A Service Discovery Infrastructure for Heterogeneous Wired/Bluetooth Networks*

Elena Pagani, Stefano Tebaldi, and Gian Paolo Rossi

Computer Science Dept., Università degli Studi di Milano
pagani@dico.unimi.it, stefano.tebaldi@unimi.it,
rossi@dico.unimi.it

Abstract. Specifications of some applications based on the Bluetooth protocol stack have already been standardized by the Bluetooth Special Interest Group. Those applications can however be accessed inside a piconet, while Bluetooth devices belonging to different piconets are not able so far of cooperating in a distributed application. In this paper, a framework is proposed that allows a Bluetooth device to search for services provided by other devices, also in case they are connected to different piconets that communicate through a wired network. The proposed framework has been implemented on a testbed platform and its suitability has been tested by using it to distribute a file transfer service over a heterogeneous network.

Keywords: Bluetooth, heterogeneous networks, service discovery and access, testbed deployment.

1 Introduction

The Bluetooth technology [1,2] has been designed with the purpose of replacing cabling amongst neighbor devices, for instance to connect computers and mobile phones to external devices and accessories via wireless links. Ongoing standardization of Bluetooth makes it an interesting solution to rapidly deploy wireless infrastructures using low cost, easily available devices such as PDAs, notebooks and cellular phones.

Application characteristics have been specified [2], that allow to deploy distributed services based on the Bluetooth protocol stack. Those applications are not compliant with the TCP/IP protocol stack and so far they can involve only Bluetooth devices (BDs) belonging to the same piconet. The usage of those services is supported by the Service Discovery Protocol (SDP), which allows to discover information about how to configure the Bluetooth protocol stack to interoperate with another device; SDP is only usable inside a given piconet. As a consequence, a BD is not able to establish a session with a remote BD connected to a different piconet.

This work presents *BSDA (Bluetooth Service Discovery and Access)*, an infrastructure that supports communication sessions between two BDs belonging to different piconets that are connected through a wired network. An extension of SDP is proposed

* This work has been partially supported by the Italian Ministry of Education, University and Research in the framework of the FIRB "Web-Minds" project.

that allows a BD to retrieve information about the services available on other BDs, independently of their current location, and about the protocol stack configuration needed to exploit those services. This extension includes mechanisms to deal with BDs dynamically changing the piconet to which they are connected over time.

2 Related works

In literature, some infrastructures have been proposed that provide device and service location functionalities.

Service Location Protocol (SLP) [3] has been proposed by the IETF to discover the services available in an intranet. It supplies the IP address and port number of the hosts providing the service. It is suitable for wired networks, while supporting host (and thus, service) mobility involves a large control overhead to update the location information. SLP supports the discovery of services defined by IANA and identified by a URL, and it is based on the TCP/IP protocol stack.

JINI [4] allows to access Java programs, or Java applications supplied by devices running a JVM. The service discovery is performed by multicasting a request to lookup servers. The service is accessed by downloading from a lookup server the Java code either implementing the service, or implementing the interface to communicate with the remote server.

Salutation [5] supports service and device discovery and access in presence of highly mobile system participants. Salutation is independent of underlying communication protocols and end host architectures, but for each transport protocol adopted in the network it requires a Transport Manager able to interoperate with that protocol. Service discovery and access is supplied by a pool of Salutation Managers (SLMs), which are organized in a backbone. The SLMs represent the system bottleneck as they have in charge both the provisioning of users with information about the available services and the set-up of the client-server connections. Mechanisms must be deployed to allow users to discover SLM addresses.

Microsoft developed *Universal Plug and Play* (UPnP) [6] for the service and device discovery in LANs. To exploit UPnP, a host has to execute the control point code, that allows to discover the available services and access them on the remote devices. UPnP can be used for services based on the TCP/IP protocol stack. Bluetooth devices could participate in UPnP infrastructures through bridges supporting the interoperability between the two systems.

2.1 Service Discovery Protocol (SDP)

SDP [1] supplies functionalities to discover the services available in a Bluetooth piconet. A service is described via a *service record*, which contains a list of service attributes. These service attributes are described in the corrisponding profile [2], and their knowledge is needed to exploit the service. The most important attributes are the *ServiceClassIDList* and the *ProtocolDescriptorList*. The *ServiceClassIDList* is a list of *Universally Unique Identifiers* (UUIDs) identifying the service classes that the service implements, while the ProtocolDescriptorList describes the Bluetooth protocol stack to be used to access the service, and the needed parameter configuration.

