An Android Distributed Testing Protocol

By Matt Farmer

Introduction

This document outlines and discusses the *Android Distributed Testing Protocol*, an original protocol designed to aid in the development of Android Applications for Small Businesses. For more information on this protocol or to ask any questions, please contact Matt Farmer via farmdawgnation.com.

Table of Contents

Identifying the Need for Distributed Testing

Android is, in my humble opinion, an excellent platform for mobile application development. I have had the fortune of being able to write a few small applications for Android, and I'm looking forward to writing more. However, one frustration that I have noticed is the tendency for programs to act erratically based on what device is running the application, or what network powers the device.

Unfortunately, this seems to be a class of problems that cannot be completely covered by Unit Testing and Integration Testing. As such, the perfect solution for this problem is for companies to buy as many different Android devices as they can, and test their application thoroughly on each one. However, this is impractical.

My proposed solution to this problem is a distributed testing framework that allows users who wish to be beta testers to download the app to their device, play with it, and fill out a survey about the experience. After which, the app will be removed from the user's device and all of the log files from the user's device, including their survey responses will be documented on a server interface for the application developer to view at his or her leisure.

This document outlines how I envision such a protocol would function on the most basic level. I cover the roles of both the Client and the Server in this situation, as well as outline a comprehensive list of commands that both parties would use to exchange information in this arrangement.

Parts of the Distributed Testing Environment

As with most distributed architectures, this distributed testing environment will consist of several clients connecting to one server. Below, I outline the roles of both the Client and the Server.

The Server

What as I define as “The Server” in this architecture consists of the following parts:

- A repository of different Android APK application files.
- A database associating those files to the developers that own them, and associating “go codes” to each application.
- A storage system (probably also database) for recording the results of user interaction with the application including log files and whatever other pertinent information can be collected from the device.

The server has the following responsibilities:

- When provided with a valid “Go Code” the server must deliver the APK file to The Client.
- When the Client is ready to begin transmitting usage data back to the Server, the Server must store that information completely and accurately in the database for use from the developer’s interface (which is not strictly defined as a part of this protocol, but could very well be a web-based interface).

The Client

What I define as “The Client” in this architecture consists of the following parts:

- A GUI for use by end-users to enter “Go Codes” provided to them by application developers.
- A module that collects log data from the device while the end-user is interacting with the application.
- A GUI that collects survey information about the application after the end-user is done.

The client has the following responsibilities:

- When provided with a “Go Code” the Client must contact the Server to request that Application.
- When a “Go Code” is valid, and the Client receives it, the Client must install that application onto the Android device.
- The Client must record all pertinent information about the execution of the application, and transmit the entirety of that information back to the Server for storage.

ADTP Specification

General ADTP Message Format

ADTP messages shall be transmitted *in-band* over a TCP/IP connection to ensure quality of service. The format of ADTP Messages is below

Bytes	**Description**
0-3	**ADTP Command** – This is a standard 32-bit integer specifying the ADTP command this message represents.
4-7	**ADTP Argument 1** – This is a standard 32-bit integer specifying some argument for the ADTP command.
8-11	**ADTP Argument 2** – This is a standard 32-bit integer specifying some argument for the ADTP command.
12-15	**Payload Boundary** – These byes are ALWAYS a 32-bit zero, to serve as a boundary between the control information and the payload information. This is present even if there is no payload. This isn't really needed, but I'm reserving it anyway in case space is needed later for a 3rd argument.
16-??	**Payload Data** – The rest of any ADTP message will be Payload data.

ADTP Client Messages

The following messages are ADTP messages that the client can send to the server, the significance of each message, and other pertinent notes. The numerical number of each command is in parenthesis to the right of the actual name of the command.

HELLO (101)

This is the first command sent by the client in any interaction with the server. It is a greeting that starts the conversation

Parameters:

- Argument 1 – Integer representing the revision of ADTP supported by the client.
- Argument 2 – Integer representing the API level of the Android device currently connected to the end-user's computer.
- Payload – None
- Expected Responses – GO_AHEAD or UNSUPPORTED_VERSION

REQUEST_APK (102)

This is the command sent by the client to transmit a "go code" to request an application from the server's repository.

