Bluetooth Stack Improvement

Project Area: Network

SUSHIL KUMAR VERMA

OPEN SOURCE CONTRIBUTOR AT HAIKU INC.(MARCH-JUNE2019)

COMPUTER SCIENCE AND ENGINEERING (CSE) STUDENT

BIRLA INSTITUTE OF TECHNOLOGY, MESRA

INDIA

UNDER THE SUPERVISION AND GUIDANCE OF

OLIVER RUIZ DORANTES

BLUETOOTH TEAM LEADER AT HAIKU INC.

MENTOR AT HAIKU INC.

About Me

Full Name: Sushil Kumar Verma
Email: shushiill.verma@gmail.com
Department: Computer Science and Engineering
University: Birla Institute of Technology, Mesra, India
Country: India
Github: https://github.com/shushill
Website: https://shushill.github.io

Project Proposal

Title: Bluetooth Stack Improvement Using Modern C++.

Improving Bluetooth Stack by implementing Protocols, Profile, etc.

Description: Haiku's Bluetooth stack implements a basic subset of general Bluetooth functionality. This functionality needs to be completed and Bluetooth 2.X and later possibilities explored. This task involves investigating the current state of the Bluetooth code, improving the existing code on newer devices (pairing, etc), and improving the stack to make it more useful by implementing incomplete Logical Link Control and Adaptation Protocol(L2CAP) and some Bluetooth stack protocols on top layers of existing layer like Radio Frequency Communication(RFCOMM) and Service Discovery Protocol(SDP).

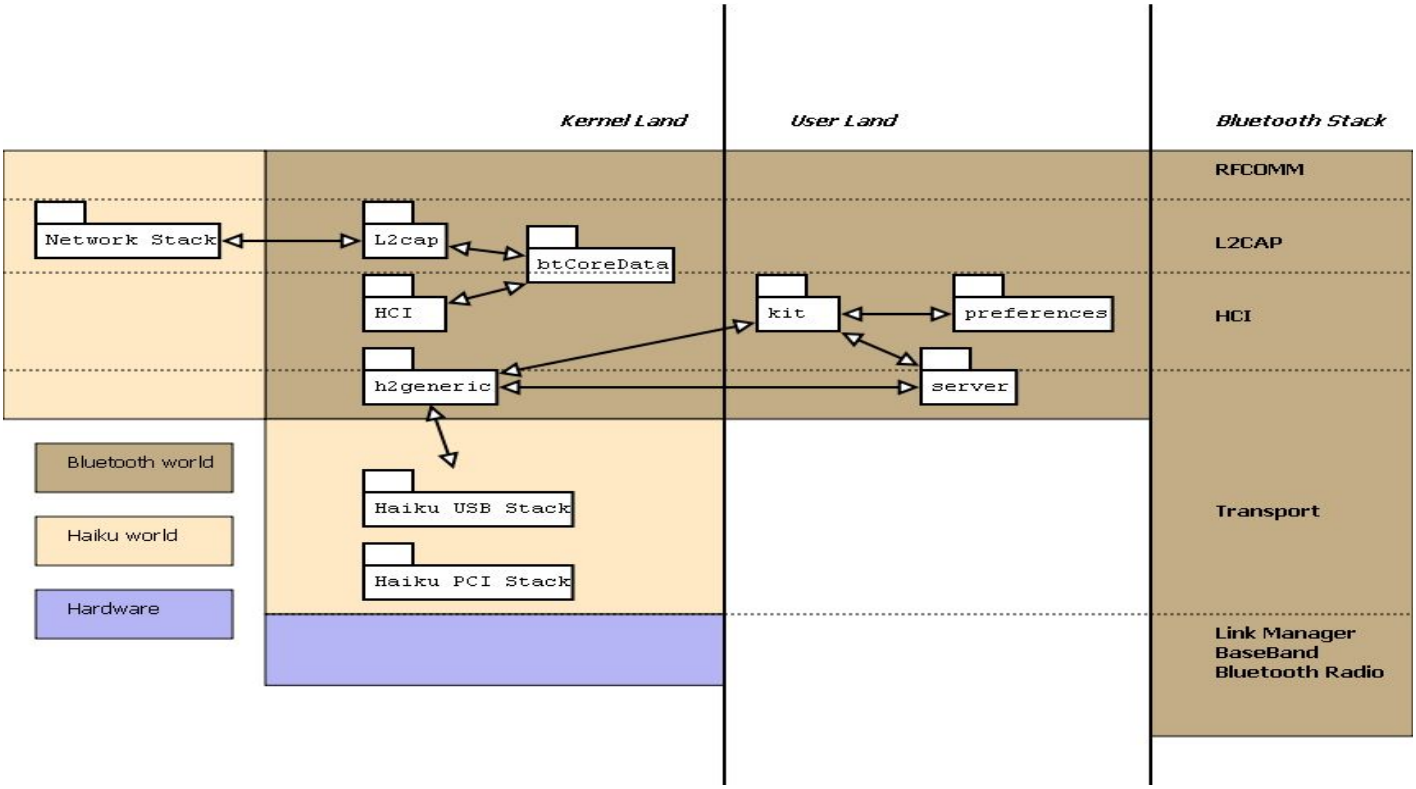
Goals: The Current Bluetooth Stack of Haiku OS lacks some of the basic functionality and protocols. To make it a better Bluetooth stack some functionalities and two protocols would be added.