Any Bluetooth device carrying services also runs an *SDP server* to make those services available to other hosts. The SDP server maintains the records for the services existing on the device identifying each record with a unique handle, and it interacts with the *SDP clients* residing on devices willing to access remote services. A client looks for a service by specifying a *service search pattern* that is the list of UUIDs that must characterize the service. A service record matches the search pattern if it contains all the UUIDs listed in the pattern. A client may as well browse through the services available on a given device. The services belong to *groups*, each one of which is identified by a UUID that is recorded in the service records: the groups are arranged in a hierarchical structure. A client can discover all the services belonging to a group by using the group UUID as the search pattern.

SDP adopts a request/response communication scheme; three kinds of requests can be generated by a SDP client:

ServiceSearchRequest: it is used to locate services whose records match the search pattern;

ServiceAttributeRequest: it is used to retrieve attribute values from a specific record, characterized via its handle;

ServiceSearchAttributeRequest: it combines the functionalities of the two requests above, that is, it allows to retrieve attribute values concerning the services satisfying the search pattern.

Requests are sent to a specific Bluetooth device, by having the SDP client that establishes a L2CAP connection with the SDP server. A SDP client could browse through the services available in its communication range only by performing the inquiry procedure to discover the BDs in range, and then by repeating the above step for each one of them separately.

3 Bluetooth Service Discovery and Access (BSDA)

BSDA (Bluetooth Service Discovery and Access) allows a BD to make its own services available to other BDs, and to search for services provided by other BDs, independently of their geographical location. In particular, the involved BDs may belong to different piconets, connected through a wired network. BSDA is based on SDP. With BSDA a BD is able to both browse through the services offered by a specific BD, and characterize all the BDs providing a given service. Moreover, a BD is able to retrieve the service record describing a particular service, and as a consequence to actually exploit the service.

In Fig. 1, we show the system we consider. BDs are connected in piconets. The role of master is assigned to a device equipped with a wired network interface, thus also acting as Access Point (AP) to the fixed network infrastructure. SDP is exploited by BDs inside piconets to find the AP and to exchange information about the available services. The service discovery functionalities are mainly in charge to the Home Agents (HAs). A Home Agent is a fixed host in Internet that works as a central repository for information about the services offered by BDs and is queried to discover which services a BD supplies. More than one HA can exist in the system. Each BD is associated with one HA and an HA can serve more than one BD. The AP allows a BD to interact with the HAs to advertise its own services and discover services of other BDs. The communication between two BDs occurs through the APs to which they are connected. Usually, communications between an AP and an HA are based on the UDP protocol. Yet, long messages[1] could be generated, that do not fit into an UDP message. In this case, TCP is used instead to avoid problems related with message fragmentation. When exporting

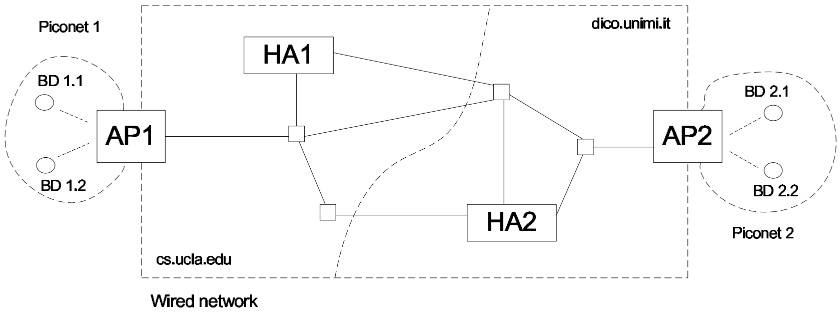

Fig. 1. BSDA operational environment

Bluetooth services in Internet, an addressing scheme must be devised alternative to the 48-bit BD address used inside piconets. That scheme must be independent of the current BD's location, while it can be a mnemonic name that allows users to easily indicate the BD whose services are searched for. As the naming scheme we use a URL, that is associated to each BD, and has the form: `"Bluetooth://"<HAAddress>"/"<id-device>` ; with <HAAddress> either the symbolic hostname or IP address of the Home Agent which maintains the information about the BD, and <id-device> the symbolic name for the specific BD. As an example, a user Bob working at the Computer Science Dept. of the UCLA may have a PDA equipped with a Bluetooth interface card, and exploiting the local Home Agent, whose associated URL is `"Bluetooth://HA1.cs.ucla.edu/BobPDA"`.

