Bachelor’s Thesis

EnergyBox: Tool improvement and GUI

Rihards Polis

LiTH-IDA/ERASMUS-G–15/001–SE
Linköping 2014

Supervisor: Ekhiotz Jon Vergara
Linköping University

Examiner: Simin Nadjm-Tehrani
Linköping University
Title

EnergyBox: Tool improvement and GUI

Abstract

EnergyBox is a parametrised estimation tool that uses packet traces as input to simulate the energy consumption of communication in mobile devices. This tool models the transmission behaviour of a smart phone by analysing a recorded packet trace from the device. The purpose of the thesis is to reimplement the original EnergyBox energy consumption modelling tool. The project aims to develop support for a graphical user interface (GUI) and a code base that is easier to modify and maintain.

The motivation for the reimplementation of the tool is to simplify its usage and to structure the code so that new features can be added. The existing features such as the calculation of total power consumed by the packet trace and the modelling of a device's energy states are reimplemented and new features are developed. Among the new features, a GUI is added to simplify the usage of the application features such as the detection of the recording device's IP address and the ability to alter the configuration parameters used as input to the energy model.

The application is written with a GUI and modularity in mind. This is achieved using Java's proprietary new GUI framework - JavaFX, which supports built-in chart and graph GUI elements, that can be easily integrated and supported. The energy modelling engines follow the semantics of the original implementation and the evaluation shows that the new implementation's results are identical to the original tool in 94.94% of the tested cases.

Keywords

EnergyBox, energy measurement, mobile 3G/WiFi, energy estimation, transmission energy, 3G, WiFi, JavaFX
Abstract

EnergyBox is a parametrised estimation tool that uses packet traces as input to simulate the energy consumption of communication in mobile devices. This tool models the transmission behaviour of a smart phone by analysing a recorded packet trace from the device. The purpose of the thesis is to reimplement the original EnergyBox energy consumption modelling tool. The project aims to develop support for a graphical user interface (GUI) and a code base that is easier to modify and maintain.

The motivation for the reimplementation of the tool is to simplify its usage and to structure the code so that new features can be added. The existing features such as the calculation of total power consumed by the packet trace and the modelling of a device’s energy states are reimplemented and new features are developed. Among the new features, a GUI is added to simplify the usage of the application features such as the detection of the recording device’s IP address and the ability to alter the configuration parameters used as input to the energy model.

The application is written with a GUI and modularity in mind. This is achieved using Java’s proprietary new GUI framework - JavaFX, which supports built-in chart and graph GUI elements, that can be easily integrated and supported. The energy modelling engines follow the semantics of the original implementation and the evaluation shows that the new implementation’s results are identical to the original tool in 94.94% of the tested cases.
# Contents

1 Introduction .......................... 1
   1.1 Context and Motivation .............. 1
   1.2 Problem Definition .................. 2
   1.3 Goals ................................ 3
      1.3.1 Graphical User Interface .......... 3
      1.3.2 No hard coding of inputs .......... 3
      1.3.3 Platform independence .............. 3
      1.3.4 Modular design .................. 3
      1.3.5 Accuracy and correctness .......... 4
   1.4 Methodology .......................... 4
      1.4.1 Approach to implementation ........ 4
      1.4.2 Evaluation ........................ 4
   1.5 Thesis outline ...................... 5

2 Background ........................... 7
   2.1 Wireless Communication Energy Consumption .......... 7
      2.1.1 Energy consumption of 3G .......... 7
      2.1.2 Energy consumption of Wi-Fi ....... 8
   2.2 Original EnergyBox .................... 8
      2.2.1 3G Engine ........................ 8
      2.2.2 Wi-Fi Engine ...................... 9