They are listed below:

- Complete the functionalities of Logical Link Control and Adaptation Protocol
- Implementation of Bluetooth protocol stack on top of existing one:
    a. Radio Frequency Communication (RFCOMM)
    b. Service Discovery Protocol (SDP)

Overview of Current Bluetooth Protocol Stack of Haiku (Before Improvement)

Bluetooth is a wireless technology for creating personal networks operating in 2.4 GHz unlicensed band, with a range of 10 meters. Networks are usually formed ad-hoc from portable devices such as cellular phones, handhelds and laptops. Unlike the other popular wireless technology, Wifi, Bluetooth offers higher level services profile, e.g., FTP-like file servers, file pushing, serial line emulation, etc



Haiku Bluetooth Diagram and Its Specification

The diagram given above provides an overview of the layered Bluetooth architecture for Haiku. It describes the layers of this architecture and how they relate to each other. Each layer is a collection of components that include protocol implementations, APIs, applications, and services.

Here is a piece of table of documentation and description of each layer includes its contents, purpose, and location in relation to the Bluetooth Stack.

Component	Description
L2CAP under network/protocols/stack	Provides socket interface to have l2cap channels. L2CAP offers connection oriented and connectionless sockets. But bluetooth stack as this point has no interchangeability with TCP/IP, A Higher level Bluetooth profile must be implemented
HCI under src/add-ons/kernel/bluetooth	Here we have 2 modules, one for handling global bluetooth data structures such as connection handles and L2cap channels, and frames
H2 generic under src/add-ons/kernel/drivers/bluetooth	The USB driver implementing the H2 transport
Bluetooth kit under src/kit/bluetooth	C++ implementation based on JSR82
Bluetooth Server under src/servers/bluetooth	Basically handling opened devices(local connected physically in our system) and forwarding kit calls to them
Bluetooth Preferences under src/preferences/bluetooth	Configuration using the kit
Bluetooth Controller (Hardware)	A Bluetooth device that implements the lowest levels of the Bluetooth architecture

Drawback of Current Haiku Bluetooth Stack (Missing Features and Protocols)

The current bluetooth is incomplete. From the diagram of Haiku Bluetooth Stack, we get the information that the upper layer which is L2CAP itself suggest that some higher level bluetooth protocol and profile must be implemented over it. To provide serial communication like RS-232, the higher level profile like **Radio Frequency Communication (RFCOMM)** must be implemented. This protocol provides emulations of serial ports over L2CAP protocol. RFCOMM protocol provides roughly the same service and reliability guarantees as TCP. Usually it has multiple ports upto 60 simultaneous connections between two Bluetooth device. To describe how devices must use its protocol to implement a particular task a **Serial**

Port Profile for RFCOMM must be implemented over RFCOMM layer for virtual serial port connections.

One of the most important members of the Bluetooth Protocol Stack is **Service Discovery Protocol (SDP).** This protocol must also be implemented in Haiku Bluetooth Stack as it provides a means for applications to discover which services are provided by or available through a bluetooth device. It also allows applications to determine the characteristics of those available services. It is bound to the L2CAP protocol.

To implement RFCOMM, Serial Port Profile for RFCOMM and SDP Protocols in this Haiku Bluetooth Stack, **L2CAP layer** must be finished i.e., should have complete implementation and provide to link for its upper layer like RFCOMM and SDP. It also lacks some of the basic functionalities which should be taken care of.

Implementation Details(Deliverables)

These implementations would take place in three Phases to improve the Haiku Bluetooth Protocol Stack. The table describes below the implementation details. It matches the corresponding changes which will take place in the Phase number.

Phase Number	Implementation of Protocols or Modification
Phase 1	To improve the L2CAP layer, improve its basic functionalities and missing features
Phase 2	To implement RFCOMM protocol and Serial Port Profile for RFCOMM connection
Phase 3	Implementation of SDP protocol

The Detailed description of each of the Phase Number has been given below.

The terms and technologies used in the Phases has been explained after the completion of Phases writing.

