Abstract—The use of devices embedded in the environment is limited by users’ ability to find them and configure their mobile devices to use them. Existing commercial implementations of device discovery technologies tend to be restricted to local networks and do not scale well beyond those environments. In this paper we introduce an architecture that extends Universal Plug and Play (UPnP) to address these limitations and therefore enable zero-configuration discovery and use of embedded devices across multiple networks.

Keywords—UPnP; device discovery; zero-configuration;

I. INTRODUCTION

Users of mobile devices such as smartphones are becoming accustomed to using their devices to connect to and use services offered by other mobile devices and devices embedded in the environment. Interactions with these other devices are supported by enabling technologies, such as near field communications (NFC), and information in the environment supplied using techniques such as 2D barcodes and augmented reality.

One key challenge in such a scenario is how users and their devices discover the devices that are available around them in an ad hoc manner. However, current networks and devices typically require configuration on discovery, and novice users may not understand what to do or know what information is required. Indeed, repeated requests for configuration may lead users to abandon devices that seem too much trouble.

Universal Plug and Play (UPnP) [1] is a zero-configuration discovery technology that specifies an architecture for discovery and interoperability of devices acting as containers for the services they provide. It has been widely adopted for various types of devices by manufacturers and has been extended to cover specific domains (e.g., the UPnP Audio Visual specification [2]). However, UPnP does not scale well beyond small numbers of devices, and is limited to communication on local networks.

By extending UPnP, we have implemented an architecture that tackles the challenge of zero-configuration discovery across wider settings and networks. In this paper we provide a brief overview of how UPnP works, identify the specific limitations we address, and examine related work in this field. We then describe our architecture and describe how our architecture enables UPnP to overcome its limitations, enabling ad hoc zero-configuration device discovery and use for mobile users of ambient devices.

II. UPnP OVERVIEW

UPnP works on networks that support multicast messaging. It provides a zero-configuration architecture that enables addressing, discovery and description for devices and the services they provide. UPnP is built upon several open source technologies such as the Simple Object Access Protocol (SOAP), HTTP and XML and is based around the concept of device as containers for services. UPnP devices and their associated services are generally defined as UPnP Device Control Protocols (DCP) published by the UPnP Forum [3]. Each service has a description that includes a default state, a number of events that can be subscribed to, and a description of the functionality of a service with actions that can be invoked at concrete end points. Devices may also contain embedded devices with services of their own.

UPnP devices that join a network advertise themselves to UPnP Control Points (CP) with the Simple Service Discovery Protocol (SSDP). These Control Points then coordinate messages between devices. Messages between devices are one of two types: (i) Control messages (such as invoking the action of a service) are expressed in XML using SOAP; (ii) Event messages, allowing a device to notify subscribers about state changes, are also expressed in XML using UPnP’s General Event Notification Architecture (GENA).

UPnP does not scale well to large numbers of devices and is limited to local networks for a number of reasons. UPnP is limited to local networks as device broadcasts use multicast messaging that is not commonly propagated beyond local networks. Devices outside of that network are then not aware when a new device becomes available. The scalability problem stems from the design of SSDP, as discussed in the last draft of the protocol [4]. The algorithm for calculating the bandwidth SSDP uses to discover services made available by UPnP devices is as follows:

\[
\frac{\left(DR \times 3 + DR \times 9 \times RS\right) \times AM}{TP} \tag{1}
\]

Where DR is the total number of SSDP clients making ssdp:discover requests over the time period TP. RS is the total number of services that will respond to the ssdp:discover requests over the time period. AM is the average size of the ssdp:discover requests and responses. 3 is the number of times the ssdp:discover request will be sent and 9 is the number of responses to the request that will be sent in the worst case. The
amount of data is proportional to the number of clients initiating ssdp:discover requests, multiplied by the number of services made available by UPnP devices. On small networks this can become a significant percentage of available bandwidth. If UPnP were scaled up to Wide Area Networks (WAN), this bandwidth would grow very large indeed. For example, consider a network of 100,000 clients (DR) sharing 5000 printers (RS) and a worst case scenario of a power outage causing all machines to refresh their service caches at the same time. Assuming requests and responses are evenly distributed over 30 seconds (TP), they would send in excess of 150 million messages per second. If we assume an average size of 512 bytes (AM) per request or response, these messages require 572.24 Terabits/second of bandwidth. Any attempt to scale UPnP beyond the local network must address this issue.

