. From a separate investigation, the transmission delay dominates the total delay and is constant, while the application delay causes the increment of the total delay. The latter is caused by the fact that every device in the network is represented by a separate Java object in the reservation manager, leading to increasing processing time when adding devices.

VII. CONCLUSIONS AND FUTURE WORKS

This paper presents and discusses a remotely manageable resource reservation manager concept for existing service discovery protocols. The reservation concept allows a client to query service availability before using it. Whenever available, the client can reserve the service and later release its reservation. As such, conflicts and contention for service access can be prevented from happening. Also unnecessary reconfiguration traffic evolving from these conflicts can be avoided.

As the concept can generally work in every SDP, it can potentially manage resource reservation across SDPs. Therefore, network gateways seem to be suitable for hosting the reservation manager. Also the fact that current gateways already have remote management functionality makes them attractive.

The most innovative part of this work is that the reservation manager benefits from common operations of existing SDPs. Therefore, it does not require modification of current standard networked devices and is, thus, backward compatible. However, the reservation manager concept does require clients to cooperate.

From our performance evaluation based on a UPnP platform, it can be concluded that the presence of the reservation manager introduces only a limited amount of extra control traffic on the network and a relatively small reservation application delay. Furthermore we found that the reservation manager reduces the number of conflict incidents drastically. Therefore, it could mitigate the bandwidth occupation significantly and raise the system scalability.

In the near future, we will combine our reservation manager with current QoS technologies that are based on traffic differentiation [14]. Service access can then be made dependent on the current state of the network. Also, the traffic priority can then be made dependent on service availability.

REFERENCES

- [1] H. Yamazaki, Z. Zhang, and I. Yoda, "Resource Management Technologies for Home Network Services", *Journal on NTT Tech. Rev.*, vol. 3, pp. 12-16, 2005.
- [2] R. Lea, S. Gibbs, A. Dara-Abrams, and E. Eytchison, "Networking home entertainment devices with HAVi", *Computer*, vol. 33, pp. 35-43, September 2000.
- [3] <http://www.upnp.org>
- [4] C. Bettstetter, C. Renner, "Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol", *Proceedings of the Sixth EUNICE Open European Summer School*, 2000.
- [5] F. Zhu, M. Mutka, L. Ni, "Classification of Service Discovery in Pervasive Computing Environments", *IEEE on Pervasive Computing*, vol. 4, pp. 81-90, October-December 2005.
- [6] G. G. Richard III, "Service Advertisement and Discovery: Enabling Universal Device Cooperation", *IEEE Internet Computing*, October 2002.
- [7] <http://www.bluetooth.org>
- [8] HAVi specification version 1.1, <http://www.havi.org>
- [9] F. Giovanelli, G. Bigini, M. Solighetto, "A UPnP-based bandwidth reservation scheme for In-Home Digital Networks", *Proc. On International Conference on Telecommunication (ICT)*, vol. 2, pp 1059-1064, 2003.
- [10] C. Elisson, "UPnP Security Ceremonies Version 1.0", *Design Document*, UPnP forum, October 2003.
- [11] F. Den Hartog, et al., "Convergence of Residential Gateway Technology", *IEEE Communications Magazine*, pp. 138-143, May 2004.
- [12] Home Gateway Initiative Requirements Release 1, www.homegateway.org
- [13] S. Konno, "Cyberlink: Development Package for UPnP Devices for Java", <http://www.cybergarage.org/net/upnp/java/>.
- [14] M. van Hartskamp et al., "UPnP QoS Architecture v 2.0", UPnP forum, October 2006.