Phase 1:

- Adding Upper layer Protocol Interface module in header file of l2CAP in headers/private/bluetooth/l2cap.h

➔ L2CA_ node messages

1. L2CA node (here node, which refers to the type of structure which contains data information as well as control information of the L2CA or L2CAP layer)
2. Define Service Primitive and Parameters for L2CA_
    a. Function definition for Connection→ Setup, Configure, Close, Disconnect
    b. Define structures for Data→ Read, Write, Packet Header
    c. Define function for Information→ ping(a echo request message from L2CA layer corresponding to echo response message from L2CAP layer), Get info, Request a call-back at the occurrence of an event
    d. Connection-less traffic→ Enable, Disable

➔ L2CAP node messages

1. L2CAP node structure
2. Define L2CAP descriptor for Connection, Channel and Command
3. L2CAP connection states→ open, wait, close
4. L2CAP channel states→ open, wait, response, close, configure, disconnect
5. Node flag for SDP and RFCOMM
6. Get Node flag Info
    a. L2CAP → User
    b. User → L2CAP
    c. Connection Flag
7. Channel List
    a. State, Channel ID, MTU(incoming/outgoing), PSM value, Bluetooth address(remote/local)

- Adding Upper layer Protocol Interface methods in configuration file src/add-ons/kernel/network/protocols/l2cap/l2cap_upper.cpp

1. Send hook information to upper layer from L2CAP layer
2. ACL Connections
    a. New Connection descriptor for remote one and link common descriptor to L2CAP node
    b. Add/Remove connection descriptor
    c. Set/Unset auto disconnect timeout
    d. Get remote connection by address and handle
3. L2CAP Channels
    a. Allocate new L2CAP channel descriptor with PSM to link channel to L2CAP node
    b. Free Channel descriptor
4. L2CAP Command Descriptor
    a. Create New Command Descriptor
    b. Set/Unset L2CAP command timeout

5. Other Stuff and timeout
    a. Default flow settings
    b. Get next free available channel ID and command identical

- Adding a missing feature in src/add-ons/kernel/network/protocols/l2cap/l2cap_signal.cpp
    ➔ Designing of process control messages or routines from upper layer to L2CAP layer

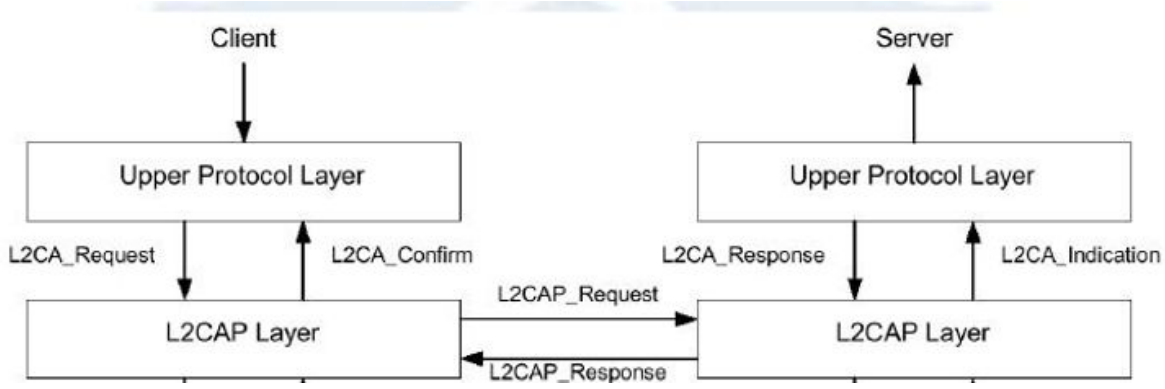


Diagram to demonstrate the Phase 1 work

Phase 2:

This phase is divided two type :: RFCOMM protocol and Serial Port Profile for RFCOMM. First we will implement RFCOMM protocol on top of L2CAP layer bluetooth stack. Then we will implement Serial Port Profile layer on top of RFCOMM layer.

A. Implementation of RFCOMM protocol

❏ Creating RFCOMM directory in src/add-ons/kernel/protocols/
❏ Creating and adding header file in src/add-ons/kernel/protocols/RFCOMM/rfcomm.h
❏ Creating and adding configuration file(source code) in src/add-ons/kernel/protocols/RFCOMM/rfcomm.cpp

- Description of Header file of src/add-ons/kernel/protocols/RFCOMM/rfcomm.h

1. Defining Frame header and type
2. Defining Control Command header and control signal
    a. Remote Port Negotiation (RPN)
    b. Remote Line Status command (RLS)
    c. DLC parameter Negotiation Command (PN)
    d. Modern Status Command (MSC)

3. Defining structure for Data Link Connection Identifier containing channel, address, signas, sessions, control flow
4. Defining functions for RFCOMM sockets and sessions

- Description of configuration file(source code) of src/add-ons/kernel/protocols/RFCOMM/rfcomm.cpp