III. RELATED WORK

The wide adoption of UPnP technology and its zero-configuration nature make it an attractive candidate for providing device discovery in environments with embedded devices. A number of groups have looked at extending UPnP beyond local networks and how to address the scaling problem. The UPnP Forum has specified two versions of an Internet Gateway Device (IGD) [5]. They define an Internet Gateway as an “edge” interconnect device between a residential Local Area Network (LAN) and the WAN, providing connectivity to the internet. Such devices expose services that allow the listing of existing port forwarding mappings and the addition and removal of such mappings. This allows UPnP devices to be contacted by external entities. However, these gateways do not provide for discovery of UPnP devices; they simply expose ports that will map to existing UPnP devices.

Venkitaraman [6] describes an architecture for extending UPnP and sharing content using Digital Living Network Alliance (DLNA) specifications [7] across broadband wireless networks. They introduce an xUPnP application that acts as a gateway between local networks and as a proxy for certain UPnP messages. The objectives of this work were “to enable users to securely share content and control media flow between their devices independent of type of network connectivity” [6]. They concluded that their approach facilitated cross-network discovery of UPnP devices. Their xUPnP application acted as a proxy for mobile UPnP devices, conserving battery life and bandwidth. However, their tunneling method is limited to a single Network Address Translation (NAT) domain and so must assume devices on different local networks will not have the same IP address. In practice IP addresses on such networks are usually assigned automatically from a small number of well-defined address ranges [8], making an IP conflict more likely as more networks are connected. Their approach must therefore be modified to scale up to a truly WAN architecture.

Louati et al. [9] propose an architecture to allow UPnP to cross local networks, which they call UPnP clusters, and create a UPnP Virtual Space. They use a modified UPnP CP called a UPnP-INS Transcoder that communicates with a peer-to-peer (P2P) INS/Twine overlay network [10] providing inter-cluster routing. Louati et al. highlight some shortcomings of this approach, namely that it “suffers from a large number of generated keys and a problem of popular services/queries”.

They introduce a new architecture called a P2P-based Naming System for Service Discovery [11] that alleviates this problem. However, they still rely on making devices globally accessible: “As a UPnP device may be accessed from a remote CP for control, eventing and presentation purposes, the CP must know the global address of the Device. The gateway includes an application-layer NAT allowing UPnP devices to be exposed to the outside world.” [11]

The work outlined above aims to connect two devices by giving (at least) one of the devices a globally accessible IP address and port. This has the advantage that the implementing software need know little about what the devices are doing or the content of messages between them but this approach has problems. First, UPnP has no inherent security. While the UPnP Forum has published both a document describing how security should be implemented [12] and a DeviceSecurity:1 Service Template, UPnP devices are not bound to implement it. Generally speaking, if a UPnP device is listening on a port, anything can connect to it. Secondly, the approach assumes that externally accessible IP addresses can be obtained. This is not the case on many networks behind a firewall.

A further problem can arise when two devices use UPnP to negotiate a secondary channel using their own, local network IP addresses: communication after the negotiation will fail as the IP addresses will not be accessible. Our approach addresses each of these problems.

IV. EXTENDING UPnP BEYOND THE LAN

In order to address the problems of scalability, access to multiple LANs and security we introduce the architecture described below. This architecture allows a large number of devices on several LANs to connect via a WAN and negotiate connections between them in order to allow not only device-to-device communication but also discovery.

![Diagram](https://via.placeholder.com/150)

**Figure 1.** The architecture: Bridges on each of two LANs communicate via outgoing connections to the Dispatcher. Devices A and C are made available on the remote LAN as Mock Devices A and C. Bridges use UPnP on the LAN and our protocol agnostic messages between themselves and the Dispatcher.

We introduce a client/server architecture with a single server (a Dispatcher), and a client (a Bridge) on each of the
networks we wish to connect. As with Louati et al. [9, 11] and Venkitaraman [6], our client includes a modified UPnP CP.

