Remote App Installation

Overview

Francisco Aleixo

Aptoide Internship

August 2016

How It Works

Remote App Installation mainly works via a combination of mDNS for service announcement/discovery and sockets (TCP) for transfering apps MD5 checksum from a smartphone to a TV for consequent download and installation.

In this document this techniques will be briefly explained aswell as some details about its implementation and troubleshooting options for users.

Service Discovery & Sending MD5

Service discovery (on Aptoide Mobile) and service announcement (on AptoideTV) are made using mDNS (Multicast DNS) technology. The heavy work behind this discovery and announcement is mainly made by the JmDNS library[1]. This library is compatible with very popular implementations of mDNS such as Apple Bonjour or Google Cast. The use of this library also means that this implementation is API agnostic.

Service discovery is implemented on RemoteInstallationSenderManager class. This class is responsible for not only service discovery but also for sending MD5 checksum and receiving consequent confirmation. In essence, the order of operations (that will be generally used) are:

1. Discover Aptoide Remote Install services by first acquiring a multicast lock and then resolving the service. Note that this is done using a regular thread with the help of an Handler for listener calls.
2. Sending MD5 checksum to the resolved service using Sockets and receiving confirmation (timeout being 10 seconds). This is done using an AsyncTask.

This class makes use of a listener to send (self-explanatory) notifications about appropriate events:

1. onDiscoveryStarted;
2. onDiscoveryStopped;
3. onAptoideTVServiceLost;
4. onAptoideTVServiceFound;

5. onMd5ChecksumSendSuccess;
6. onMd5ChecksumSendUnsuccess.

Service Announcement & Receiving MD5

Service announcement (on AptoideTV) is implemented using a background Android service. The idea is that this service is always running on background (independent if AptoideTV is currently running) so it’s always listening for remote app installations requests. It is important to note two important situations:

1. On AptoideTV launch the service is restarted if previously running (this can serve as a troubleshooting option in case of an unexpected problem).
2. On Network SSID change the service will be also restarted so it can stop the previous announcement and announce the service on the new network.

In terms of announcing, receiving MD5 and confirming that it has been received, the process if very similar to the previously explained Service Discovery.
In this case, the service announces its service to the network and initiates a ServerSocket that awaits connections and requests.

Here the only event that the listener implements is the following:
1. onMd5AppInstallationRequest - for processing the md5.

When it receives the payload, it first checks to see if the payload is a valid MD5 string. Afterwards, it launches an intent so AptoideTV can receive the request. This intent has the following format:

```
"aptoideinstall://md5sum="+md5sum+"&auto_download=true&remote_install=true"
```

Changes to the AptoideTV intent handlers have been made to accomodate for remote install (hence the remote_install tag).

Settings

The use of this service is optional and can be toggled on and off via the Settings menu (Settings -> Preferences -> Allow remote app install). By default the service is active (on).

Troubleshooting

In case of device discovery problems, based on previously explained topics, the recommended steps (ordered by relevance) are the following:

1. Ensure that your AndroidTV has the AptoideTV app installed.
2. Ensure that both the AndroidTV and your smartphone are connected to the same network.
3. Relaunch the AptoideTV app.
4. Ensure that the network is not blocking IP Multicast packets. If you can use services like Google Cast™ without problems, then this should not be the issue.

Memory allocation studies

Since this process involves a fair bit of threading and networking, a demo project using these classes has been tested for memory leaks. It involves logging the memory heap allocation using Android Studio Monitor.

For service discovery, a DialogFragment with a ListView containing discovered services has been tested for memory leaks using the following order of operation:

1. Launch DialogFragment that will, in turn, start discovering devices and add them to the list view.
2. Dismiss this dialog.
3. Repeat steps 1 and 2 (x10).
4. Trigger manual Garbage Collect.
5. Generate heap allocation report and analyze the results.

The results clearly showed that if appropriately handled, then no leaks will be created. Appropriate handling includes, for example, not forgetting to stop discovery after it has been started (using the defined methods) -- otherwise leaks will occur.