Bluetooth Introduction and Detailed JSR 82 Explanation

Murali Annavaram & Bhaskar Krishnamachari
Ming Hsieh Department of Electrical Engineering
USC

Basics

- Named after Hagar Blatand (Bluetooth) 10th century King of Denmark
- Industry standard for wireless personal area networks
- Bluetooth SIG/IEEE 802.15.1
- First developed by Ericsson ~1994, SIG formed in 1998. Versions 1 - 1.1 - 1.2 - 2.0 - 2.1

Bluetooth Communication

- Frequency-hopping, Time division duplex system
- 79 1MHz channels in 2.4GHz ISM band, 1600 times a second
- Output power - Class 1: 100mW, Class 2: 2.5mW, Class 3: 1mW

Network Topology

- Piconet: Master-slave configuration
  - One master, up to seven active slaves, 255 parked
  - Master determines the FH sequence and clock

Network Stack

Link Modes, Data Rates
- Synchronous Connection Oriented and Asynchronous Connection-less modes

<table>
<thead>
<tr>
<th>Packet type</th>
<th>FEC</th>
<th>Symmetric max rate (kb/s)</th>
<th>Asymmetric max rate (kb/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM1</td>
<td>2/3</td>
<td>168.8</td>
<td>168.8</td>
</tr>
<tr>
<td>DH1</td>
<td>no</td>
<td>172.8</td>
<td>172.8</td>
</tr>
<tr>
<td>DM3</td>
<td>2/3</td>
<td>256.1</td>
<td>387.2</td>
</tr>
<tr>
<td>DH3</td>
<td>no</td>
<td>290.4</td>
<td>595.0</td>
</tr>
<tr>
<td>DM6</td>
<td>2/3</td>
<td>386.7</td>
<td>477.8</td>
</tr>
<tr>
<td>DH5</td>
<td>no</td>
<td>453.9</td>
<td>723.2</td>
</tr>
<tr>
<td>AUX1</td>
<td>no</td>
<td>185.6</td>
<td>185.6</td>
</tr>
</tbody>
</table>

Application profiles
- Standardizes different communication patterns for specific applications; e.g.,
  - Printing profile
  - Cordless telephony profile
  - File transfer profile
  - Service Discovery profile

Service Discovery Protocol (SDP)
- A client server protocol
- Defines how services are represented
- How to access service discovery database

Bluetooth in EE579 Context
- Bluetooth is a wireless communication standard
  - Primarily designed for low power consumption, with a short range
    - Class 1: 100 mW ~100 meters
    - Class 2: 2.5 mW ~10 meters (Used in N95)
    - Class 3: 1 mW ~1 meter
- Create a personal-area-network (PAN) with Bluetooth devices over a short range
- All devices in the PAN can communicate and share information instantly
- N95 supports V2.0 + EDR ~ 3.0 Mbits/sec
- More info: www.bluetooth.org

Bluetooth vs Infrared
- N95 also supports Infrared for wireless communication
  - Needs line-of-sight
  - One-to-one communication
  - More secure
- No alignment needed
- Multiple devices can communicate simultaneously
- Bluesnarfing & BlueBugging may compromise information
- Bluetooth is not inherently insecure
- Need to use strong keys for security
Bluetooth Naming

- Bluetooth devices use a 48 bit unique address (similar to MAC address).
- Typically use a name, instead of 48 bit address.
  - Naming is just a simple indirection, different devices can have the same name.
  - During discovery process we will use the name to connect.

Bluetooth Transport Protocols

1. RFCOMM: RFCOMM essentially allows transferring bits between two Bluetooth connected device just as if they are connected using a serial port.
2. OBEX: OBEX is a transfer protocol that defines data objects and a communication protocol two devices can use to exchange those objects.

Bluetooth Connections Steps

1. A Bluetooth service provide device transmit the following sets of information on demand:
   - Device name, class, services
   - Service provider can decide whether or not to make the service discoverable.
2. Requestor devices performs a service discovery to find other devices and their services.
3. Use of the service from the provider may require pairing.
   - Pairing is the process of establishing trust, usually allowing the two parties to share a key.
4. Trusted parties can then communicate.

Bluetooth Support in Java: JSR 82

- JSR 82 exposes the Bluetooth software stack to developers working on the Java platform.
  - Service Discovery Protocol (SDP), the Serial Port Profile RFCOMM for serial emulation, Object Exchange APIs.
- We can break down the core Java Bluetooth APIs, found in javax.bluetooth, into four categories: initialization, discovery, device management, and communication.

Bluetooth Usage Model

- Initialization: All Bluetooth-enabled applications must first initialize the Bluetooth stack.
- Client: A client consumes remote services. It first discovers any nearby devices, then for each discovered device it searches for services of interest.
- Server: A server makes services available to clients. It registers them in the Service Discovery Database (SDDB), in effect advertising them. It then waits for incoming connections, accepts them as they come in, and serves the clients that make them. Finally, when the service is no longer needed the application removes it from the SDDB.

