An Architectural Framework to deploy Scatternet-based Applications over Bluetooth

Nitin Pabuwal , Navendu Jain and B. N. Jain
Department of Computer Science and Engineering
Indian Institute of Technology, New Delhi 110016, India
Emails: pabuwal_n,navendu,bnj@cse.iitd.ernet.in

***Abstract*— Bluetooth is a promising wireless personal area network technology and is on the verge of being ubiquitously deployed over a wide range of devices. The basic unit of a Bluetooth network is a centralized master-slave topology, namely a piconet, that can be easily extended into a multi-hop ad-hoc network called a scatternet. Scatternets increase Bluetooth's usability multi-folds such that numerous applications may be built over them to unleash the potential of Bluetooth. The main ingredients of a scatternet-based application include a topology formation and a routing algorithm, which themselves are of many types.**

A standard architectural framework shall prove extremely useful to integrate all these applications and algorithms in a seamless, modular and re-usable fashion, hence saving one from re-inventing the wheel most of the time. In this paper, we present one such novel architectural framework that constitutes of highly portable and plug-n-play modules to deploy scatternet-based applications over Bluetooth. These constituent modules may be developed independently and be easily integrated at run-time. The algorithm modules may be built in an application-oriented fashion to deliver better performance to particular type of applications that may have specific requirements and constraints. To the best of our knowledge, this is a first attempt to propose a standard architecture to deploy applications over Bluetooth. We provide details of our proposed set of APIs and describe a multimedia application that we have built in complete accordance with our architecture.

I. INTRODUCTION

In December , an industry consortium known as the Bluetooth SIG standardized Version of Bluetooth [1]: a promising wireless personal area network technology to provide a low power, low cost, globally uniform radio interface between any two devices with Bluetooth support. Today, Bluetooth is on the verge of being ubiquitously deployed over a wide range of devices.

Piconet (Fig. 1(a)) is the basic unit of a Bluetooth network. A piconet is a centralized master-slave topology with atmost eight active members at a time, with every slave within ten meters of the master. The logical extension to a piconet is to join multiple piconets and create a multi-hop ad-hoc network called a scatternet (Fig. 1(b)). Within a scatternet, adjacent piconets are connected by common nodes that act as bridges. Peaceful existence of upto ten piconets in a coverage region is guaranteed by the fact that communication within a piconet is based on frequency-hopping with hops/second, with the hopping sequence being a function of the Bluetooth address of the master and hence varying across separate piconets.

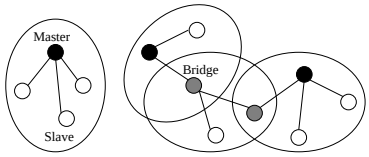

Fig. 1. (a) A Piconet (b) A Scatternet

Scatternets increase Bluetooth's usability multi-folds and open many new arenas to provide excellent value-added services to the users. Numerous applications may be built over them to unleash the potential of Bluetooth. The main ingredients of a scatternet-based application include a topology formation and a routing algorithm, which themselves are of many types as the Bluetooth specification [1] does not define any specific ones. Every implementation of these applications requires an implementation of one of these algorithms hence making one re-invent the wheel most of the time. This calls for a standard architectural framework to integrate all the applications and algorithms in a seamless, modular and re-usable fashion.

To build a highly flexible and robust architectural framework to deploy a wide variety of scatternet-based applications over Bluetooth, we have identified the following requirements:

A. Platform Independent

Any application that wishes to be deployed over Bluetooth should aim to be portable over a wide range of platforms. This platform independence is required on two accounts:

1) Stack Independence: With the coming of Bluetooth, came a plethora of stacks, both open-source and proprietary. A basic Bluetooth stack spans atleast four layers of the OSI network model from the physical to the transport. Most of these layers are directly accessible to the applications running at the top. With each stack having its own API and specifications, porting an application over each of these stacks poses a herculean task. So the idea is to provide a standard API between the applications and the stacks such that an application complying to this standard API becomes independent of the underlying implementation.

