Bluefi
Bringing Bluetooth into the Cloud

Ryan McLelland

Introduction

As more and more everyday electronics become Bluetooth enabled, the utility of the technology is only going to increase. At the moment, Bluetooth radios are limited in range to around 100 meters, and this range is greatly decreased when indoors. The idea behind the Bluefi project was to enable Bluetooth devices to discover and interact with each other when they are outside of the range of their Bluetooth radio. This would be made possible through the creation of 'Bluetooth Access Points', which would be connected to the Internet through either Wifi or an ethernet connection, and which would also have a Bluetooth adapter. A local Bluetooth device would connect to an access point, which would tunnel the Bluetooth communications over the Internet to another access point, which would then forward it to the remote Bluetooth device. Implementing this technology would expand the uses of current Bluetooth protocols, and also enable new protocols to be implemented for Bluetooth devices, such as multicast.

Background

The Bluetooth stack is comprised of multiple protocol layers, all of which are separated from the physical layer by the Host Controller Interface (HCI). Atop this sits the L2CAP protocol, which is an unreliable packet communication layer, and built on top of that is RFCOMM, which is a reliable packet communication layer. One of the most important pieces of the Bluetooth stack is the Service Discovery Protocol (SDP), which is used to advertised exactly what capabilities a certain Bluetooth device has through the use of Bluetooth services. Every Bluetooth service implements one or more Service Classes, which are defined by the Bluetooth SIG and which identify what a certain service's capabilities are. Each Bluetooth devices registers one or more services with its local SDP, which will then transmit the device's list of services to any inquiring Bluetooth devices. If an inquiring device wishes to use one of the services offered by another device, it will then initiate a connection to that device, and specifically to the desired service.

Architecture

The Bluefi project was implemented using the BlueZ Bluetooth stack for Linux, which provides APIs to every layer of the Bluetooth stack from the HCI up to the application layer. The initial implementation of the project was mainly done at the service layer, and it essentially served as a wrapper around SDP. The general procedure for the intended use of this system is as follows. Every 30 seconds, a Bluetooth Access Point performs a general inquiry in its local area and compiles a list of all of the accessible local Bluetooth devices, and also what services are offered by each device, and this information is then transmitted to remote access points. When an access point receives a list of services from another, it registers these services with its local SDP, advertising them as if they were its own, but it first appends the name of the remote Bluetooth device to the name of the service.

When a Bluetooth client wishes to try to communicate with a remote device, it will first initiate a connection with its local access point. Once connected, the SDP on the access point will provide the device with a list of remote services that are being offered by devices at other access points. If the device wishes to utilize one of these services it will then connect to this service on the access point. The access point will then open a connection with the remote access point, and that access point will initiate a Bluetooth connection with the real service on the remote Bluetooth device. Once these connections are established, Bluetooth packets will be sent to the access points, which will wrap them in IP packets and transmit them to the other access point, from which they will be sent on to the other Bluetooth device.

Implementation

The actual implementation of the access point was done using a thread based design. Upon start-up, the main thread of the program immediately spawns another thread, whose purpose is to listen for incoming connections from other access points and then receive their service list. This thread also maintains a list of services that the access point is currently advertising so that if the remote access point stops being able to contact a device, the access point can unregister that device's services. The main thread then starts a timer, and every 30 seconds it performs an inquiry for local Bluetooth devices. For each device it finds, it sends that device's information to the other access points, then retries a list of services for that device and transmits that list to the other access points as well. These are

referred to as Service Sender and Service Receiver threads, respectively.

For each service that a Service Sender thread transmits to other access points, it then spawns a thread that begins listening on a new port for incoming connections for other access points. When a connection is received it means that another access point is trying to use the local service represented by that thread, and it then initiates a Bluetooth connection to the local device that actually offers the service. Once that connection is made, it will transmit Bluetooth packets back and forth between the Bluetooth device and the other access point. This is referred to as a Bluetooth server thread.