1. Declaring global and local prototypes for RFCOMM task, session list and socket list
2. Declaring RFCOMM connection function → setup, configure, disconnect, close, timeout
3. Defining a method to establish an L2CAP channel to the peer RFCOMM entity using L2CAP service primitives
4. Defining Socket Interface functions like
    a. Initialisation of socket
    b. Connect on socket (Client)
    c. Create and attach new socket (Server)
    d. Bind and connect socket (Server)
    e. Listen on socket (Server)
    f. Accept connection on socket (Server)
    g. Process control calls on socket and provide interface to RFCOMM multiplexor channel(Server)
    h. Get peer address (server)
    i. Send and Receive data on socket. (to/from remote device)
    j. Detach and destroy socket
    k. Disconnect, Close and Abort connection on socket
    l. Handle Session task
    m. Connection and confirmation Indicator
5. Defining Sessions Interface functions
    a. Only one RFCOMM sessions between any pair of device
    b. When establishing new DLC
        i. Check if RFCOMM session is already there
            1. If so establish new DLC on that
    c. If there is one RFCOMM session, close L2CAP connection due to one RFCOMM connection only
    d. Process connect on RFCOMM session
    e. Establish an L2CAP channel to the peer RFCOMM entity using L2CAP service primitive
    f. Send/Receive data on RFCOMM session
    g. Start the RFCOMM multiplexor by sending SABM command on DLC and await UA response from the peer entity
    h. Close and disconnect all DLC for given session

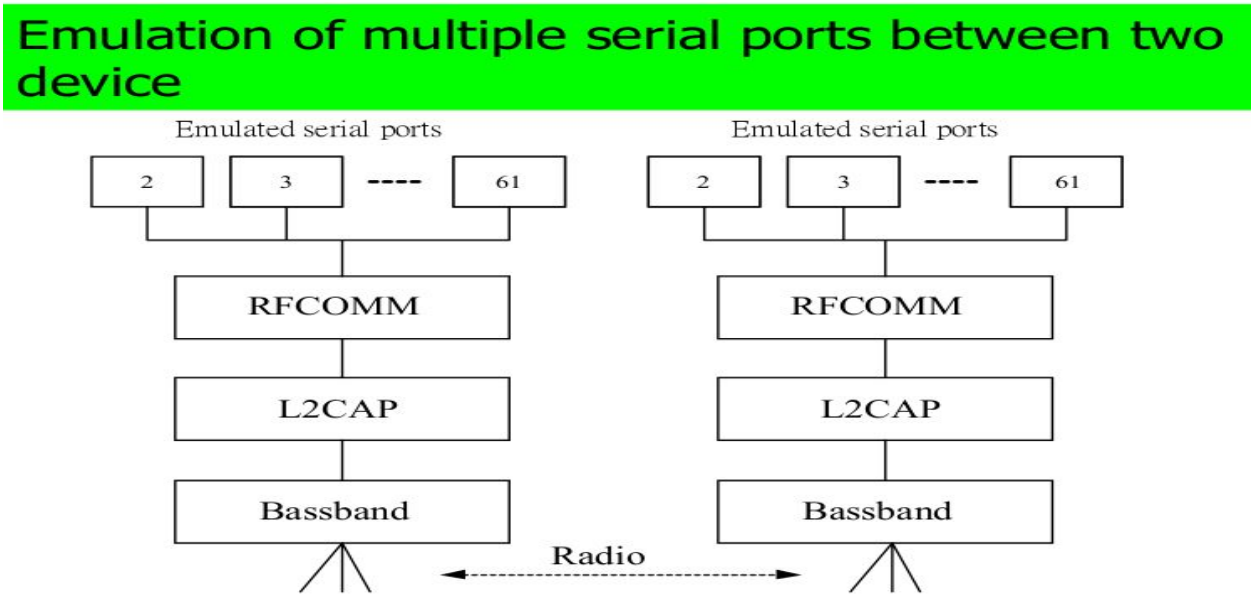
6. Link Loss Handling
    a. If Notification received for L2CAP link loss
        i. RFCOMM session should free all the resources
7. Process functions for incoming RFCOMM,
    a. Process functions for SABM, DISC, UA, DM frame
8. Process functions for RFCOMM commands
    a. Process functions for MSC, RPN, RLS, PN commands

B. Implementation of RFCOMM Serial Port Profile

Description of Configuration file of RFCOMM Serial port profile which is going to be implemented.

1. Declaring global functions
2. Parse command line options
    a. Bluetooth Address
    b. RFCOMM channel for profiles
        i. LAN
        ii. Serial Port
    c. Open pseudo terminal and run in background
3. Set Signal Handlers
4. Operations in Client Mode
    a. Functions definitions to open RFCOMM connection at specified address server and channel or else obtain it via SDP
    b. Provide access to the server remote serial port via stdin/stdout
5. Operations in Server Mode
    a. Become Daemon
    b. Listen on any address and advertise virtual serial port via SDPD
6. Display usage message and exit functions

Diagram below demonstrates the Phase 2 work



Phase 3:

SDP protocol is divided into four types. They are listed below

- Service Discovery Protocol
- Service Discovery Protocol Database
- Service Discovery Protocol Query Control
- Service Discovery Protocol for RFCOMM

Implementations of SDP protocol is given in detailed information.