2) Device Independence: Bluetooth intends to get deployed on just any wireless device that may ever exist. To achieve

[1]N. Pabuwal is currently associated with Ensim Corporation and N. Jain is a Computer Science Graduate Student at University of Texas at Austin.

device independence, the application should be written in a programming language that is portable across most of these devices. Java comes as a natural choice. Java 2 Platform Micro Edition (J2ME), with its Connected Limited Device Configuration (CLDC) augmented with the Mobile Information Device Profile (MIDP) [2], is the most widely deployed version of Java and spans almost every wireless device that possesses the capability to connect to a network.

So to meet both the above requirements of *stack independence* and *device independence*, we need a standard API between the applications and the stack and that too Java-based. In March , the Java Community Process (JCP) finalized Version of the Java Specification Request 82 (JSR-82) [3], the first standard Java API for Bluetooth. The JSR-82 API has been designed to operate on any device with a J2ME CLDC Reference Implementation (RI) and with a minimum K of total memory for the Java 2 platform. Though our architecture is not dependent on the Bluetooth API being used (as long as it is Java-based), presently the functionality has been defined keeping JSR-82 in mind but should hold good for any other API that provides atleast as much functionality as JSR-82.

B. Plug-n-Play

Scatternet formation and routing algorithms are responsible for defining the characteristics of a scatternet. A scatternet can deliver better performance to particular type of applications if these algorithms are application-oriented. By simply switching among algorithms the users can easily decide what works best for them. So to allow selection of algorithms at run-time, we have integrated them into the architecture in a plug-n-play fashion. In this way, the user may also download and run algorithm modules that may be developed independently but in accordance with the architecture.

This paper is organized as follows. In section II we describe our proposed architecture, and then provide the specifications of the APIs in section III. We also present a state-of-the-art multimedia application based on our proposed architecture as an illustration in section IV and conclude the paper in V.

II. Architecture

Fig. 2 shows the components of our proposed architecture. Availability of an underlying Java-based Bluetooth API such as JSR-82 is the lowest-level requirement for applications complying to our architecture to be deployed on a device. Devices running similar scatternet formation and routing algorithms connect to form a network over which the application runs. The various modules–User Interface and Application, Scatternet Formation Algorithm, and Router–interact through a fixed set of proposed APIs hence making each of them completely pluggable and re-usable. Details of the APIs have been provided in the next section.

A. User Interface and Application

The User Interface provides the interaction with the end-user and encapsulates the Application which is an implementation-instance of the protocol being defined by the deployed application to interpret application-level data. The architecture

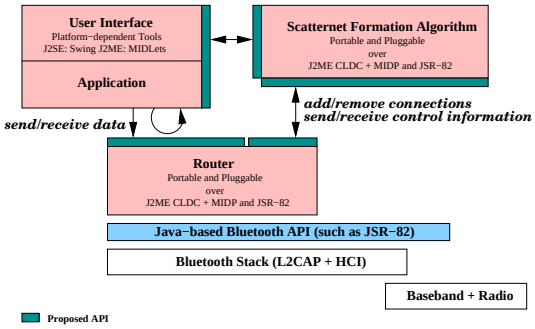

Fig. 2. Architecture

does not require this module to be portable over different devices and hence it may be built using any of the tools provided by the target device, such as MIDLets over J2ME and Swing over J2SE (Java 2 Platform Standard Edition). In this way more sophisticated Interfaces may be provided to users working with more resourceful devices. Only the encapsulated Applications are expected to comply to the same protocol. The Interface may provide access to only those features of the application that can be supported by the target device. The details are implementation-dependent. It may also provide the user with an option to choose the scatternet formation and routing algorithms.

The encapsulated Application shall normally interact only with the Scatternet Formation Algorithm and the Router modules, but may access the Bluetooth API directly if required.

B. Scatternet Formation Algorithm

The Scatternet Formation Algorithm is responsible for constructing the scatternet. Each algorithm has a specific protocol that defines how the topology should be constructed and maintained. Maintenance is required in order to account for mobility as devices continuously move in and out of range.

The Algorithm module defines a Maximum Transmission Unit (MTU) size for each packet sent and received over the scatternet. The Application and the Router are expected to get the MTU-size from it.

C. Router

The Router defines how data is to be routed over the scatternet. The Router module takes care of constructing and maintaining the routes as well as sending and receiving data over them. It provides abstraction of two types of packets: *Application Data* and *Control Information*, being sent and received by the Application and the Scatternet Formation Algorithm respectively. A third type, the *Route Information*, is used by the Router for internal packets to perform its own task of constructing and maintaining the routes.

