NDN, Technical Report NDN-0015, 2013. http://named-data.net/techreports.html
Revision 1: November 7, 2013

NDNBlue: NDN over Bluetooth

Arjun Attam*, Ilya Moiseenko†
*Indian Institute of Technology, Delhi. arjunattam@gmail.com
†University of California, Los Angeles. iliamo@ucla.edu

***Abstract*—Named Data Networking is a network protocol that in future is able to displace IP protocol from its current role of a dominating inter-networking layer and also to have an edge in such areas of networking as PAN (Personal Area Networking) where IP protocol has never been universally used. This project looks into the problem of fitting Named Data Networking request-response communication model into constrained Bluetooth networking stack for both mobile and desktop platforms. We provide a cross platform proxy layer, which works between NDN stack and Bluetooth stack to achieve NDN connectivity over Bluetooth links for multiple platforms.**

***Index Terms*—Information-centric networks, named-data networking, bluetooth, device-to-device communication**

I. Introduction

NDNBlue is an endeavor to extend the scope of NDN applications with Bluetooth. The present CCNx software router [1] implementation for Named Data Networking works with TCP and UDP tunnels and also can be extended with NDNLP [2] — a link level protocol for Ethernet.

With the advent of mobile devices, Bluetooth has become a popular wireless technology standard used for exchanging data over short distances by creating a personal small area networks. Bluetooth focuses on short distance communication, typically around 30 ft. As compared to traditional wireless, Bluetooth enables applications and devices to connect through an ad-hoc, low-cost and power efficient medium.

Bluetooth development is a very platform and manufacturer dependent, and therefore, NDNBlue is currently restricted to a few platforms for now: Linux and Android. In operation, NDNBlue is similar to NDNLP as it operates as a proxy layer between the NDN stack (network layer) and the Bluetooth stack (link layer.) It uses RFCOMM, a reliable transport protocol for Bluetooth.

This document describes the design of a proxy layer, which exposes Bluetooth interface to NDN software router. Section VII contains protocol choices and security caveats for Bluetooth. The source code is publicly available on Github and links are located in section VIII.

II. Requirements

NDNBlue works on Linux and Android devices. NDNBlue is compatible with BlueZ, the official Bluetooth protocol stack on Linux (refer to the implementation specifics section for more information on Bluetooth stacks and platforms.) NDNBlue requires the NDN software router and its C library to be installed on Linux and the corresponding services application on Android.

III. Configuration

The process of establishing a Bluetooth connection to run NDN is initiated by devices finding other physically proximate Bluetooth enabled devices. The user needs to decide between an incoming connection (server) or an outgoing connection (client.)

A. Discovering Devices

NDNBlue for Android can discover nearby Bluetooth devices to identify the server device for outgoing connections. The process requires that the server be set as discoverable by other Bluetooth devices.

B. Security caveats

It is recommended that the devices should be authenticated using the one-time pairing process in the OS level settings before starting NDNBlue. This is to eliminate the differences in security implementations by various Bluetooth stacks.

IV. Design

To add a Bluetooth interface to NDN software router, NDNBlue follows a similar approach as NDNLP. NDNBlue is implemented as NDN application behaving as a proxy between Bluetooth platform dependent stack and NDN platform independent stack, and runs on Linux and Android. Alternatively, NDNBlue could be integrated into the NDN software router code base, but this option was disregarded due to frequent changes of the NDN daemon itself, which would lead to a big maintenance costs and stability issues of this solution. As a result, by working between the Bluetooth stack and NDN software router, NDNBlue relays packets and enables a user to add a Bluetooth face to the router.

A. Bluetooth transport level

NDNBlue uses RFCOMM as the transport protocol for Bluetooth. In addition, NDNBlue uses the Bluetooth Service Discovery Protocol (SDP) to register a server application which allows client devices to find out the RFCOMM channel of the server [6]. By using a common hard coded UUID, NDNBlue over Android can connect to the NDNBlue on BlueZ.

NDNBlue uses nonblocking Bluetooth sockets, via polling (Linux) and threading (Android.)