A. Service Discovery Protocol (Header and Configuration file**)**

1. Defining macros for data representation
    a. Data Element Header field
    b. Data Element field → Type and Size Descriptor
2. Service Attributes Definitions and Descriptions
    a. Universal Attribute Definitions
    b. Service Discovery Server
    c. Service Class Attribute Definitions → Attribute ID, Attribute value(Data Element)
    d. Protocols Descriptor list and Attribute definitions
3. Protocol Description
    a. PDU Format definitions → Header, Parameter and Transaction type
4. SDP Protocol Data manipulation and Routines → defining functions to get info, read, write, examine of data elements

5. SDP connection
    a. Creating a session → implement a functions for open, open_local, close, error
6. SDP services after connections
    a. Registration of services
    b. Unregistration of services
    c. Change and error in services
7. SDP Search API declarations
    a. Service Search Request
        i. Service Search Pattern → UUID list, Service Record list
        ii. Maximum Service record Count
    b. Service Search Response

B. Service Discovery Protocol Database(Configuration file**)**

1. Start SDP daemon
    a. Set Signal handlers and Initialise server
2. Description of Profile
    a. Attribute and Profile descriptor
    b. Create Profile for LAN and SP
    c. Function to find profile and attribute descriptor
    d. Create structure for Profile descriptor List, Service class ID List, Service provider, Protocol Descriptor List, Browse Group List, Service name, Service database State, Version number List
3. Description and Definition of Structure for Service Provider
    a. Register and Unregister of SDP
    b. Updation of a Provider data
    c. Record handler of a provider
4. Structure for SDP Server
    a. Functions File descriptor Index entry
    b. Establishing Server for L2CAP connection
        i. Allocate memory for descriptor index and incoming buffer
    c. Accept connection from client
    d. Process Request from the Client
    e. Close file descriptor and close the server
5. External API for Transaction Type
    a. SDP_Service, Register and Unregister response
    b. SDP_Service Search Request and Response
    c. SDP_Service Attribute request and Response
    d. SDP_Service Search Attribute Request and Response
    e. SDP_service Error Response

C. Service Discovery Protocol Query Control (Configuration file**)**

1. Process Commands by using Service Search Attribute Requests
2. Functions to print information

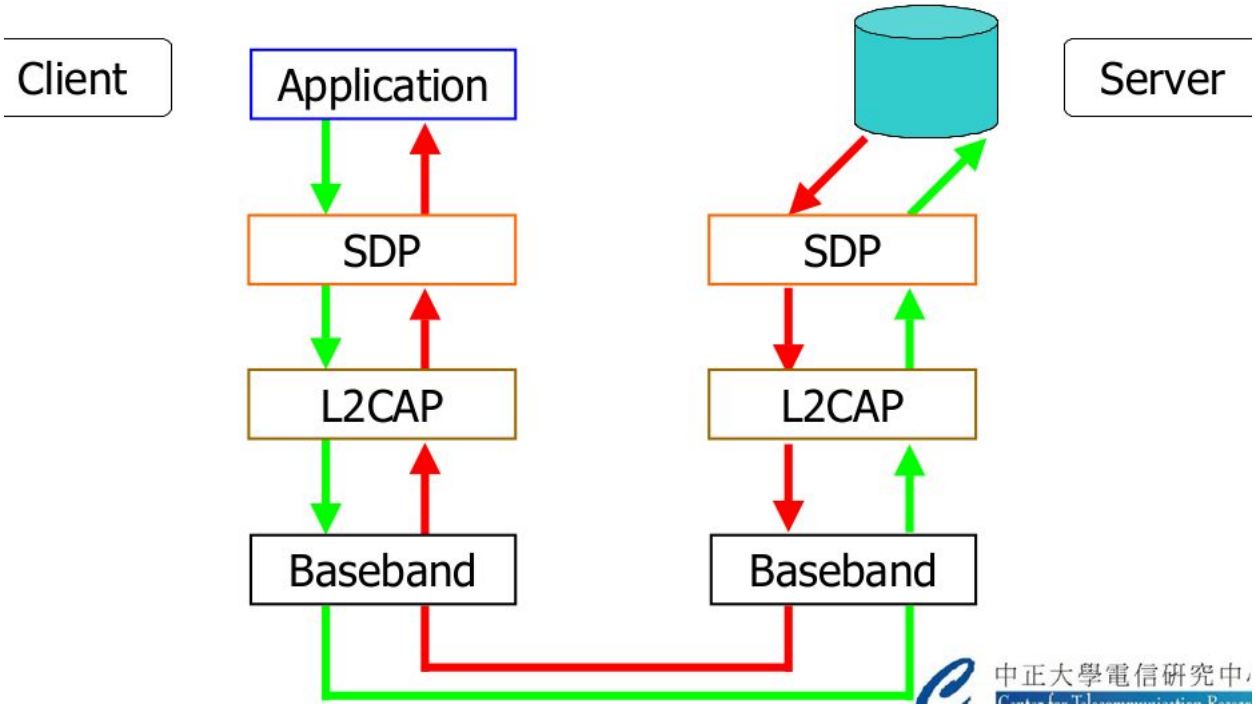
a. Service Record Handle and Class ID list
b. Protocol Descriptor list
c. Bluetooth Profile descriptor List