BSDA includes procedures to publish the services supplied by the BDs, to search for services provided by other BDs, and to establish a session through the wired/wireless network to exploit a service. For the sake of simplicity, we initially assume that BDs do not move. We then explain how the protocol copes with BDs that dynamically change the piconet to which they belong.

A BD wishing to make its local services available through Internet, besides of storing the appropriate service records in its own SDP server, must advertise them to its HA, via its own AP (*Service Registration* procedure). The BD requests its AP to register the BD's services at the appropriate Home Agent, by sending it a `Registration` message containing the BD's URL. Upon reception of this message, the AP sends an SDP request to the BD to discover all the services the BD makes available. The ser-

[1] E.g., messages carrying information about several services.

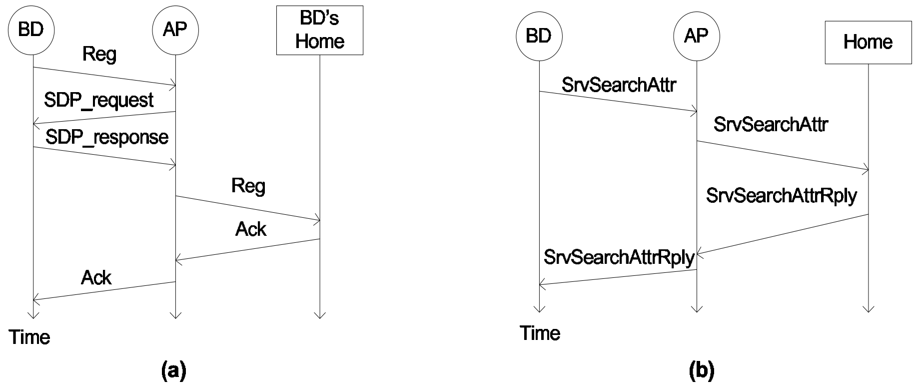

Fig. 2. (a) Registration procedure. (b) ServiceSearchAttribute procedure.

vice records obtained this way are used by the AP to build a `Registration` message that the AP sends to the HA indicated in the BD's URL, thus replicating them at the HA repository. In Fig. 2(a), we show the communication pattern for the registration procedure.

The HA repository can be queried by exploiting BSDA when a BD wants to access a service of another device (*Service Search* procedure). Two kinds of searches can be performed: either a BD can perform a query about the services supplied by a specific BD, or it can request an HA the identities of all the BDs registered at it that provide a certain type of service. In both cases, the BD directly communicates with its own AP, that then takes in charge the communication with the appropriate HA. In the former case, the service client BD (BD_c) sends to its own AP (AP_c) a `ServiceSearchAttribute` message, containing the URL of the BD searched for and an SDP service search pattern. This message is a modification upon the homonymous SDP message. The URL indicates AP_c which is the HA to contact, while the SDP search pattern is the set of UUIDs to be used to select the appropriate service records among those the HA owns concerning the BD server indicated in the URL. As an example, consider a user Alice whose portable PC PC_A exploits the HA at the Computer Science Dept. of the University of Milan. If Alice wants to perform a file transfer to Bob's PDA, a connection must be established between PC_A and its current AP. Then, the PC_A's `ServiceSearchAttribute` message must be forwarded by that AP to HA1.cs.ucla.edu, *without* the need of contacting the PC_A HA. In Fig. 2(b), we show the communication pattern for the processing of the `ServiceSearchAttribute` messages. The `ServiceSearchAttributeReply` contains the address of the AP to which the BD searched for is currently connected, and the service records satisfying the query.

In case a device wants to discover all the devices registered at a given HA and supplying certain services, the client BD sends to its AP a `ServiceSearch` message containing the URL of the HA to be contacted and the SDP search pattern that the service records must match. The client may as well indicate the maximum number of matches it wants in response. The communication pattern is similar to that of Fig. 2(b).