Figure 1 illustrates the important features of the architecture. Each LAN hosting a number of UPnP devices contains a single Bridge. This Bridge includes a modified UPnP CP that provides discovery and communication between devices. Each Bridge contacts the Dispatcher and negotiates at least one outgoing connection. These connections are two-way, and a Bridge and the Dispatcher can communicate so long as outgoing connections are not blocked. A message between devices on different networks is sent via the Bridge to the Dispatcher, forwarded to the remote Bridge on the network hosting the target device, and from that Bridge to the device itself. In the following sections we describe in more detail the implementation of the Dispatcher, Bridge and the construct we use to provide inter-network availability, the Mock Device.

A. The Dispatcher

The Dispatcher sends messages between Bridges. The messages that can be sent may be classified into two groups (i) device advertisements, and (ii) inter-device messages. Device advertisements are broadcast messages, such as those detailing a new device joining the network. In existing local network service discovery protocols, advertisements are generally implemented as multicast messages. Our implementation includes two device advertisement message types: DeviceAdded and DeviceRemoved messages. These messages are sent by a Bridge to the Dispatcher when a new UPnP device becomes available or unavailable.

Inter-device messages are those messages between UPnP devices on different networks that are routed via Bridges and the Dispatcher. Such messages are sent to a specific destination or set of destination devices, e.g. one device invoking a method of a service on a second remote device, or a device informing remote subscribers that a state variable has changed. We use four inter-device messages: Inter-Device Request and Reply, and a specialized request and reply message type for streaming binary assets. Messages are usually wrapped up as Inter-Device Request and Reply types, but when binary assets are being transferred we do not need to change the content of the messages, so we use simpler message types that allow us to decrease the message overhead. These BulkRequest and BulkReply messages include only sections of the binary asset and necessary routing information.

The messages between Bridges concern UPnP devices and services but the Dispatcher knows nothing of the UPnP protocol itself. The Bridges wrap any protocol specific information and the Dispatcher deals with those messages on a higher level. For example, when a UPnP device becomes available, the local Bridge gathers all the necessary information to replicate that device at another Bridge and wraps it in a DeviceAdded message before sending it to the Dispatcher. The Dispatcher does not look at the protocol specific information, but only at the routing information necessary to forward the message on to other, appropriate Bridges.

This feature gives us a level of security not implemented in UPnP, i.e. any Bridge or device is only shared beyond the local network if we choose to do so. Even when a device is shared we have complete control over which Bridges it is shared to, and which devices can communicate.

B. The Bridge

Bridges provide device discovery and communication between LANs via the Dispatcher. They use a modified UPnP CP to discover local devices and make remote devices available through the creation of local Mock Devices. Each LAN we wish to connect requires a single Bridge on that LAN.

When a new Bridge comes online it initiates and negotiates a secure connection to the Dispatcher. This has the advantage that only the Dispatcher needs a globally accessible IP address, and local firewall issues are limited so long as outgoing connections are not blocked. The Bridge has a unique ID and the Dispatcher associates any devices or services on that connection with that Bridge. UPnP devices have their own unique IDs and so any device communication can be routed between devices using first the device ID to find the right Bridge, and then via the dedicated connection to that Bridge. Replies travel a reciprocal path using the originating device ID.

Once connected, a Bridge creates a DeviceAdded message for each available UPnP device. The message includes the device’s unique ID, the Bridge’s unique ID, and encapsulates UPnP protocol information, specifically the XML describing the UPnP device and its services (the Device Description and Service Description as specified in [1]). These messages are sent to the Dispatcher, which forwards them to other appropriate Bridges. This same process is used whenever a new UPnP device becomes available at a Bridge.

When a device becomes unavailable, the Bridge creates a DeviceRemoved message containing the device ID and Bridge ID. This is forwarded to the Dispatcher and then to any Bridges that were previously forwarded a DeviceAdded message for the same device. Furthermore, if a Bridge itself becomes unavailable, the Dispatcher can send a single DeviceRemoved message to other Bridges with only the Bridge’s unique ID. Those Bridges can then determine which devices were hosted at that device and remove them.

Rather than simply scaling up UPnP, our approach reduces the number of messages between devices as we do not send messages to the Dispatcher for each SSDP message. We send a single DeviceAdded and DeviceRemoved message to the Dispatcher. In comparison, UPnP mandates that devices send SSDP keep-alive messages across a whole network periodically for the time a device is available. Furthermore, the Dispatcher can be selective about where DeviceAdded messages are forwarded, and then send DeviceRemoved messages only to relevant Bridges.

