Network IO library for Android to generate raw packets

Mainak Biswas
2011MCS2586
mcs112586@cse.iitd.ernet.in

Sathyam Doraswamy
2011MCS2591
mcs112591@cse.iitd.ernet.in

September 15, 2012

1 Motivation

Most smartphones do not expose a network library to send raw IP packets. Applications are only allowed to send TCP and UDP packets without being able to modify TTL fields, MSS advertisements, etc, or send ICMP packets. Android supports a native application development environment (NDK) that can expose this functionality. These functionalities, once exposed, can provide the user with options to have better control over the packets sent out into the network through simple Java API. These can also be used by researchers carring out research on various network layer parameters to improve network performance. Since most of the research is carried out on wireless network these days, using smartphones for performing experiments, such a library can prove to be immensely helpful. Thus it was decided to use the power provided by NDK toolset to use native-code languages such as C/C++ to implement a network IO library in Android to expose a richer API to applications to do raw socket programming.

2 Broad overview of the project

In this project we plan to build a network IO library for Android developers using the NDK toolset to provide an API to set/modify packet headers both at the transport and the IP layer and to to do raw socket programming.
Raw socket programming can be done using C/C++. To provide a similar functionality to android developers who code using Java JNI interface needs to be used.
A JNI interface is a programming framework that enables java code running in a Java Virtual Machine (JVM) to call, and to be called by, native application code (programs specific to a hardware and operating system platform) and libraries written in other languages such as C, C++ and assembly.
The java code written is compiled to generate the java class file. The javah (C header and Stub File Generator) produces C header files and C source files from a java class. These files provide the connective glue that allow Java and C code to interact with each other. The C/C++ compiler compiles the C files conatining native code to object module. A shared library is then built which

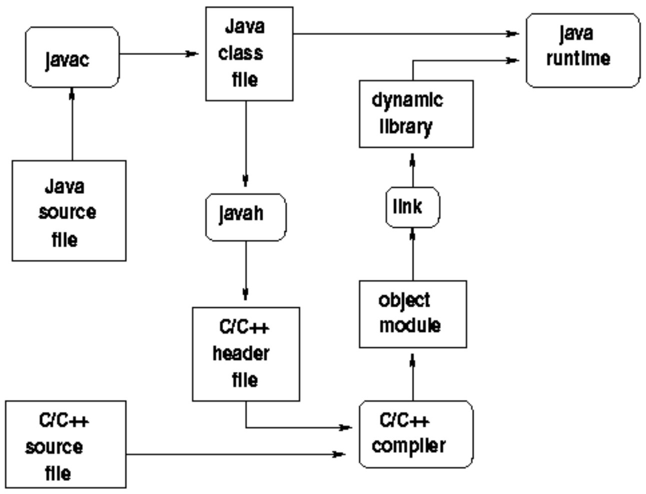

Figure 1: An overview of the JNI framework

is used by Java Runtime System.
In this way any C/C++ code that is written can be used by the android java program. This makes it feasible to expose raw socket programing functionality to android java programmers.

3 Feasibility Study

In order to ascertain that a library for network IO could be implemented using the NDK toolset a sample test application was built. First of all the android SDK and NDK was installed on two computers using MacOS and Ubuntu 11.10. Then the eclipse development environment to build android applications was set up. The installation was tested by deploying a sample program consisting of native code. Deployment was done both on a phone and an emulator. The application simply printed a message that was returned by native C code. Further many other sample applications using native code features were tested.

To explore the raw socket programming capabilities provided by C a sample code was written using linux socket API to send custom packets, where the TTL field, sender IP, destinaion IP and port numbers could be modified in the packet being sent. On the receiver side, Wireshark was used to sniff the network in order to make sure that the custom packets are indeed being sent.

To test whether these network features can be used in android, a new JNI application was written that used the same C code as above. This application was deployed on a phone which was connected to the IITD WiFi network. Custom packets sent by the phone were tested by Wireshark and we got positive results.

Thus we came to the conclusion that a network IO library that provides an API to generate custom packets is feasible.

4 Work Plan

The basic concept of low level sockets is to send a single packet at a time with all the protocol headers filled in by the program (instead of the kernel). We plan to provide an API to modify fields as specified in the following table.

TCP Header	Source Port, Destination Port, Sequence Number, Acknowledgement Number, Data offset, Reserved, Flags, Window size, Checksum, Urgent Pointer, Options, Padding
UDP Header	Source Port, Destination Port, Length, Checksum
IP Header	Version, Type of Service, Total Length, Identification, Flags, Fragmentation offset, TTL, Protocol, Destination IP, Options, Checksum
ICMP Header	Type, Code, Checksum

Apart from the fields mentioned in the table, if some other field needs to be modified, API could be provided for that also.