3 New EnergyBox .......................... 11
   3.1 Design Choices ....................... 11
      3.1.1 Model-View-Controller ............ 12
      3.1.2 JNetPcap ........................ 12
   3.2 Graphical User Interface .............. 12
      3.2.1 Command Line Interface .......... 14
   3.3 Recording Device IP Detection .......... 14
   3.4 Implementation ...................... 15
      3.4.1 Configuration files ............... 15
      3.4.2 Forms and Controllers ............ 16
      3.4.3 JNetPCap Implementation and Packet Class ... 18
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.4 Engines</td>
<td>18</td>
</tr>
<tr>
<td>3.5 Dependencies and Platform Specifics</td>
<td>20</td>
</tr>
<tr>
<td>4 Correspondence Evaluation</td>
<td>21</td>
</tr>
<tr>
<td>4.1 Evaluation Setup</td>
<td>21</td>
</tr>
<tr>
<td>4.1.1 Trace selection</td>
<td>22</td>
</tr>
<tr>
<td>4.2 Evaluation Parameters</td>
<td>22</td>
</tr>
<tr>
<td>4.2.1 3G Parameters</td>
<td>23</td>
</tr>
<tr>
<td>4.2.2 Wi-Fi Parameters</td>
<td>23</td>
</tr>
<tr>
<td>4.3 Evaluation Results</td>
<td>23</td>
</tr>
<tr>
<td>4.3.1 3G Validation</td>
<td>24</td>
</tr>
<tr>
<td>4.3.2 Extended 3G Volume Validation</td>
<td>25</td>
</tr>
<tr>
<td>4.3.3 Wi-Fi Validation</td>
<td>25</td>
</tr>
<tr>
<td>5 Conclusions and Future Work</td>
<td>27</td>
</tr>
<tr>
<td>5.1 Conclusions</td>
<td>27</td>
</tr>
<tr>
<td>5.2 Future Work</td>
<td>28</td>
</tr>
<tr>
<td>5.2.1 Code Base</td>
<td>28</td>
</tr>
<tr>
<td>5.2.2 Optimization</td>
<td>28</td>
</tr>
<tr>
<td>5.2.3 Features</td>
<td>29</td>
</tr>
<tr>
<td>A Form Screenshots</td>
<td>33</td>
</tr>
<tr>
<td>B Class method descriptions</td>
<td>37</td>
</tr>
<tr>
<td>Bibliography</td>
<td>41</td>
</tr>
</tbody>
</table>
This thesis work was performed as a partial fulfilment of a Bachelor degree in Computer Systems at Riga Technical university, and was performed as part of an exchange program at Linköping university, department of Computer and Information Science.

1.1 Context and Motivation

Mobile phones have become an essential part of people’s lives and with the increased popularity of smart phones "any time anywhere" wireless connectivity to the Internet has become widespread.

While mobile device technology has remarkably advanced in areas like display resolution, processing power and wireless connectivity, battery capacity has not had such a leaping advance in comparison. This has resulted in a bottleneck in the quality of experience for the user since sharper screens, growing number of processor intensive applications and more demand for high bandwidth connections to the Internet require more energy consumption. Thus if the user wants to harness the full potential of their mobile device they must recharge it often and for extended periods of time.

With the widespread adoption of third generation (3G) communication technology and increasing availability of Wi-Fi connections having your device be constantly connected to the Internet has become the norm. Since wireless communication accounts for a great portion of the energy consumption in mobile devices, implementing software that performs efficient transmissions with these devices is the key to reducing wasted battery power without restricting usability [1]. Therefore there needs to be an easy to use way to estimate and measure the energy
consumption in mobile devices using this software.

EnergyBox is a parametrised estimation tool that uses packet traces as input to simulate the energy consumption of communication in mobile devices. This tool models the transmission behaviour of a smart phone by analysing a recorded packet trace from the device (see Figure 1.1).

![Figure 1.1: Overview of the original EnergyBox tool [2].](image)

1.2 Problem Definition

The current implementation of EnergyBox is a prototype used in research projects which employs several other tools (e.g., TShark or gnuplot) and lacks a graphical user interface (GUI). In particular, all input is handled through the command line. The way the tool is designed does not provide any interactivity with the user or extended information about the packet transmissions (e.g., protocols, headers, payloads). Therefore it requires opening the packet capture file (PCAP) in another program.

The main usability drawbacks identified are the following:

- **GUI**: EnergyBox is currently exclusively a command line tool which has some advantages (e.g., processing of traces en masse) but greatly limits its usability.

- **No packet inspection**: having no packet inspection within the tool leads to poor workflow since the user must switch back and forth between the tool and an external application such as Wireshark.

- **Hard coded inputs**: the configuration parameters that should be loaded from a file are hard coded. Changing any of these requires changes to the source code and recompilation.

- **Reliance on external applications**: tools that must be set up programmatic-ally in predefined code lines and compiled again for the tool to execute properly hamper usability.

- **Platform independence**: the current EnergyBox tool requires some external applications to be installed in advance, which can be more complicated to set up depending on the operating system (OS). Also if deployment is
required on an OS that has a different file system than what the hard coded paths are set up for, source code revision is inevitable which requires not only tools and applications for running the original EnergyBox but also tools for handling and compiling the new code.

