Summer Intern Project Report

Aman Kumar Bhupendra Kastore

Project Guide: Shyjumon N

Contents :

1. IPv4 and IPv6 ping application (android platform)
2. Python script
3. Packet flow
4. Patch file for CTS pingTest

Android Ping Application

# Usage

There are two applications, one serves the purpose of server (Myping.apk) and other is client (PClient.apk) .

# Installation

Open adb shell and type

adb install path-to-Myping.apk

adb install path-to-PClient.apk

# Working

Myping application requires no input . Just run the application in the background on the device being pinged.

PClient application requires IP address and a port number. Both IPv4 and IPv6 addresses of the device will be displayed in Myping start screen. Use these addresses as input to PClient application. When pinging with IPv4 address use port 8080 and use 8081 with IPv6 address.

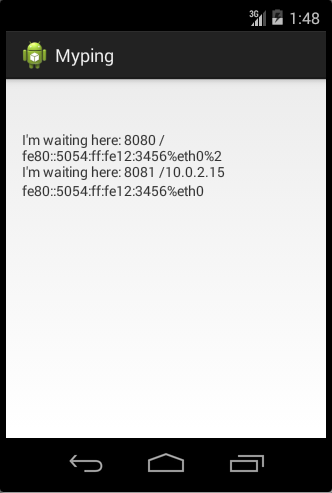
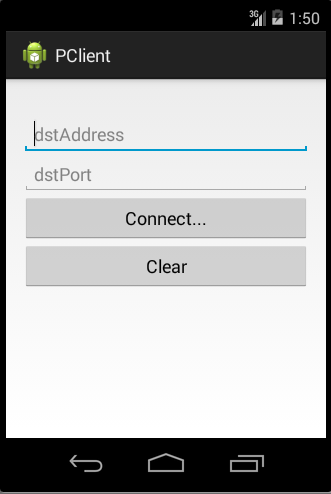
On successful ping the server will reply with a message and IP address of client will be displayed on the server screen. Note that if server is pinged on IPv4 (or IPv6) address, IPv4 (or IPv6) address of client is used as well as displayed on server screen.

**Network Requirement**

The application works only on wifi connectivity. The routers in between should also be able to handle IPv6 packets (if using IPv6 address for connectivity).

GSM connectivity for IPv6 is not supported due to inability of gsm service routers to handle IPv6 packets.

# 

****

Screenshots of Myping and PClient

# Internals of Working

The application uses socket connectivity. Sockets are application layer API that uses TCP/IP protocols through which connection can be established. For making a connection a **server** socket and a **client** socket is required.

server socket continuously listens to incoming connections. While creating a server socket an IP address and a port number is required (called binding address). Binding address needs to be specified which socket creation. In case of binding to IPv6 address , interface (eth0, wlan0, ..etc ) of IPv6 address must also be specified. Client must specify this address and port number while making the connection.

Client socket is used to connect to a server socket using server’s bind address. After the connection is established , reads and write calls can be made to send datagram packets from both sides.

# Usefulness

* Self Ping
  + The application can be used to send ping message to same device using loopback address. For Ipv4 loopback address is “127.0.01” and for IPv6 loopback address is “::1”.
* Travels through all OSI layer
  + Normal ping command uses ICMP messages to communicate, which originates and terminates in network layer only. This is not useful if we need to study higher layer functions. Packets sent using sockets travels through entire OSI layer.
* Choice of IP protocol
  + Ping message can be sent over either IPv4 or Ipv6.

Function Tracer Script

# Motivation

Application level logs (logcat) are insufficient to determine the internal working of operating system. To understand how operating system handle flow of packet through different layer, we need to look at functions called by kernel during these operations. Hence we need function call stack.

One way to implement call stack is to add print statement in the beginning of every function. Doing this for Linux kernel is a tedious task because of large number of files. Hence we wrote a script using tools such as lexer and parser to add print statement in the beginning of every functions.

## Usage

Requirements: python 2.7

Python PLY module (A tool for lexer and parser)

**Input**: Takes 4 inputs

1. File containing typedefs datatypes arranged in a single column

2. File containing macros arranged in a single column

3. Root directory containing files/subdirectories (all files in this directory will be converted)

4. Output directory name

**Output**:

files with added printk statement in same hire achy as in root directory.