The Router needs to fill internal header information within every packet and hence every packet sent and received through it should accommodate extra space for the header. The header size should be obtained from the Router itself.

Both, the Algorithm and the Router modules, are expected to be completely portable and should be built using only J2ME CLDC and MIDP APIs as that is the most widely deployed version of Java. Also, the modules should interact with the Bluetooth stack only through the intermediate API.

Important to note is that since two devices may connect only when they are running the same scatternet formation and routing algorithms, for resolution purpose, different algorithms should identify themselves as separate services. Before connecting to a device, the local device is first expected to initiate a service search on the remote device to check if it provides the same services that represent the algorithms being used.

D. Interactions

The component modules of the architecture interact only through a proposed set of APIs. Each module implements specific Java interfaces that are exposed to other modules to provide abstraction.

1) Application and the Scatternet Formation Algorithm: The Scatternet Formation Algorithm implements *Scatternet* interface through which the Application may ask the Algorithm to join, cancel joining, or leave a scatternet. The Application may ask for information about its local device's present status in the scatternet, depending on which immediate connected devices might be known. It may also get the list of all devices in the scatternet or its piconet, provided the algorithm's implementation supports it. The Application also obtains a reference to a Router module through the Algorithm that it uses to send and receive data.

The Application implements *ScatternetListener* interface through which the Algorithm conveys information of addition and removal of devices from the network, both at the scatternet and the local piconet level.

2) Application and the Router: The Router implements *Router* interface through which the Application sends and receives packets of the *Application Data* type. This is a one-way interaction. The Router never initiates a communication and hence does not require the Application to implement a specific interface.

3) Scatternet Formation Algorithm and the Router: The Router extends *RouterControl* abstract class which in turn implements *Router* interface. Hence the Algorithm may send and receive packets of the *Control Information* type using the *Router* interface. Additionally, the Algorithm conveys information of addition and removal of immediate connections of the local device to the Router that it may use to update its internal state and routes. The Algorithm may obtain the list of all devices in the scatternet through the Router (if supported by it) in order to pass it to the Application. It may ask the Router to shutdown and return to initial state whenever it is asked by the Application to leave the scatternet. The Algorithm also asks the Router for its service identifier while initiating a service search on the remote device.

The Algorithm implements *RouterListener* interface. Loss of existing connections is detected by the Router which it communicates to the Algorithm. If supported, the Router also conveys information of addition and removal of devices at the scatternet level to the Algorithm which it further passes to the Application.

No specification has been proposed for the interaction of the modules with the Bluetooth API and should only depend on the API's specifications.

III. API Specification

Here we provide the details of our proposed APIs for interaction between different modules of the architecture. All classes are contained in package *com.iitd.bluetooth.scat*. All members listed are *public*. Description may be found in [4].

A. Classes

1) Interfaces:

Router
RouterListener
Scatternet
ScatternetListener

2) Classes:

AppData
RouterControl

3) Exceptions:

DeviceStatusException

B. com.iitd.bluetooth.scat.*Router*

1) Fields:

static byte ALGO_PKT
static byte APP_PKT
static byte ROUTE_PKT

2) Methods:

int getHeaderSize()
boolean ready(byte type)
int receive(AppData data, byte type)
void send(AppData data, byte type)

C. com.iitd.bluetooth.scat.*RouterListener*

1) Methods:

byte connectionRemoved (javax.bluetooth.L2CAPConnection conn, java.lang.String bdaddr)
void deviceAdded(java.lang.String bdaddr)
void deviceRemoved(java.lang.String bdaddr)
int getMTU()
byte getStatus()

D. com.iitd.bluetooth.scat.*Scatternet*

1) Fields:

static byte BRIDGE
static byte MASTER
static byte SLAVE
static byte UNKNOWN

2) Methods:

boolean cancelJoinScatternet (ScatternetListener listener)
java.util.Enumeration getBridges()
java.lang.String getFriendlyName (java.lang.String bdaddr)
java.lang.String getMaster()
java.util.Enumeration getMasters()
int getMTU()
java.util.Enumeration getPiconetDevices()
Router getRouter()
java.util.Enumeration getScatternetDevices()
java.util.Enumeration getSlaves()
byte getStatus()
boolean joinScatternet (ScatternetListener listener, java.lang.String routerName)
boolean leaveScatternet (ScatternetListener listener)