This approach solves the problem of scaling SSDP, however it introduces a new challenge of deciding what devices to share and where. Clearly, if all devices are shared to all Bridges, each local network ends up with a very large number of devices and this will not scale. In fact, we end up sending even more data than if we had one large network. However, it is reasonable to assume that not all users will need or want access to all available devices. A user may want access to her home devices wherever she goes, and this can be
achieved by remembering her home Bridge, and making devices on that Bridge available locally by remembering the UPnP ID of the device she is using and associating it and the Bridge ID with that user. A user may want access to work devices and this too can be achieved by allowing that user access through either her home Bridge ID, or again the UPnP ID of a device she might use to access those devices. Our architecture enables any of these approaches. In our current implementation we have a small number of devices and networks, and we manually configure access based on UPnP ID and Bridge ID for each user. This is discussed in more detail in the Future Work section below.

C. The Mock Device

When a Bridge receives a message containing details of a remote device, it makes that device available on the local network through the creation of a Mock Device. A Mock Device is a local UPnP device created using the XML describing the remote device, and the XML describing the services the remote device provides. The only change made to the description XML is the IP address and port of the remote device. This is changed to the local IP address of the Bridge and a new port is assigned. Other URIs in the description XML are relative to this IP address and port and so the Bridge can then intercept SSDP, SOAP and GENA messages for the remote device. As far as devices on the local network are concerned, this is a real UPnP device with valid URIs that can be invoked. The modified control point manages all the usual functions of the Mock Device, e.g. regular ‘keep-alive’ broadcasts. When a Bridge receives a DeviceRemoved message for a remote device for which it currently is hosting a Mock Device, the Bridge sends an ssdp:byebye message to other local devices. This informs them that the Mock Device is now unavailable. The Bridge then terminates the Mock Device.

When a UPnP device on a local network subscribes to a state variable of a Mock Device, the modified UPnP CP catches that subscription request, wraps it up and sends it to the Dispatcher as an Inter-Device Request. The Dispatcher forwards the message to the Bridge that hosts the remote device, and the local UPnP CP subscribes to that state variable and records the ID of the bridge and the device that requested the subscription. The remote device will respond with the duration of that subscription as defined in the UPnP Device Architecture [1]. This reply is wrapped up and an Inter-Device Response is sent to the Dispatcher, which forwards it on to the Bridge from which the subscription request originated. The Bridge unwraps the message and sends the response to the UPnP device that originated the request. If there is any change to the state variable, the remote device notifies the UPnP CP, which wraps the notification up in an Inter-Device Request and forwards it to the Dispatcher with a list of Bridge IDs and device IDs of those devices that have subscribed to the state variable. The Dispatcher forwards the message on to each appropriate Bridge, and the Bridges send notifications on to relevant devices.

When a device wishes to invoke a method of a service on a Mock Device it will send a SOAP message, as if to a real UPnP device. The modified UPnP CP hosting the Mock Device will catch that SOAP message, wrap it up as an Inter-Device Request and forward it on to the Dispatcher. The Dispatcher forwards the message on to the Bridge hosting the remote device. The Bridge unwraps the message and sends it on to the remote device. When the device responds, the modified UPnP CP sends the response back to the originating device along a reciprocal path using an Inter-Device Response message. However, the data returned from the remote device assumes the requesting device is on the same local network. It can include local IP addresses that would not be accessible on the remote network. Furthermore, in some cases UPnP devices can use messages to negotiate non-UPnP channels to communicate. The modified Control Point therefore needs to know about the content of messages and what they could mean.

For example if the user of a MediaRenderer:1 service invokes the Browse action on a local Mock Device representing a remote device hosting a ContentDirectory:1 service (e.g. a user of Windows Media Player clicks on a library listed in the Other Libraries section), the UPnP CP in the local Bridge wraps this request and forwards it to the Dispatcher. The Dispatcher identifies the Bridge hosting the remote device and forwards the Inter-Device Request message on. The remote Bridge unwraps the message and invokes the Browse action on the ContentDirectory:1 service with the parameters the MediaRenderer:1 service used originally. The results are wrapped up as an Inter-Device Response message and returned to the Dispatcher, which sends the reply to the originating Bridge. The response will contain a number of parameters one of which, Results, may contain a list of objects each described by an XML fragment as defined in the ContentDirectory:1 Service Template [13]. The ‘res’ or resource elements included in such a fragment identify binary assets such as files and each element contains a URI that identifies the binary asset. If the URI has an IP address that is not available globally, any devices not on the same network as the remote device will not be able to access that binary asset. We therefore alter the XML by rewriting the URI to include the IP address of the Bridge hosting the Mock Device and a unique port for that URI. Should a device attempt to access the URI, we can then intercept this request, send it as an Inter-Device Request message to the Dispatcher, then to the remote device’s Bridge and finally on to the remote device itself.