D. Service Discovery Protocol for RFCOMM(Configuration file**)**

1. Functions to look for RFCOMM channel number in the Protocol Descriptor List
2. Parse Protocol Descriptor List
    a. Function to identify a communication protocol
    b. Function to provide Protocol Specific parameters
        i. L2CAP PSM (Protocol/Service Multiplexor)
        ii. RFCOMM server channel number

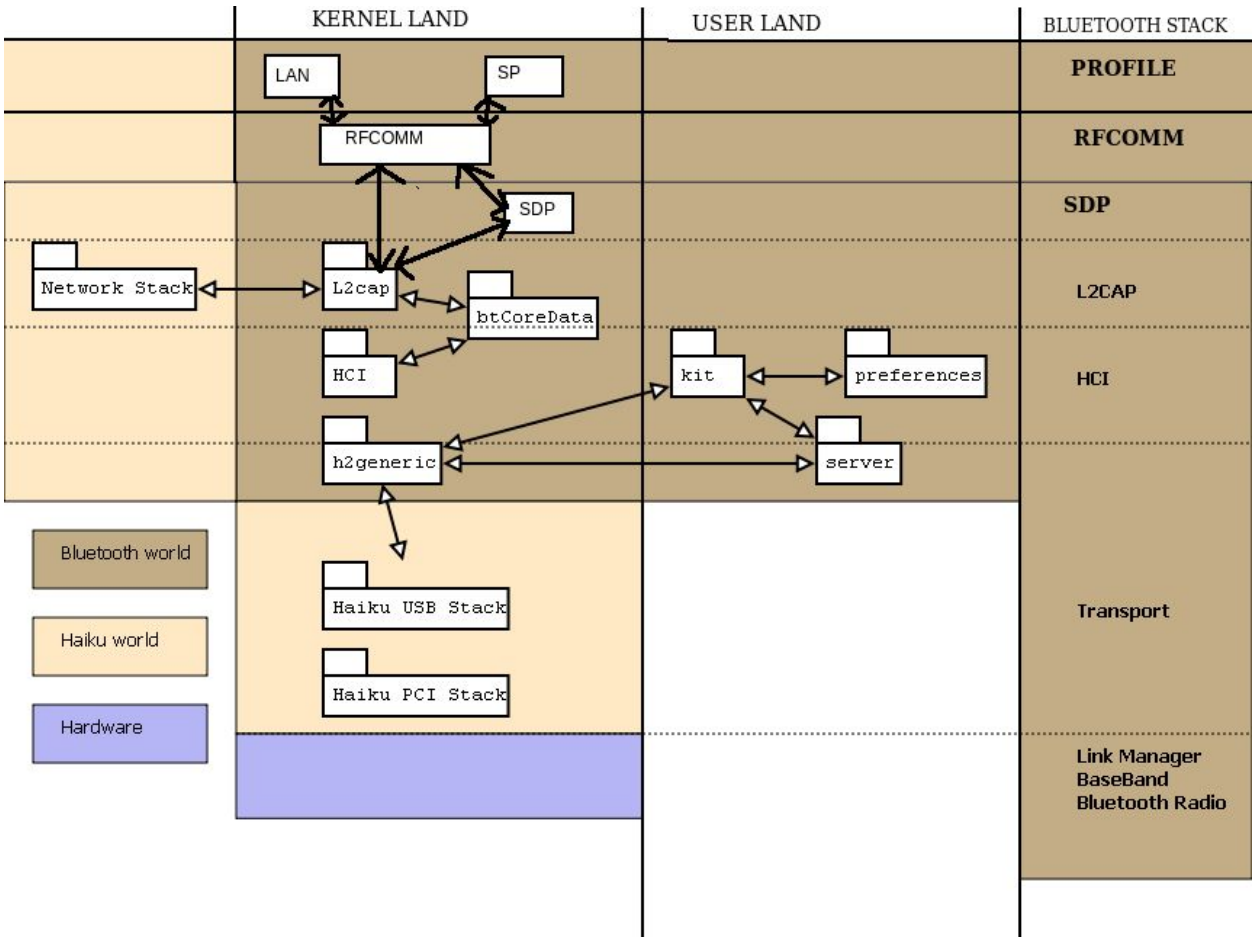
Diagram below demonstrates the Phase 3 work.

SDP protocol to bound over L2CAP protocol and linked with RFCOMM protocol.