# Internals of script

Lexer is tool to find a token satisfying a regular expression from a stream of words. PLY is a python implementation of lexer. It uses LR-parsing which is reasonably efficient and well suited for larger grammars.PLY provides most of the standard lex/yacc features including support for empty productions, precedence rules, error recovery, and support for ambiguous grammars.

To determine beginning of the function we need to specify regular expression of a C function.

A C function starts with a return **datatype** followed by **function name**, ‘**(**‘ , **parameter list** , ‘**)**’ , and ‘**{** ‘ .

Using this information we designed a regular expression of function declaration and added a print statement after **‘{‘** which print the name of the current function and file in which it is defined. Unmatched tokens are printed as it is. Hence in this way a new file is generated along with added printk statement.

Regular expression of function declaration:

*([a-zA-Z0-9]|\\_)+ \s\* \( [^\(\)]\*\s\*[^\(\)]\* \) \s\* \{*

The script also maintains same hierarchy as in the root directory using **os.walk** method in python.

# Usefulness

* This script helps in getting complete function stack in reverse order.
* This simplifies debugging of kernel functions .
* Helps in learning control flow through various kernel layers.

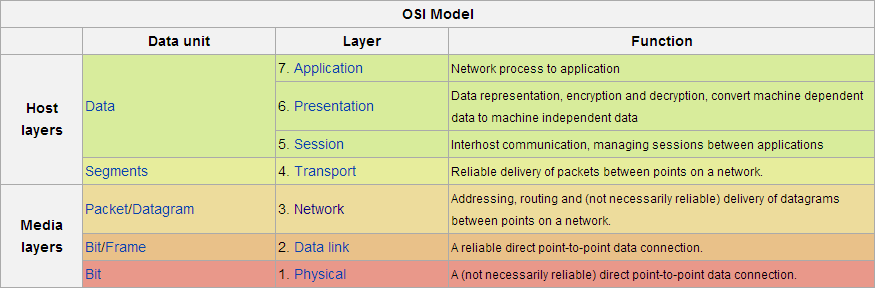
Packet flow

There are two types of packet flow. Outgoing packet flow is termed as TX flow and receiving packet flow is termed as RX flow.

To simplify the flow, the network module is organized into 7 different layers namely : Physical, Data Link, Network, Transport, Session, Presentation, and Application. Each layer has a specific role to play in the packet processing, which is independent from each other.

These layer support one or more protocols and may apply their own header on the original datagram to recognize the protocol followed in these layer.

The complete packet flow , along with some diagrams and important function calls and their importance for both TX and RX flow is described in the attached PowerPoint file “Network Packet Flow (Linux )”.



OSI model and their function (Source : wiki)

Patch file for CTS PingTest

Android CTS test is standard test to verify compatibility of various modules on a certain device. PingTest checks that the device has kernel support for the IPv6 ping socket. This test allows ping6 to run without root privileges. Only Linux 3.11 or above can pass this test. In order to pass this test for other Linux versions some minor modifications need to be done in the kernel code. The exact modifications are described in patch by David Miller.

The commit numbers along with description are discussed below:

1. 6d0bfe : Add IPv6 support to the ping socket

This adds the ability to send ICMPv6 echo requests without a raw socket. Instead of having separate code paths for IPv4 and IPv6, make most of the code in net/ipv4/ping.c dual-stack

## c26d6b4: ping: always initialize ->sin6\_scope\_id and ->sin6\_flowinfo

If we don't need scope id, we should initialize it to zero. Same for ->sin6\_flowinfo.

1. fbfe80c: net: ipv6: fix wrong ping\_v6\_sendmsg return value

ping\_v6\_sendmsg currently returns 0 on success. It should return the number of bytes written instead.

1. a1bdc45: net: ipv6: add missing lock in ping\_v6\_sendmsg
2. cf970c0: ping: prevent NULL pointer dereference on write to msg\_name

A plain read() on a socket does set msg->msg\_name to NULL. So check for

NULL pointer first.

Attachments

* Patch file for kernel 3.4.5
* Python script
* PowerPoint presentation on IPv4 Vs Ipv6
* Documentation on Networking with IPv6
* PowerPoint presentation on Network Packet Flow (Linux)
* Android Ping application (Project files)