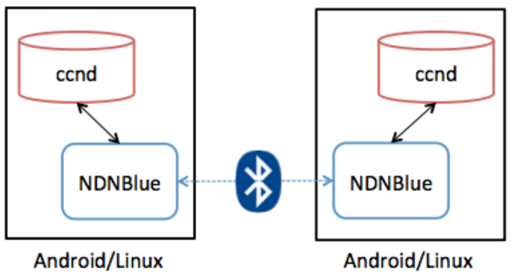

Figure 1: Basic architecture

B. NDNBlue on Linux

NDNBlue on Linux is dependent on the NDNLP implementation. NDNLP and NDNBlue can work side-by-side. It connects to local `ccnd` using a UNIX domain socket face, and it is dependent on `ccnd` for routing and forwarding. Unlike NDNLP, NDNBlue does not support control face for control commands. Face management and prefix registration are handled by NDNBlue which can manage one Bluetooth connection.

To use NDNBlue on Linux, download the source from Github and build binaries with the `waf` files. After ensuring that `ccnd` is running, use one of the following:

- `ndnblue /prefix server`
- `ndnblue /prefix client server-MAC-addr`

You can choose any `prefix` as desired, and `server-MAC-addr` is the 48-bit MAC address of the NDNBlue server.

C. NDNBlue on Android

The restrictive nature of Android enforces certain changes to the approach taken by NDNBlue to communicate with the CCNx Service Control application. UNIX domain sockets on Android require administrator user privileges through native code which is not possible without changing the stock firmware and bootloader of the device through rooting. Therefore, unlike NDNBlue for BlueZ, NDNBlue on Android connects to CCNx using TCP sockets, which do not require any changes to the device firmware. By compiling CCNx with the `-DCCN_LOCAL_TCP` flag, TCP sockets are used instead of UNIX domain sockets. [3]

This Java implementation of NDNBlue on Android transmits `ccnb` encoded packets over Bluetooth and registers itself as a callback handler for Interests and ContentObjects at local `ccnd`. Simple face management and prefix registration is handled by NDNBlue.

To use NDNBlue for Android, build the code on Github and ensure CCNx services are running on the device. Then,

- Pair Bluetooth devices using OS level settings.
- Open app on device and choose a prefix
- Start server, or,
- Discover server's Bluetooth address and start client

After the connection is established, CCNx apps on Android can be used over Bluetooth.

V. Tests

NDNBlue was tested with CCNx 0.7.1 running on two laptops with Xubuntu 12.04 32-bit and two Samsung SCH-I535 phones running Android 4.2.1. The laptops have Broadcom Bluetooth adapters, and one of them runs in a VirtualBox environment. Before running NDNBlue, the devices were paired using the Bluetooth settings at the operating systems level. For Android devices, pairing requires the devices to be in discoverable mode. In our case, once the devices are paired, it is possible for establish a connection even if the devices are not discoverable, but this is device dependent.

We wrote an Android version of `ccnping` [4] to test communication between two devices running NDNBlue, across Linux and Android platforms. Along with the Bluetooth interfaces, the test devices were also connected to other TCP/UDP/IP interfaces, and NDNBlue faces behaved as expected.

The RTT as measured using ccnping were between a minimum of 59 ms to a maximum of 152 ms, with a median of 98 ms and there were no timeouts.

VI. Limitations and Future Work

With the variations in Bluetooth stacks and platforms, NDNBlue only supports Linux and Android. While NDNBlue works across different Bluetooth platforms, the application code is not portable across different stacks.

The current implementation of CCNx Service Control on Android requires an IP address to start services and therefore the Android device needs to be connected to a wireless network before NDNBlue can be used. This affects the ability of Android devices to setup ad-hoc Bluetooth networks running NDN. The issue can be resolved by changing the CCNx implementation on Android to accommodate the possibility of using non IP based networks. However, once the services start, running NDNBlue does not require connection to a wireless network.

As of now, NDNBlue is restricted to only one Bluetooth connection per device. Since SDP is supported, it is possible to use dynamically allocated RFCOMM channels in server devices, and thereby alleviate this limitation. Currently, there are no APIs to support broadcast communication in a Bluetooth piconet, which can have one master with at most seven slaves, and thus NDNBlue cannot broadcast messages.

VII. Implementation specifics

A. Platforms

In most cases, Bluetooth radio devices are bundled with their own drivers, libraries and tools, collectively referred to as Bluetooth stacks, which can be roughly divided into two categories:

- Desktop operating systems implementations: focussing on manufacturer flexibility and diverse use cases
- Embedded systems implementations: focussing on specific features limited by the scope of the portable devices like phones, automotive controls and health peripherals.