Overview of Bluetooth Protocol Stack of Haiku

(After Improvement and Implementation of new Protocols)



The diagram given above provides an overview of the layered Bluetooth architecture for Haiku after implementation of new protocols and improvement of old bluetooth stack. All the improvements and implementation are described in the **Implementation Detail section** and detailed information has been given in the **Phases**. These works will be done in Kernel Land.
Communication between layers would be done using sockets(endpoint of a communication link). Once the transport protocol and port number to communicate on are chosen, the rest of Bluetooth communications is essentially the same type of programming most network programmers are already accustomed to: sockets! A server application waiting for an incoming Bluetooth connection is conceptually the same as a server application waiting for an incoming

Internet connection, and a client application attempting to establish an outbound connection behaves the same whether it is using RFCOMM, L2CAP, SDP, TCP, or UDP.

The terms and technologies used as well as the keywords(abbreviations) used while depicting in the Phases are explained below as a Glossary section in the ascending order of alphabets.

Haiku Bluetooth Diagram Glossary and Terms used in the Phases

ACL Asynchronous Connections Link. One of the two types of data links defined for the Bluetooth Systems, it is an asynchronous (packet-switched) connected between two devices.
Bluetooth Device Class A parameter that indicates the type of device and which types of services that are supported. The class is received during the discovery procedure.
Bluetooth Service Type One or more services a device can provide to other devices. The service information is defined in the service class field of the Bluetooth device class parameter.
Channel A logical connection on the L2CAP level between two devices serving a single application or higher layer protocol.
Connectable Device A Bluetooth device in range that will respond to a page message and set up a connection.
DCID Destination Channel Identifier, used as the device local end-point for an L2CAP transmission. It represents the channel endpoint on the device receiving the message. It is a device local name only .
Destination The Bluetooth device receiving an action from another Bluetooth device. The device sending the action is called the source. The destination is typically part of an established link.
Device Discovery The mechanism to request and receive the Bluetooth address, clock, class of device, used page scan, and names of devices.
Discoverable device A Bluetooth device in range that will respond to an inquiry message.
DLCI Data Link Connection Identifier. This is a 6-bit value representing an ongoing connection between a client and a server application. It is used in the RFCOMM layer.
HCI Host Controller Interface. An (application-optional) layer which provides a command interface to the Baseband layers.
L2CAP Logical Link Controller and Adaptation Protocol. This protocol supports higher level protocol multiplexing, packet segmentation and reassembly, and the conveying of quality of service information.
LAN Local Area Network.
Master device A device that initiates an action or requests a service on a piconet. Also the device in a piconet whose clock and hopping sequence are used to synchronize all other

devices in the piconet.
MUX Multiplexing Sublayer. A sublayer of the L2CAP layer.
Name Discovery The mechanism to request and receive a device name.
Non-connectable Device A device that does not responds to paging is said to be in non-connectable mode. The opposite of a non-connectable device is a connectable device.
Non Discoverable Device A device that cannot respond to an inquiry is said to be in non-discoverable mode. The device will not enter the inquiry response state in this mode.
Packet Format Each packet consists of 3 entities, the access code, the packet header and the payload. Their are a number of different packet types.
Packet header The header contains link control info and consists of 6 fields: AM_ADDR : active member address, TYPE : type code , FLOW : flow control, ARQN : acknowledge indication, SEQN : sequence number & HEC : header error check. The total size of the header is 54-bits.
Packet Switched A network that routes data packets based on an address contained in the data packet is said to be a packet switched network. Multiple data packets can share the same network resources.
Packet Type 13 different packet types are defined for the baseband layer of the Bluetooth system. All higher layers use these packets to compose higher level PDU's. The packets are ID, NULL, POLL, FHS , DM1 ; these packets are defined for both SCO and ACL links. DH1, AUX1, DM3, DH3, DM5, DH5 are defined for ACL links only. HV1, HV2, HV3 , DV are defined for SCO links only.
Page State A mode that a device enters when searching for other devices. The device sends out a page packet (ID packet), using the page hopping sequence, to notify other devices that it wants to know about the other devices and/or their services.
Payload format Each packet payload can have one of 2 possible fields, the data field (ACL) or the voice field (SCO). The different packets, depending on whether they are ACL or SCO packets can only have one of these fields. The one exception is the DV packets which have both. The voice field has a fixed length field, with no payload header. The data field consists of 3 segments: a payload header, a payload body and a CRC code (with the exception of the AUX1 packet).
PDU Protocol Data Unit. (i.e., a message.)
PPP Point to Point Protocol.
Profile A description of the operation of a device or application.
RFCOMM Serial Cable Emulation Protocol based on ETSI TS 07.10.
RS-232 A serial communications interface. Serial communication standards are defined by the Electronic Industries Association (EIA).
SAR Segmentation and Reassembly. A sublayer of the L2CAP layer.
SDPD Service Discovery Protocol database that contains the service discovery-related information.
SDP Client The SDP client may retrieve information from a service record maintained by

the SDP server by issuing an SDP request.