Discovery Process: Device Discovery

- DiscoveryAgent class discovers both devices and services.
- DiscoveryListener is implemented by clients to get notifications from DiscoveryAgent.
- Client initiate device discovery using startInquiry().
- DiscoveryAgent invokes the callback methods deviceDiscovered() and inquiryCompleted().

\[ \text{DiscoveryAgent} \rightarrow \text{startInquiry()} \rightarrow \text{deviceDiscovered()} \rightarrow \text{inquiryCompleted()} \rightarrow \text{DiscoveryListener} \]
Discovery Process: Service Discovery

- Each device can offer multiple services; hence DiscoverAgent also provides API for service discovery
- `selectService()` starts serviceDiscovery
- `serviceDiscovered()`, `serviceSearchCompleted()` are the callbacks

Discover Services with UUID

1. UUID: Universally Unique Identifier
   - Service provider calls `uuidgen()` to create the UUID
   - Client use UUID when searching for service
2. Service Discovery Database & Service Discovery Protocol
   - Client can use SDP to call SDP in a server which in turn accesses SDB to discover services
   - Server creates a service record and inserts into SDB
   - Clients querying the remote SDD for available services will find the service record

Device Management

- `LocalDevice()` class is called by the client application to find information about its own Bluetooth services
  - `setDiscoverable()`
  - `updateRecord()`
  - `getBluetoothAddress()`
  - `getProperty()`
- `RemoteDevice()` class is called by the client to get information about the service provider
  - `getBluetoothAddress()`
  - `authenticate()`
  - `authorize()`
  - `isEncrypted()`
  - `isTrusted()`

Communication API

- JSR 82 supports a low-level data transfer (Logic Link Control & Adaptation) and a stream communication interface (RFCOMM)
- L2CAP interface not commonly used
  - Need significant effort from user to handle the packets, fragmentation etc.
- Stream communication API is used extensively in MIDP development
  - `btspp://hostname[CN | UUID];parameters`
    - `hostname` is either localhost, or the Bluetooth address
    - `CN` is the port number to communicate
    - `UUID` is service identifier
    - `parameters` authenticate, authorize, and encrypt information
    - Eg: `btspp://hostname;authenticate=true;authorize=true;encrypt=true`

Initialization Code Snippet

- Both the server and client initialize the Bluetooth stack at the beginning of using JSR 82 API
- Server set the discovery mode to: GIAC (always visible), LIAC (visible for a short time), NOT_DISCOVERABLE (never visible)
- Client retrieves the discovery agent

Service Connection Code Snippet: Server

- Server Code for Setting up Bluetooth Services:
  - `String stService = “Chat”;
  - private static final String stUUID = “??”://Generated using uuidgen
  - private UUID btUUID = new UUID(stUUID, false);
  - Define the server connection URL
  - `String connURL = “btspp://localhost:*|btUUID.toString()”;//name=stService;`  
  - Create a server connection (a notifier)
  - `StreamConnectionNotifier scn = (StreamConnectionNotifier) Connector.open(connURL);`
  - Wait for client requests
  - `StreamConnection scn = scn.acceptAndOpen();`
  - Read data being sent from the remote device
  - `String s = datIn.readUTF();`
**Service Connection Code Snippet : Client**

- Client first does the discovery process and then uses the discoveredServices class to connect to the service
  - // First get pointer to service record
  - ServiceRecord sr = (ServiceRecord)discoveredServices.elementAt(i);
  - // Use SR to get URL
  - String connURL = sr.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
  - // Open connection
  - StreamConnection sc = (StreamConnection)Connector.open(connURL);

**Server Discovery : Client**

- Client implements DiscoveryListener to look for nearby devices
  - public class BtClient implements DiscoveryListener {
    - public void deviceDiscovered(..) { // Keep track of discovered remote devices
    - public void inquiryCompleted(int param) { // Trigger service search }
    - public void servicesDiscovered() { // Keep track of discovered services
      - public void serviceSearchCompleted() { // dispatch thread to handle services
    - try { look4Service = discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
    - } catch(BluetoothStateException bse) { // Handle error

**Things to Remember:**
- Discovery is expensive; so cache discoveries locally
  - RemoteDevice[] localCache = discoveryAgent.retrieveDevices(DiscoveryAgent.CACHED);

**Service Discovery: Client**

- Once a device is discovered you can then call searchServices() API
  - RemoteDevice rd = discovered device;
  - discoveryAgent.searchServices(attrs, uuids, rd, this);

- Then client calls the connection API and open dataoutput stream and writes to it.
  - StreamConnection streamConnection = (StreamConnection)Connector.open(connectionURL);
  - DataOutputStream dataout = streamConnection.openDataOutputStream();
  - dataout.writeUTF(messageOut);