The `ServiceSearchReply` message contains the URLs of the BDs satisfying the query; the client BD may then refine the search by issuing a `ServiceSearchAttribute` for one of the obtained URLs. Once a client BD has obtained the address

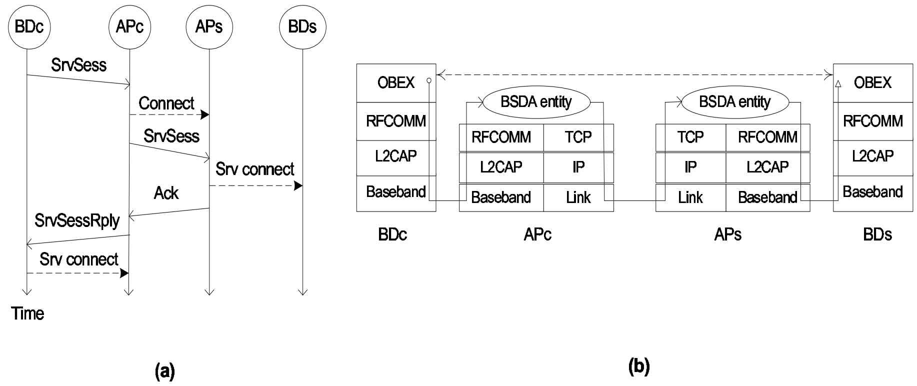

Fig. 3. (a) ServiceSession procedure. (b) Protocol stacks and connections used during an OBEX Object Push session.

of the server BD's current AP, and the service record with the profile of the service the client wants to exploit, a session can be set up to access the service. Let BD_c be the client BD with access point AP_c and BD_s be the server BD connected to the access point AP_s. In order to build a session, AP_c simulates to be the service entity for BD_c, and it connects to AP_s that in turn simulates to be the client entity for BD_s. To this purpose, BD_c sends to AP_c a `ServiceSession` message, containing the AP_s address, the BD_s URL and the service record of interest. AP_c starts a BSDA entity acting as server for the desired service, able to accept the BD_c connection request. Then, AP_c sends the `ServiceSession` to AP_s, which starts a BSDA entity acting as client, that establishes a connection with BD_s. If this procedure is carried out successfully, then AP_s sends an Acknowledgement to AP_c. AP_c sends BD_c a `ServiceSessionReply`, possibly specifying the correct session parameters, if they are different from those specified in the service record. This may for instance occur if the RFCOMM channel that should be used to connect to the server[2] is already in use on AP_c, and then AP_c has to use a different value. When BD_c receives the reply, it is able to set up a session with AP_c and to use the service. All the service messages sent by either BD_c or BD_s are received at their respective APs, where they are encapsulated by the BSDA entities to be forwarded to the other AP, which decapsulate the service message and transmits it to the local BD. The described procedure is shown in Fig. 3 (a). It is worth to notice that the BSDA entities are completely unaware of the service characteristics and implementation details. Their unique task is to appropriately encapsulate and decapsulate messages to transfer them between heterogeneous protocol stacks (Fig. 3(b)).

[2] According to the service record

As an example, let us suppose that a BD_c wants to utilize the OBEX Object Push capability [2] of another BD_s that is not in communication range, but that is available through AP_s. BD_c sends its own AP_c a `ServiceSession` request containing the OBEX Object Push service record previously obtained via a `ServiceSearchAttribute` request. This service record may be as follows:

```
Service Name: OBEX Object Push
Service Class ID List:
"OBEX Object Push" (0x1105)
Protocol Descriptor List:
"L2CAP" (0x0100)
"RFCOMM" (0x0003)
    channel: 4
"OBEX" (0x0008)
```

AP_c then sets up an RFCOMM server listening on channel 4, as indicated in the `Protocol Descriptor List` attribute. This RFCOMM server allows BD_c to establish the service connection with AP_c. AP_c then establishes a connection with the remote AP_s to which the server BD_s is connected and sends it the `ServiceSession` message. The receiving AP_s establishes an RFCOMM connection with the local BD_s by using the information contained in the service record and replies with an `Ack` message. When the source BD_c receives the `ServiceSessionReply` from its own AP_c, it connects to AP_c and starts using the service. Figure 3(*b*) shows the protocol stacks and the established connections among the entities participating in the session. The OBEX client does not create the RFCOMM connection with the real OBEX server, that is not in communication range, but with its own AP that will send data received from the OBEX client to AP_s to which the OBEX server is connected. AP_s will send the data to destination.