SDP Server The SDP server maintains a list of service records that describe the characteristics of services associated with the server.

SDP Session The exchange of information between an SDP client and an SDP server. The exchange of information is referred to as an SDP transaction.

SDP Transaction The exchange of an SDP request from an SDP client to an SDP server, and the corresponding SDP response from an SDP server back to the SDP client.

SEQN Sequential Numbering scheme. It provides a sequential numbering scheme to order the data packet stream.

Serial Interface An interface to provide serial communications. service This term refers to a service that one device provides for others. Examples are printers, PIM. synchronization servers, modems (or modem emulators).

Service (SDP layer) A service is any entity that can provide information, perform an action, or control a resource on behalf of another entity. A service may be implemented as software, hardware, or a combination of hardware and software.

Service Attribute Each service attribute describes a single characteristic of a service.

Service Class Each service is an instance of a service class. The service class definition provides the definitions of all attributes contained in service records that represent the instances of that class.

Service Layer The group of protocols that provides services to the application layer and the driver layer in a Bluetooth device.

Service Record A service record contains all of the information about a service that is maintained by an SDP server.

Service Record Handle A service record handle is a 32-bit number that uniquely identifies each service record within an SDP server.

Slave Device A device in a piconet that is not the master.There can be many slaves per piconet.

Source The Bluetooth device initiating an action to another Bluetooth device. The device receiving the action is called the destination. The source is typically part of an established link, though not always (such as in inquiry / page procedures).

UA Channel User Asynchronous data channel. One of the 5 logical channels defined for the bluetooth system. The UA channel carries L2CAP transparent asynchronous user data. It is normally carried in the ACL link.

UART Universal Asynchronous Receiver Transmitter. A device which converts parallel data into serial data for transmission, or it converts serial data into parallel data for receiving data.

UI Channel User Isochronous data channel. One of the 5 logical channels defined for the bluetooth system. The UI channel carries L2CAP transparent isochronous user data. It is normally carried in the ACL link. It is supported by timing start packets at higher levels.

UUID Universal Unique Identifier. Used in the SDP layer.

Testing:

To implement all the phases successfully, Testing is very much important and critically needed. The ability to tests everything easily might help a user to make sure that everything works fine on his Haiku OS. Testing will be performed in two main ways. They are as follows:

- **Unit Testing** → The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality. Unit testing will be heavily needed while writing source code for the different number of individual Phases for Bluetooth Stack.
- **Integration Testing** → The goal of Integration testing is to test whether combined parts of a Bluetooth Protocol Stack function works correctly and efficiently. The goal of Integration testing would be achieved if all the protocols would work properly.
  To perform the Integration Testing, it should have a suitable development environment both in terms of hardware and software. Following are the list of some hardware/software I have and would be used throughout the project. They are given as follows:

1. Hardware :
    a. DELL Laptop of processor Intel® Core™ i5-7200U CPU @ 2.50GHz × 4, 8GB Integrated RAM with an in-built USB Bluetooth(Intel Corp.) .
    b. A Smartphone of Samsung Galaxy J7(2016) of CPU Octa-core (4x1.4 GHz Cortex-A53 & 4x1.0 GHz Cortex-A53) Octa-core 1.5 GHz Cortex-A53 with a Bluetooth version(4.1).
    c. An USB (pen-drive)Stick for a bootable Haiku Image, (an alternative storage device).
2. Software:
    a. Host Operating System(OS) : Kali Linux version (2018.02) Kernel Image: 4.18.0-kali2-amd64 x86_64 GNU/Linux.
    b. VirtualBox (version 6.0): Running Haiku OS with an Nightly Image of (hrev 52989) x86_64 version.
    c. Smartphone OS: Android 7.0 “Nougat”.

Smartphone(Samsung Galaxy J7) and Laptop(Dell) will be used in Bluetooth Testing

connection between two devices. If any of the devices failed to work properly, then new device will be used in place of old device while testing if needed according to the demand. Using these given hardware/Software, the three phases will be tested. Each of the testing of layers has been elaborated below:

- Using L2CAP layer, device authentication, pairing, ping(using L2CAP layer) and functional testing of L2CAP layer using data transfer between two devices.
- Using RFCOMM protocol layer, connection establishment between two devices. To check whether the device acts as a server and client properly by using one of the Serial Port profile of RFCOMM.
- Using SDP protocol layer, the service discovery, Inquiry Procedure of the device when acting as a client and server with respect to remote devices.

References:

- http://urnenfeld.blogspot.com/search/label/bluetooth
- https://en.wikipedia.org/wiki/Bluetooth
- https://en.m.wikipedia.org/wiki/List_of_Bluetooth_protocols
- https://www.freebsd.org/doc/handbook/network-bluetooth.html
- https://www.netbsd.org/docs/guide/en/chap-bluetooth.html

Thank You for Reading !!