As of now, every desktop operating system has a different stack, and in some cases there are multiple stacks available by default, for example Widcomm and MSFT Bluetooth for Microsoft Windows. BlueZ is the most popular Bluetooth stack for Linux distributions, and it supports all core Bluetooth protocols and layers. Before Android 4.2, the BlueZ stack was also used by plurality of Android phone manufacturers, with the exception of Samsung. However, with the advent of 4.2, Android has moved to a different proprietary stack developed by Broadcom called Bluedroid. [7] The Android documentation recommends using the Bluetooth Java APIs in the Android SDK to ensure portability over Android devices. Therefore, the original idea of having a common BlueZ C code to run on Linux and Android was consequently dropped since native Android development for Bluetooth has issues with device portability.

B. Device Discovery

A Bluetooth device can either establish an outgoing connection (client) or an incoming connection (server.) The words client and server are only used to signify who initiates the conversation, and does not imply the client-server relationship of typical network programming. It is possible for Bluetooth server device to function as a client, and vice versa. Before establishing the connection, a client device needs to choose a target device and a transport protocol, while a server device needs to choose a transport protocol and then begin listening for incoming connections. The target device address is found by device discovery.

Bluetooth devices are identified with a unique static 48-bit address, which is identical to the Machine Address Code (MAC) address for Ethernet and a user friendly name, which may or may not be unique. Unlike TCP/IP where Ethernet MAC addresses are discarded for practical purposes in the higher layers, to establish an outgoing Bluetooth connection, a client application needs to know the unique Bluetooth address of the remote device. The device inquiry process is used to detect nearby Bluetooth devices by broadcasting a special message and waiting for replies, which helps to identify the remote devices. The Bluetooth discovery process involves inquiry scan of roughly 10 seconds, followed by a page scan to find user friendly names. [8]

A Bluetooth device can control whether it is discoverable by other Bluetooth devices, and different manufacturers have different default settings. Typical Linux machines are discoverable by default, but almost all Android phones cannot be detected by other Bluetooth devices by default, and they need to be declared discoverable for the discovery process. This is done using the Android device settings.

C. Security

With the diversity in Bluetooth stacks, security implementations vary. Bluetooth authentication and encryption is typically handled at the operating system level, and therefore some security configuration is recommended before using NDNBlue. The one-time process of authentication between two Bluetooth devices is called pairing, wherein a Bluetooth PIN sequence is used as a shared secret. Pairing of Bluetooth devices is handled by the Bluetooth settings provided by the operating system, and it enables Bluetooth applications on each device to request encryption and authentication as required.

D. Transport Protocol

RFCOMM is the only protocol supported by all Bluetooth stacks. For example, BlueZ is bundled with multiple transport protocols, including RFCOMM and L2CAP, but as of now the Android SDK only supports RFCOMM [5]. RFCOMM is a general-purpose reliable streams-based protocol and it closely resembles TCP. However, unlike TCP, RFCOMM only supports 30 ports or channels, which significantly affects the choice of channel for server applications. Due to its widespread usage, RFCOMM is entitled to support and documentation.

The Service Discovery Protocol (SDP) for Bluetooth enables a remote client device to discover services identified with a 128-bit Universally Unique Identifier (UUID.) [6]

VIII. Source code

- github.com/arjun27/NDNBlue-Android
- github.com/arjun27/NDNBlue-BlueZ
- github.com/arjun27/CCNPing-Android

References

[1] "CCNx" [Online]. Available: http://www.ccnx.org

[2] Junxiao Shi, Beichuan Zhang, "NDNLP: A Link Protocol for NDN", Technical Report NDN-0006, 2012 [Online]. Available: named-data.net/techreport/TR006-LinkProtocol.pdf

[3] "CCNx - Feature #100760: TCP connections to ccnd" [Online]. Available: redmine.ccnx.org/issues/100760

[4] "ccnping" [Online]. Available: github.com/NDN-Routing/ccnping

[5] "Android BluetoothSocket API" [Online]. Available: developer.android.com/reference/android/ bluetooth/BluetoothSocket.html

[6] Tim Howes, "Discovery Whitepaper: Service Discovery Applications", Bluetooth Special Interest Group [Online]. Available: www.bluetooth.org/DocMan/handlers/ DownloadDoc.ashx?doc_id=144841

[7] "Broadcom Press Release" [Online]. Available: broadcom.com/press/release.php?id=s721534

[8] Albert Huang, "An Introduction to Bluetooth Programming" [Online]. Available: people.csail.mit.edu/albert/bluez-intro/