Parameters:

- Argument 1 – 32-bit zero
- Argument 2 – 32-bit zero
- Payload – alphanumeric go-code
- Expected Responses – TRANSMIT_APK or INVALID_APK

TRANSMIT_LOGFILE (103)

This is the command sent by the client when it is transmitting the log file from the application to the server.

Parameters:

- Argument 1 – The 32-bit unique identifier for the application that was just tested.
- Argument 2 – The return code from the application.
- Payload – The log data from the application.
- Expected Responses – TRANSMIT_SUCCESS or TRANSMIT_ERROR

TRANSMIT_COMMENTS (104)

This is the command sent by the client when it is transmitting comments the user has entered about the application into the client GUI.

Parameters:

- Argument 1 – The 32-bit unique identifier for the application that was just tested.
- Argument 2 – 32-bit zero.
- Payload – The user's comments.
- Expected Responses – TRANSMIT_SUCCESS or TRANSMIT_ERROR

VERIFY_APK (105)

This is the command sent to the server by the client to verify that the APK file has been successfully downloaded.

Parameters:

- Argument 1 – The 32-bit unique identifier for the application
- Argument 2 – 32-bit zero.
- Payload – The MD5 hash of the APK as calculated by the client.
- Expected responses – VALID_APK or INVALID_APK

GOODBYE (106)

This is the command sent to the server to signify the end of an exchange between the server and client. This will cause the server to terminate the TCP/IP connection.

Parameters:

- Argument 1 – 32-bit zero.
- Argument 2 – 32-bit zero.
- Payload – None
- Expected responses – none.

ADTP Server Messages

The following are the specifications for the messages the ADTP server will send to the client.

GO_AHEAD (201)

This message is a reply to the client's HELLO message, stating that the server supports the version of the ADTP protocol that will be used by the client.

Parameters:

- Argument 1 – 32-bit zero.
- Argument 2 – 32-bit zero.
- Payload – None.
- Expected Responses – REQUEST_APK, TRANSMIT_LOGFILE, or TRANSMIT_COMMENTS

TRANSMIT_APK (202)

This message is a reply to the client's REQUEST_APK message. This will include the contents of the APK.

Parameters:

- Argument 1 - The 32-bit unique identifier for the application that is being sent.
- Argument 2 – 32-bit zero.
- Payload – The binary APK file.
- Expected Responses – VERIFY_APK

VALID_APK (203)

This is a reply to the client's VERIFY_APK message that states that the MD5 hash provided by the client matches that of the APK on the server and it is safe to install to the Android device.

Parameters:

- Argument 1 – A 32-bit validation confirmation number, for diagnostic purposes.
- Argument 2 – 32-bit zero.
- Payload –None
- Expected Responses – GOODBYE

INVALID_APK (204)

This can be a reply to the client's VERIFY_APK message or the REQUEST_APK message. Signifying that either and MD5 hash doesn't match or a requested APK does not exist.

Parameters:

- Argument 1 – The integer representing VERIFY_APK or REQUEST_APK
- Argument 2 – 32-bit zero.
- Payload –none
- Expected Responses – GOODBYE

TRANSMIT_SUCCESS (205)
This is a reply to either the client's TRANSMIT_LOGFILE or TRANSMIT_COMMENTS.

Parameters:

- Argument 1 – The integer representing TRANSMIT_LOGFILE or TRANSMIT_COMMENTS
- Argument 2 – 32-bit zero.
- Payload – none.
- Expected Responses – TRANSMIT_LOGFILE, TRANSMIT_COMMENTS, or GOODBYE

TRANSMIT_ERROR (206)
This is a reply to either the client's TRANSMIT_LOGFILE or TRANSMIT_COMMENTS.

Parameters:

- Argument 1 – The integer representing TRANSMIT_LOGFILE or TRANSMIT_COMMENTS.
- Argument 2 – 32-bit error code.
- Payload – Possibly a more descriptive error message, but could also be empty.
- Expected Reponses – GOODBYE

UNSUPPORTED_VERSION (000)
This is a reply to the clients HELLO message that will always signify that the server does not support the version of the ADTP protocol requested by the client.

Parameters:

- Argument 1 – The minimum revision of the protocol required by the server.
- Argument 2 – 32-bit zero.
- Payload – None.
- Expected Responses – GOODBYE

Revision History

Below is the revision history of the ADTP Specification.

- **REVISION 1** August 2010
  - Initial specification.