3.1 Mobility support

So far, we assumed that BDs never move. When a BD moves from one piconet to another, its AP changes and as a consequence the information maintained on its HA becomes stale.

To prevent the usage of outdated information, the entries in the HA repository are maintained according to a soft-state approach: an AP must periodically send a `Registration` message to the HAs of the BDs it is currently connected to, or those BDs information is removed. If no change is needed to the information held by the HA, those periodic messages could contain just the URL of the BD whose entry must be refreshed, thus saving bandwidth. This mechanism guarantees that old information is deleted from the HA also in case the AP crashes or it becomes disconnected. On the other hand, if an AP notices that it lost the connection to a BD, it may expedite the updating of the corresponding HA information by sending it a `Deregistration` message, which provokes the BD's entry deletion. This message can be as well generated by a BD requiring its AP to remove it from its HA, because it is not anymore willing to provide other BDs with its own services.

In spite of these mechanisms, it is not guaranteed that an AP does not contact another AP not anymore supporting a given BD, because of the network latency. Similarly, an HA may receive a request concerning either a BD which deregistered, or a BD whose entry is expired. If an AP receives a message addressed to a BD currently not connected to it, then the AP issues an error message in reply. If an HA receives a query concerning a BD such that either is currently disconnected or the HA has no information for it at the moment[3], the HA replies with either a `ServiceSearchReply` or a `Service-SearchAttributeReply` containing an error code. In both cases, the client should perform a retry after some time, when the information about the sought BD has been updated.

4 Protocol implementation and testing

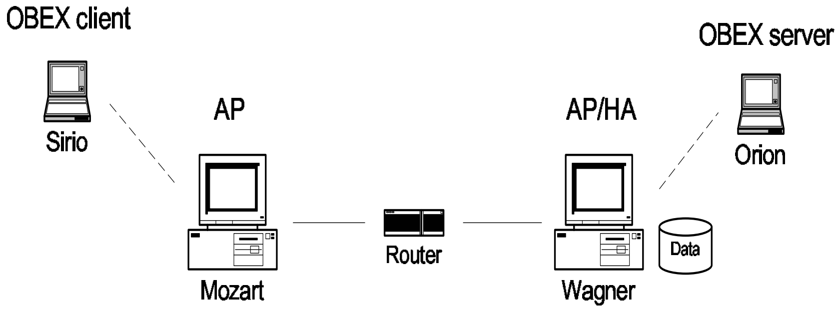

Fig. 4. Layout of the test network

We implemented the proposed solution on a testbed platform. We use two notebooks and two desktop PCs each of them equipped with a 'Ericsson ROK 101' Bluetooth module. The notebooks work as BDs and the PCs as APs. One of the PC acts also as Home Agent for the BDs. In Fig. 4, we show the layout of our testbed network. All the four machines run RedHat Linux with Kernel 2.4.18 and use BlueZ (`http://bluez.sourceforge.net`) as the Bluetooth protocol stack. We tested our protocol with the OBEX Object Push service. `Sirio` acts as OBEX client and `Orion` as OBEX server. The goal is to transfer a file from `Sirio` to `Orion` even if they are not directly connected. BlueZ does not include the OBEX protocol, thus we used OpenOBEX (`http://openobex.sourceforge.net`). All the software modules that constitute the system were implemented in C. In the following, we provide a brief description of the testbed implementation.

Two relevant data structures have been implemented, namely the HA repository and a structure recording the characteristics of the connections an AP has active with its own BDs and possibly other APs. The communications channels shown in Fig. 4

[3] The BD's status is expired at the HA.

are implemented through sockets. The information maintained in the HA repository for each registered BD is: the BD's URL, the address of the associated AP, the list of its services, a timestamp indicating the registration time and the active flag indicating whether the registration is still active or not. If the active flag is 0, the HA can either remove the entry, if it needs memory, or maintain it and update it when the corrisponding BD re-registers again. In the latter case, the non-active entries are not considered in replying to the queries. The entry fields are initialized upon the reception of a `Registration` message. The timestamp value is obtained using the `gettimeofday()` function. An entry is considered valid if its timestamp is lower than 120 seconds. Every service session involves two connections: one between two APs, that is a TCP (or UDP) connection and one between AP and BD, that is a Bluetooth connection. As a consequence, every AP needs a mechanism to associate these two types of connection to properly forward the data among them. Each AP stores a list containing a mapping of wired and Bluetooth sockets for each service-related connection. An AP monitors all its network connections exploiting the *select()* system call. Based on which socket becomes active the AP acts accordingly. At the bootstrap, `Sirio` and `Orion` have to

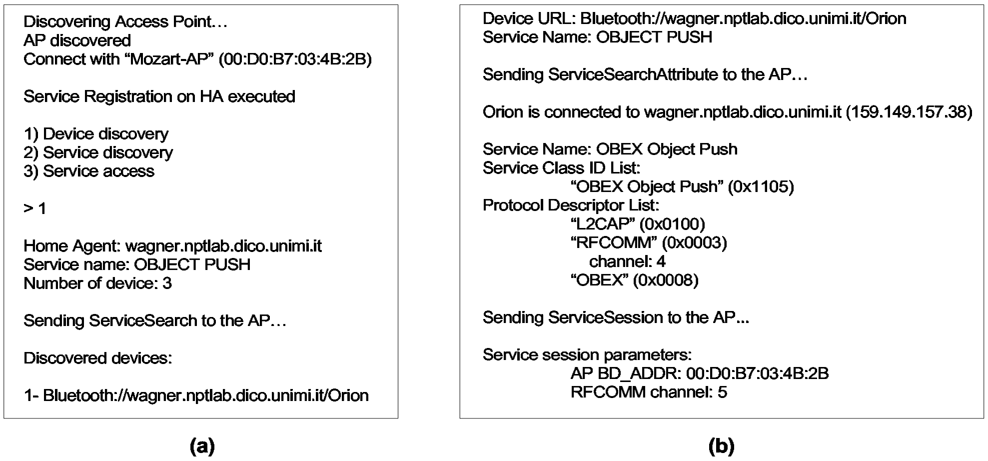

Fig. 5. BD command interface example: (a) `Sirio` registration and device discovery; (b) service search and access on `Sirio`.

find an AP. When the connection with the AP is established, a command interface is shown to the user. Using this interface, the user can exploit BSDA to discover BDs having desired services, to obtain specific information on these services and to exploit them.

In Fig. 5, we show the user interface provided by a BD. In Fig. 5(a), the `Sirio` user chooses to discover which BDs have the OBEX Object Push service. It provides the HA address and the name or UUID of the service and obtains the URLs of the BDs that match the search pattern (in this case only Orion). In Fig. 5(b), the BD obtains the specific information about the service and asks its own AP to establish a service session.

As we can see, `Sirio` obtains in reply the needed parameters to exploit the service via `Mozart`. These parameters are used as input for the OBEX client code included in OpenOBEX.

5 Concluding remarks

In this work, we present an original infrastructure supporting the access of Bluetooth services by Bluetooth devices located in a piconet different from that of the server device. The infrastructure has been implemented on a testbed platform and its suitability has been analyzed by exploiting it to guarantee the remote accessibility of a file transfer service based on the Bluetooth protocols.

As a future work, we plan to perform measurements with the testbed implementation, to perform a fine tuning of the timer used for the soft-state maintenance of the entries in the HA repositories. A peer-to-peer infrastructure could be designed, connecting the Home Agents, that would allow to extend the service search range. Moreover, we are studying the possibility of adapting the infrastructure to be used in a scatternet, that is, an ad hoc wireless network obtained by connecting several piconets.

References

1. Bluetooth Special Interest Group: *"Bluetooth V1.1 Core Specifications"*. May 2001, `http://www.bluetooth.org`.
2. Bluetooth Special Interest Group: *"Bluetooth V1.1 Profile Specifications"*. Feb. 2001, `http://www.bluetooth.org`.
3. Guttman E. and Perkins C. and Veizades J. and Day M.: *"Service Location Protocol - Version 2"*. RFC 2608, IETF, June 1999. Work in Progress.
4. Sun Microsystems: *"Jini Architectural Overview"*. Technical White Paper, 1999, `http://www.jini.org`.
5. The Salutation Consortium: *"Salutation Architecture Specification (Part-1)"*. Version 2.1, 1999, `http://www.salutation.org`.
6. Microsoft Corporation: *"Understanding Universal Plug and Play"*. White Paper, 2000, `http://www.upnp.org`.