Going the other way, for each service that a Service Receiver thread receives from another access point and registers with its local SDP, it will spawn a thread that will create a Bluetooth socket and begin listening for incoming Bluetooth connections from local Bluetooth devices trying to use this service. When a Bluetooth device connects to this service, the thread will then initiate a connection to the other access point (specifically the corresponding Bluetooth server thread's port), and then begin tunneling Bluetooth packets back and forth between the local device and the other access point. This is referred to as a Bluetooth client thread.

Implementation Issues

Upon reaching the point in implementation where the access points could begin to be tested, a problem was discovered with the nature of current commercial Bluetooth devices. The architecture of the access point protocol relies upon the fact that through the BlueZ library one can register any service with the SDP and alter its attributes, however most commercial Bluetooth devices do not actually present the user with a list of services when they are trying to initiate a connection, only a list of devices. This undermines the access point architecture because, that being the case, a user could only ever select to connect to the access point, and they would never know what remote Bluetooth device they were connecting to.

A simple alteration to the Bluetooth clients would make our access points work as intended, they need only display the user a list of candidate services for each device rather than solely the devices, but it is obviously not possible to modify every commercial Bluetooth application to support this need. Using a Bluetooth enabled cell-phone and setting up the environment so that only a single device was situated around the remote access point, a file was successfully

transmitted over Bluetooth and across access points to the remote device, however this was a very controlled test and the user of the cell phone had to select as if they were simply transmitting the file to the local access point. Also, using the 'sdptool' program provided with the BlueZ library, the access point implementation was able to be tested and verified, but beyond that no further testing was accomplished.

Alternative Architecture

When the user interface problem of Bluetooth clients was discovered, a new access point architecture was developed that has the potential to overcome this difficulty and make the access points functional for commercial Bluetooth devices, although this architecture was not actually implemented. The plan that was developed is to create 'virtual' Bluetooth devices on the access point for each remote Bluetooth client that exists around each other access point. Having done this, an HCI socket, which is used for communication between the host controller on the Bluetooth hardware and the application, would be opened in “raw” mode and begin receiving packets. Raw mode indicates that the HCI layer does not do any processing on the packet at all before it forwards it over the socket. With this socket packets would be received from the Bluetooth hardware and forwarded over the internet to the corresponding remote access point, which would then open a socket to the HCI layer of the destination Bluetooth device and again forward the packet. Using the HCI sockets would prevent the access points from having to worry about the service layer at all, because the local Bluetooth device would simply communicate with the SDP of the remote Bluetooth device, rather than the SDP of the access point as in the previous implementation. This would make the access point implementation much less complex.

There are, however, a few outstanding issues with this design as well. The main problem is that current Bluetooth radios do not support a 'promiscuous' mode, meaning that they can only receive packets that are addressed directly to them. With that being the case, all of the virtual Bluetooth devices created on the access point would have to have the same Bluetooth address as the address of the physical Bluetooth adapter on the access point. This presents a problem because if all of the virtual devices have the same address, there is no way for the actual physical adapter to demultiplex the packets that it is receiving from Bluetooth clients and determine which packets are destined for which devices. This problem would not be present if each virtual device could be assigned a

unique Bluetooth address, but with currently available Bluetooth hardware this would not be possible. This could also be overcome if there were another way to determine which virtual device an incoming packet was destined for, but none could be thought of at present.

The other problem involves finding a way to override the standard device inquiry response procedure so that the one physical Bluetooth adapter on the access point could issue multiple inquiry responses, one for each virtual Bluetooth device. This should be possible to do, however the specific location in either the BlueZ library or the device driver where this could be done was not located. Also, it is unclear whether commercial Bluetooth clients would accept multiple inquiry responses from the same Bluetooth address.

Conclusion

Although this project did not end up producing something that is commercially useful, it was still a very interesting and informative venture that could potentially prove useful in the future. With new technologies emerging daily and more and more household items coming with Bluetooth connectivity, it is inevitable that eventually there will be some changes to the Bluetooth protocols. Hopefully at some point in the future Bluefi will be able to be completed and become a useful addition to the Bluetooth world.