One advantage of this approach is that we can add extra functionality. For example, if two users on a network served by Bridge A wish to stream a file from the ContentDirectory:1 service, we could simply pull it across the network once and cache the file at Bridge A. Similarly, if Bridge A also has a device hosting a ContentDirectory:1 service, that too could be streamed across the whole network.

V. CONCLUSIONS

The architecture described in this paper enables device discovery and communication between local networks using existing discovery technologies, in this case UPnP. We have implemented this with a single Dispatcher connecting three remote networks, each with a single Bridge and a number of UPnP capable devices, such as a laptop running Windows 7. We have demonstrated browsing the available media at any of the Bridges and streaming files between Bridges on demand.
In choosing to design our architecture we have consciously constrained communication between networks so that it must travel through our Dispatcher. This gives much greater control over which devices are shared to where, and the security of communication between devices, than simply opening ports on a firewall as the Internet Gateway Device specification [5] describes. Furthermore, we have begun to address the problem of SSDP scalability by building into the architecture the ability to limit which devices are shared between networks using a variety of strategies. While these strategies have yet to be evaluated, each addresses the limitations of SSDP by limiting the total number of clients (DR) on any network, thereby minimizing the total bandwidth required as calculated in (1).

This method can be applied to other discovery technologies. While the Bridges need detailed knowledge of UPnP, the Dispatcher knows nothing of it. There is no reason that the architecture could not be extended to other discovery technologies so long as the following conditions hold. (i) The technology has a model similar to UPnP, i.e. devices as containers for services. In practice, most services are hosted on web servers and so we could simply create a device container for the web server in each case. (ii) Each device and service needs a unique ID. (iii) We need to be able to detect when a device becomes available or unavailable. (iv) The devices must be well specified such that we can identify messages that will need to be changed depending on network accessibility. (v) We must be able to insert a ‘man in the middle’ between communicating devices.

VI. FUTURE WORK

We are investigating three main areas to move our current research forward: (i) moving away from a single Dispatcher, (ii) strategies for sharing devices between Bridges, and (iii) protocol agnostic device and service descriptions that enable semantic searching for available devices and services.

A single server, no matter the size, would struggle routing messages between large numbers of Bridges and devices, especially if large amounts of data are streamed between two networks. To address this problem we are investigating the use of multiple Dispatchers. As with Louati et al. [9, 11], the idea of a P2P network of servers is appealing, not least because using a Distributed Hash Table (DHT) algorithm gives us a light-weight way of routing traffic between devices (hashing on the Bridge ID) that scales well. For example, using one such algorithm, Chord [14], in an N-Bridge system in a steady state, each Bridge would maintain information about approximately O(log N) other Bridges, and resolve all lookups via O(log N) messages to other Bridges. Our initial investigations suggest a group of ‘super-peers’ each with a list of available Dispatchers would give Bridges a starting point to contact any such P2P network, and they could then be assigned to a Dispatcher. This would allow us to balance the load across available Dispatchers. If two devices, A and B, want to communicate, we would identify the Dispatchers that are directly connected to the Bridge hosting A and the Bridge hosting B and create a direct connection between the Dispatchers. After the first contact, little routing overhead would be required.

Our architecture also enables a number of strategies for limiting and choosing those devices shared between Bridges. For example, we are investigating a device-centric approach where Bridges make devices available based on those Bridges where devices have previously been recorded. We can vary how the records are chosen (most used, most recently used, etc.) and the life of such records. We are investigating user tools that make these data visible and allow users to turn on and off sharing as they wish.

Finally, we plan to extend the architecture to other discovery technologies and are investigating how we might create more protocol agnostic descriptions of devices and the services they provide. This would give us a common terminology for searching across discovery technologies. UPnP is a very well described set of devices and services, and so may lend itself well to the creation of semantic ontologies that describe those services and devices, and form a basis for such protocol agnostic descriptions.

REFERENCES