1.3 Goals

In order to address the above mentioned limitations of the current EnergyBox, there are four main goals considered in this project aiming at a new implementation. These must be accomplished after which the degree of accomplishment will be measured by applying methods described in section 1.4 and later in chapter 4. The four goals are now reformulated in terms of requirements specification for the new software.

1.3.1 Graphical User Interface

The technologies used to build it should have the ability to enable a user friendly GUI that offers methods for showing in charts the analysis outcomes and provide easier ways to visualize the data. The visual features of the GUI should be consistent regardless of the OS. The command line functionality must be retained and added to the new implementation.

1.3.2 No hard coding of inputs

The new implementation of the EnergyBox tool needs to provide a method to input all the parameters that the system needs in a simple and coherent way without any modifications to the source code. This includes turning model parameters into input files that can be changed or modified at runtime as well as file paths that are independent from the OS, file system or the location of the application within it.

1.3.3 Platform independence

The system must be reimplemented using a platform independent technology so that EnergyBox could be deployed on a variety of operating systems without changes to the code or recompilation. Ease of deployment must also be taken into consideration, as the set up process for the original EnergyBox tool is overly complicated. The aim would be for the user to be able to install the application easily - within a few minutes and with less steps comparing to the original tool on the same OS.

1.3.4 Modular design

The code base must be written in a way that allows not only the realization of the previously mentioned goals but also the ability to modify and maintain the code. This would make adding new features (such as different energy consumption models or configuration parameters) or updating the code to support newer version of the technologies used to build it.
1.3.5 Accuracy and correctness

The new implementation of EnergyBox must produce the same results given the same inputs comparing to the original tool. The output of the original EnergyBox is deterministic, which means that if the semantics of how the tool and the application work are the same, the results should match in every test case.

1.4 Methodology

The purpose of the project is not to improve the current engine of EnergyBox (i.e., the energy modelling aspect), but rather to improve the implementation of that tool within an application. The metrics for evaluating the success of the project rely on quantifying and measuring validity of the application’s outputs comparing to the original tool as well as assessing the usability of the interface and the application as a whole.

1.4.1 Approach to implementation

This section provides an overview of the main aspects and concerns regarding the approach considered to reimplement EnergyBox. One of the major goals is to have the application work on multiple platforms. This can be achieved by either having multiple versions of the application for every platform or one version that works for all platforms. A variety of concerns arise in the design choices that have a significant impact on the project regardless of the overarching technology or programming language behind it:

- **GUI toolkit/framework**: The toolkit impacts the visual style of the application as well as the architecture of the project, as some frameworks offer the choice of either having a completely programmatic GUI layout or separating the layout and functionality parts.

- **Third party libraries**: they have the potential advantage of providing features that would otherwise require significant effort to develop versus the potential disadvantages of having application dependency issues. This can lead to complicated set up procedures or the possibility that as the other parts of the architecture receive updates the dependency becomes incompatible with the rest of the system.

- **Original design versus new implementation**: as mentioned before the aim of the project is to add features to the system instead of redesigning it, but in some cases an alternate approach may prove to be more suitable to the application as a whole. This must be considered with the imperative that the results of both the original tool and the new implementation must match in every test case.

1.4.2 Evaluation

The evaluation of the implementation will follow the following procedures. The preservation of functionality and correctness of the modelling aspect will be as-
essed using a set of results gathered from inputs (packet traces coupled with their respective parameters) that represent different scenarios for the usage of the application. These inputs, when processed by the application, will yield energy estimations that will be compared to the output of the original tool. Any deviations may indicate discrepancies in the semantics of the new implementation. Validation will be performed on the same traces that were used to test the accuracy of the original tool. Two different approaches of validation testing will be applied: volume testing and diversity testing. Volume testing will take 30-50 different traces without any context and compare the results from the original tool and the revised application. For diversity testing 5-7 scenarios will be defined, where the nature of the packet traces could be drastically different (i.e. downloading a mobile application versus chatting or playing an online video game). 3-5 traces will be chosen for each scenario to compare results.

This evaluation approach can guarantee the credibility of the validation evaluation by providing both statistically adequate and contextual data. Only performing volume testing ignores context, which leaves the possibility that all of the chosen traces are similar thus not guaranteeing the validity of the application’s results if the circumstances were different. Only performing diversity testing does not provide sufficient ground for adequate empirical evidence.

1.5 Thesis outline

The outline of the thesis is as follows:

- **Chapter 1** defines the problem that the project aims to solve and describes the main goals and ideas of the project.

- **Chapter 2** describes the background technology and existing implementation that the project is based on.

- **Chapter 3** outlines the designs for various components of the system based on the goals and ideas from chapter 1. