NetInf Streaming Solution User Guide

Al Hinai, Abdullah   Andersson, Conny   Forouzani, Sepehr
Halteh, Faris   Ivanou, Aliaksandr   Karkanis, Iosif
Klingsbo, Lukas   Lan, Fangming   Lång, Magnus
McCain, Daniel Sean   Noorani Subramanian, Varun
Omer, Enghin   Santos Rivera, Juan De Dios

January 16, 2015

Contents

Chapter 1

Android

1.1 Introduction

The User Guide is intended for those who need to deploy the NetInf Streaming solution on both servers and phones. The guide does not explain the technical aspects of the code, there is a separate document for that called the System Description. The User Guide explains how to build and configure the components of the NetInf streaming solution.

1.2 Required Android SDK packages

- Android SDK Tools
- Android SDK Platform-tools v.21
- Android SDK Platform-tools v.20
- Android SDK Build-tools v.21.1.2
- Android 5.0 SDK Platform
- Android Support Repository
- Google Repository

These packages can be found in the Android SDK Manager. For more information how to add SDK package, visit `http://developer.android.com/sdk/installing/adding-packages.html`. Later versions of these packages can be used, but since the version numbers need to be hard-coded in the Gradle build scripts, the build scripts will need to be changed.

1.3 Android phone udev Rules

In order for your SDK to recognize the Android devices in Ubuntu Linux, you have to write the following commands in your terminal:

- wget `https://raw.githubusercontent.com/snowdream/51-android/master/51-android.rules`
- sudo install -m644 51-android.rules /etc/udev/rules.d
- rm 51-android.rules
- sudo service udev restart

For more details, visit: `http://developer.android.com/tools/device.html`

1.4 Building with Gradle

To build the project using Gradle, execute the following command in the terminal while in the root of the android-netinf directory:

```
./gradlew build
```

For more information how to build the project with Gradle, visit: `http://tools.android.com/build/gradleplugin`

1.5 Importing into Android Studio

To import the project into Android Studio, select "Open an existing Android Studio project", and select the root folder of the source distribution. In the dialog that pops up, make sure that the path in the "Gradle project:" text box is the root folder of the source distribution (see Figure 1.1), and click OK.

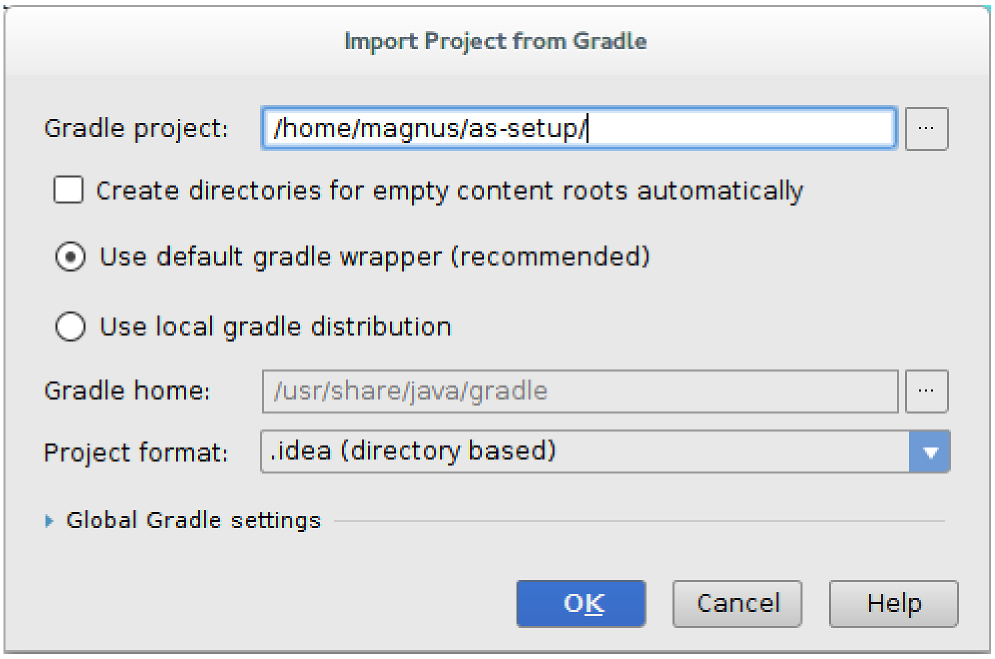
1.6 Running and testing the application on a device

First, you have to connect your Android device to your computer. You can use an "Android Virtual Device (AVD)" to view streams, but you will not be able to stream them.

In Android Studio, press the green arrow or green bug icon on the tool bar and select your device in the dialog that pops up. If you selected the bug icon, the debugger will immediately be attached to the app.

If you are not using Android Studio, you can install the app to the phone directly from Gradle using the following command:

```
./gradlew installDebug
```



Import Project from Gradle

Gradle project: /home/magnus/as-setup/

☐ Create directories for empty content roots automatically

● Use default gradle wrapper (recommended)

○ Use local gradle distribution

Gradle home: /usr/share/java/gradle

Project format: .idea (directory based)

Figure 1.1: The Android Studio import dialog

1.7 Application preferences

The application has a settings page to manage the developer settings. This settings activity can be opened by touching the screen four times using three fingers on the main page.

To change these preferences from the code, go to the "android-service" directory and open the xml folder where you will find the file *preferences.xml*.

The preferences are divided into seven categories. These are:

- Database
- After successful GET
- UDP
- HTTP
- Stream
- Storage
- Settings

1.7.1 Database

Clears the database content from the user device.

1.7.2 After successful GET

Enables or disables caching, the default value set to true.

1.7.3 UDP

Allows the app to discover and connect to NetInf UDP services (IP and Port) in the network. Default value set to automatic. If needed set it to a specific address, it can be changed statically from the user settings page in the app.

1.7.4 HTTP

Allows to discover and connect to NetInf TCP services (IP and Port) in the network. Default value is automatic searching. If needed set it to specific address, it can be changed statically from the user setting page in the app.

1.7.5 Settings

Restores the default preference values.

1.8 Streaming and encoding settings

In the **edu.projectcs.falun.streamer** package within the android-netinf project, users can find the settings that are related to the streaming of videos and the camera preview.

From the *Encoder.java* class, the bit rate, i-frame intervals, estimated frame time for window and the mime type can be changed from the following variables: BIT_RATE, IFRAME_INTERVAL, FRAMETIME_ESTIMATE_WINDOW, and MIME_TYPE.

From the *CameraPreview.java* class, settings such as the maximum width, maximum height and the frames per second (fps) of the video stream can be changed by modifying the following variables: MAX_WIDTH, MAX_HEIGHT and DESIRED_FPS.

1.9 Application Snapshots

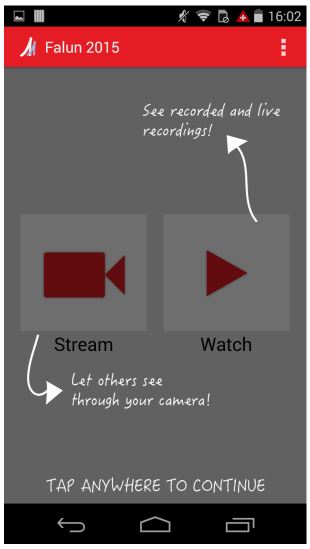

16:02
Falun 2015
See recorded and live recordings!
Stream
Watch
Let others see through your camera!
TAP ANYWHERE TO CONTINUE

Figure 1.2: First time screen

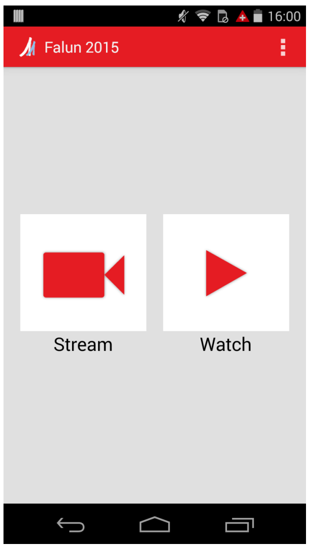


Figure 1.3: Main screen

When first using the application, a small tutorial will be presented. (Figure 1.2).

The main screen, Figure 1.3, contains two buttons: a Stream button, which allows the user to stream video to other users; and a Watch button, which allows the user to show live streams recorded by other users.

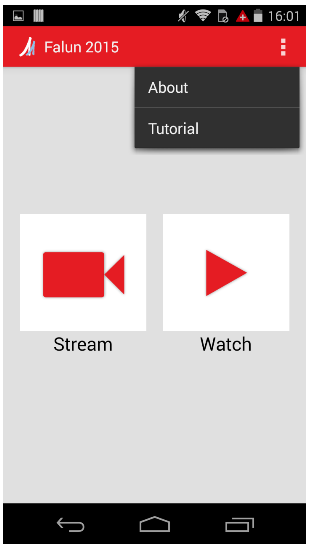


Figure 1.4: Tutorial screen

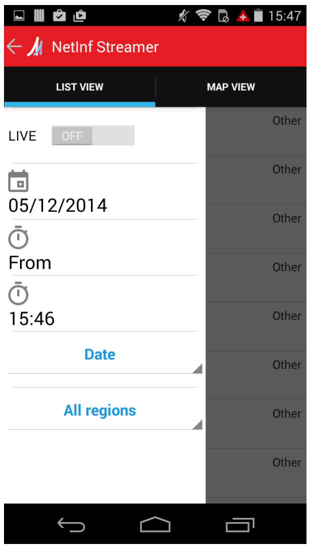


Figure 1.5: Watch screen

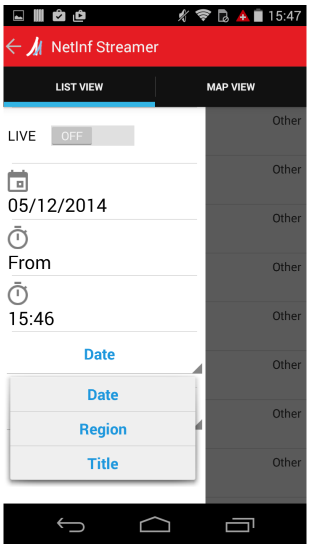


Figure 1.6: Sort options

Users can view the tutorial again by clicking the overflow menu (i.e. the three squares on the right top corner of the main page. (Figure 1.4).

If the user presses the Watch button, they will be presented with the Watch page, which contains two tabs: one is list view and the other is map view. In both these views the user can select between live or non-live streams, as well as the date and time of previously published streams (i.e. non-live). (Figure 1.5).

The streams can be sorted on the date of the recorded stream, the region it was recorded in, and the title of the video. (Figure 1.6) .

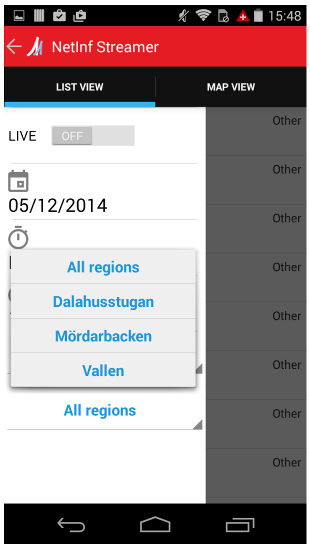


Figure 1.7: Region options

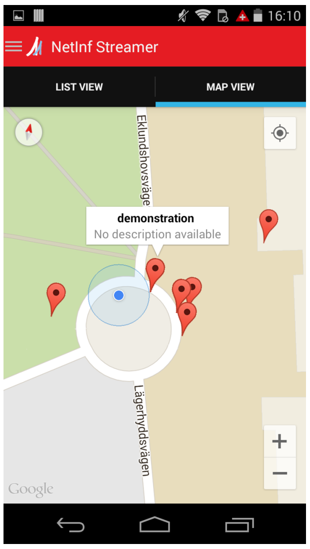


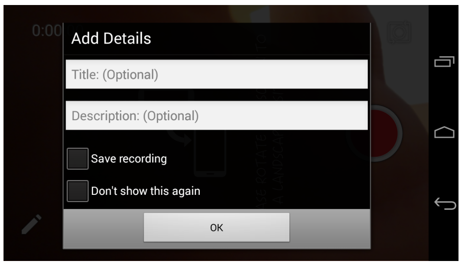
Figure 1.8: Map view

In the list view, there is also the possibility to filter streams by the region they were recorded in. (Figure 1.7).

When the user filters by region, while in the map view, only the streams recorded in that particular region will be shown on the map. (Figure (1.8).

When first entering the Streaming screen, or if the preference is selected, a dialog will appear where the user can enter the title and/or the description of the stream. In the dialog there are also two checkboxes where the user can choose whether or not the stream should be saved on the phone, and whether or not the user wants this dialog to appear automatically every time the Stream page is entered. (Figure(1.9).

To reach the developer settings, touch the screen four times using three fingers on the main page. (Figure (1.10).


Add Details

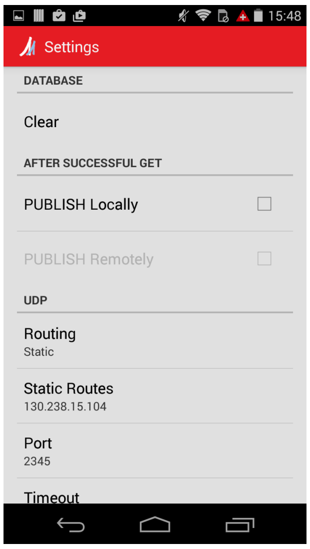
Title: (Optional)

Description: (Optional)

Save recording

Don't show this again

Figure 1.9: Streaming options


Settings

DATABASE

Clear

AFTER SUCCESSFUL GET

PUBLISH Locally

PUBLISH Remotely

UDP

Routing
Static

Static Routes
130.238.15.104

Port
2345

Timeout

Figure 1.10: Settings screen

Chapter 2

Backend

2.1 EENR NetInf Router

The Ericsson Erlang NetInf Router (EENR) is an implementation of a NetInf router. This implementation has been updated with several features and fixes. A few examples of this include a new, static, routing model, using Mnesia to store data, a stand-alone NRS featuring document-database-like searches in metadata, and a statistics collection platform.

2.1.1 EENR NetInf Router dependencies

Erlang/OTP 17+ is needed to build and run this application. g++ and lib-netfilter-queue needs to be installed to be able to build it. Our build tool (erlang.mk) will fetch the rest of the dependencies from remote sources using git and wget, so make sure that you have them installed. The following dependencies are automatically fetched when building the first time:

jiffy Used for JSON parsing

cowboy Used for handling everything regarding the HTTP stack

lager Used for logging

proper Used for property testing

2.1.2 Building the EENR NetInf Router

To build the router with the default settings you simply go to the root folder of the application and execute make, it will fetch all the dependencies and build them and the router. You need internet connectivity the first time you run it as the build tool needs to download the dependencies.

2.1.3 Starting the EENR NetInf Router

To start the router with the default settings run `make start` in the root directory of the source tree. This will start the router using port 8082 for TCP connections and 2345 for UDP connections. If you want to send custom parameters you can use:

```
erl -pa ebin deps/*/ebin -config eenr -s eenr -eenr Par Val
```

Where `Par` is the parameter name and `Val` is the value of that parameter. A full list of parameters that can be set can be seen in src/eenr.app.src. A few useful ones are:

- tcp_port (default value: 8082)
- udp_port (default value: 2345)
- request_timeout (default value: 10000)

2.2 NRS

The NRS is a centralized NetInf database for use by the EENR router. It also allows queries in the metadata of NDOs, similar to a document database. However, it's currently implemented using Mnesia rather than a document database.

2.2.1 NRS dependencies

Erlang/OTP 17+ is needed to build and run this application. Our build tool (erlang.mk) will fetch the rest of the dependencies from remote sources using git and wget, so make sure that you have them installed. The following dependencies are automatically fetched when building the first time:

jiffy Used for JSON parsing

lager Used for logging

proper Used for property testing

ranch Used to accept TCP connections

2.2.2 Building the NRS

To build the NRS with the default settings you simply go to the root folder of the application and execute make, it will fetch all the dependencies and build them and the router. You need internet connectivity the first time you run it as the build tool needs to download the dependencies.

2.2.3 Starting the NRS

To start the NRS with the default settings run: make start. Only one node in the network should run the NRS. If you want to send custom parameters you can use:

```
erl -pa ebin deps/*/ebin -s nrs -nrs Par Val
```

Where `Par` is the parameter name and `Val` is the value of that parameter. The only parameter that the NRS provides is:

- api_port (default value: 8912)

2.3 WallE

WallE is the cleaning robot of our Android streaming solution. Its purpose is to tag streams that have been interrupted by a network outage or application crash, so they will no longer appear as live. It also deletes recorded streams that are shorter than a set length (currently 2 seconds by default), stopping streams from showing up in the history if a user starts and immediately stops them.

2.3.1 WallE dependencies

Erlang/OTP 17+ is needed to build and run this application. Our build tool (erlang.mk) will fetch the rest of the dependencies from remote sources using git and wget, so make sure that you have them installed. The following dependencies are automatically fetched when building the first time:

jiffy Used for JSON parsing

lager Used for logging

2.3.2 Building WallE

To build the NRS with the default settings you simply go to the root folder of the application and execute make, it will fetch all the dependencies and build them and the router. You need internet connectivity the first time you run it as the build tool needs to download the dependencies.

2.3.3 Starting WallE

To start the router with the default settings run: make start. WallE should only run on a node that runs the NRS. If you want to send custom parameters you can use: `erl -pa ebin deps/*/ebin -s walle -Application Par Val` where Par Val is each custom parameter, Par for the parameter name and Val for the value of that parameter. A full list of parameters that can be set can be seen in src/walle.app.src. A few useful ones are:

- nrs_endpoint (default value: localhost, 8912)
- udp_endpoint (no default value)
- dead_search_interval – how often to query for dead streams (default value: milliseconds, 240000)
- search_interval – how often to query for live streams to see if they are dead (default value: milliseconds, 60000)
- minimum_length – the minimum length that a video needs to be to be kept (default value: milliseconds, 2000)