E. com.iitd.bluetooth.scat.ScatternetListener

1) Fields:

static int PICO_ALT
static int SCAT_ALT

2) Methods:

void deviceAdded(java.lang.String bdaddr, int type, byte status)
void deviceRemoved(java.lang.String bdaddr, int type, byte status)

F. com.iitd.bluetooth.scat.AppData

1) Fields:

static byte BCAST
static byte MCAST
static byte RECVD
static byte UCAST

2) Constructors:

AppData(byte type)
AppData(byte type, java.lang.Object bdaddr, byte[] data, int size)

3) Methods:

void finalize()
java.lang.Object getBluetoothAddress()
byte[] getData()
int getSize()
byte getType()
void setBluetoothAddress (java.lang.Object bdaddr)
void setData(byte[] data)
void setSize(int size)

G. com.iitd.bluetooth.scat.RouterControl

1) Constructors:

RouterControl(RouterListener listener)

2) Methods:

abstract void addConnection (javax.bluetooth.L2CAPConnection conn, java.lang.String bdaddr, byte status)
void addRouterListener (RouterListener listener)
abstract java.util.Enumeration getScatternetDevices()
abstract javax.bluetooth.UUID getUUID()
abstract void removeConnection (javax.bluetooth.L2CAPConnection conn, java.lang.String bdaddr, byte status)
void removeRouterListener (RouterListener listener)
abstract void shutdown()

H. com.iitd.bluetooth.scat.DeviceStatusException

1) Constructors:

DeviceStatusException()
DeviceStatusException(java.lang.String msg)

IV. Multimedia Application

We shall now describe the multimedia application that we have built in complete accordance with our proposed architecture. The application may be used as a reference to build further applications.

The multimedia application provides features such as *Chat*, *File Transfer, Image Transfer,* and *Video Streaming*. All these features are push services and are initiated by the user who wants to send data to other users. Data may be sent over a unicast, multicast, or a broadcast session. Destinations are identified by their Bluetooth device address. All the features may not be exposed to the user in every implementation of the application as that depends on the tools being provided by the target device. If some data for an unexposed feature is received, it is simply discarded.

A. Protocol

The protocol for the application defines how communication among instances of the application deployed over separate devices takes place.

Fig. 3 shows the structure of each packet transmitted by the application over the scatternet:

Router Header (**Header**): *variable bytes* - accommodates space for the Router's header. Size may be known through the Router being used.

Code (**C**): *1 byte* - identifies the feature the packet belongs to: *Chat, File Transfer, Image Transfer,* or *Video Streaming*. Takes the value *0, 1, 2,* or *3*, respectively.

Status (**S**): *1 byte*- identifies the position of the packet in a particular transmission i.e. whether its the first,

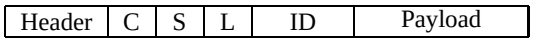

Fig. 3. Packet Structure

intermediate, or the last packet. Takes the value *0x01*, *0x02*, or *0x04*, respectively.

Identifier Length (**L**): *1 byte* - gives the length of **ID**.

Identifier (**ID**): **L** *bytes* - identifies the transmission the packet is a part of.

Payload Data (**Payload**): *variable bytes* - is the payload.

Important to Note:

Total length of the packet never exceeds the MTU-size defined by the Scatternet Formation Algorithm.

No sequence number is sent out with the packets as reordering within the scatternet is assumed to be absent.

Fields **S**, **L** and **ID** are not present if the **C** field is *0* i.e. if the feature is *Chat*. This is because a chat message is restricted to one packet only and hence no identification for the transmission is required. And so if the **C** field is *0*, the **Payload** starts immediately after it.

Every time the application needs to make a transmission, it breaks the data into MTU-sized packets and sets their **ID** to the name of the file being transferred. At the receiving end, the data is collected in a file named *sink*_**ID**_*source[.ext]*, where *sink* and *source* are Bluetooth addresses of the receiver and the sender respectively; and *ext*, if present, is the extension part of the file-name being used as **ID**. In this way uniqueness is guaranteed and two simultaneous receptions do not mix.

B. Modules

We have built all the modules present in the architecture that work and interact exactly in the way proposed. Few of their particulars include:

1) User Interface: The User Interface has been built over J2SE using Swing components and shall not be portable to small devices that provide only J2ME. With this interface, the user may join and leave a scatternet; send, receive and view data when in connected state; and check logs. The user may select a list of destinations for its packets from a list of known devices that keeps updating through notifications received via the Scatternet Formation Algorithm module. While trying to join a scatternet, the user may choose the Algorithm and Router modules to be used to define the network characteristics. Various snapshots of the interface may be found in [4].

2) Scatternet Formation Algorithm: We have built two scatternet formation algorithms of our own as separate modules. When started, these modules register their service identifiers with the stack and try to discover neighboring devices that provide similar services and then connect as per their respective protocols. Once the local device is a member of a scatternet, the algorithms continuously try to maintain the scatternet by connecting to more devices and accounting for mobility.

3) Router: We have implemented some of the existing wireless ad-hoc routing protocols, including Destination Sequenced Distance Vector (DSDV), Clusterhead Gateway Switch Routing (CGSR), and Cluster Based Routing Protocol (CBRP), as separate modules. When started, these modules register their service identifiers with the stack and then construct and maintain the routes as per their respective protocols. DSDV and CGSR modules also provide with a list of all devices in the scatternet as they need to maintain it within their routing tables.

C. Simulations

We had ported Motorola's JSR-82 RI [5] over BlueZ [6], an open source Bluetooth stack for Linux, to conduct experiments over BlueBird hardware modules [7] to be obtained through Inventel; but because of unavailability of the BlueBird modules, we conducted the experiments over Impronto Simulator [8] developed by Rococo. This Java-based simulator provided us with JSR-82 and J2ME APIs and our code could run over it without any changes and its correctness could be verified.

Our main objective had been to achieve good perceptual performance and it was met. Theoretically, performance of the multimedia application depends mostly on the application-oriented topology formation and routing algorithms.

V. Conclusion

In this paper we have presented a novel architectural framework to deploy scatternet-based applications over Bluetooth personal area networks. Applications complying to our architecture shall be able to integrate seamlessly with pluggable Scatternet Formation Algorithm and Router modules to have network characteristics best-suited to their requirements. These modules would be completely portable over devices providing a Java-based Bluetooth API such as JSR-82. Only the User Interface module is intended to be built separately for different target devices as they should be able to use more sophisticated tools provided by more resourceful devices. We have also presented a state-of-the-art multimedia application that we have built in complete accordance with our architecture. A variety of scatternet-based applications and portable application-oriented algorithm modules may now be built and integrated as per our architecture to provide excellent value-added services to the users of Bluetooth.

References

[1] *Specifications of the Bluetooth System v 1.1b*, Bluetooth SIG Std., 2001. [Online]. Available: http://www.bluetooth.com

[2] *J2ME CLDC and MIDP*, Sun Microsystems Std. [Online]. Available: http://java.sun.com/products/cldc/

[3] *JSR-82 Specification v 1*, Java Community Process (JCP) Std., 2002. [Online]. Available: http://jcp.org/jsr/detail/82.jsp

[4] N. Pabuwal, "Scatternet-based Multimedia Applications over Bluetooth Personal Area Networks," Master of Technology Thesis, Computer Science and Engineering, Indian Institute of Technology, New Delhi, India, May 2002. [Online]. Available: http://www.cse.iitd.ernet.in/ csd97403/pubs/pabs-thesis.ps.gz

[5] JSR-82 Reference Implementation. Motorola, Inc. [Online]. Available: http://e-www.motorola.com/java/

[6] M. Krasnyansky. Bluez, the official Bluetooth stack for Linux. [Online]. Available: http://bluez.sourceforge.net

[7] BlueBird Hardware Modules. Inventel, Inc. [Online]. Available: http://www.inventel.com/blue_modules.html

[8] Impronto Simulator 1.1. Rococo Software Ltd. [Online]. Available: http://www.rococosoft.com/products/index.html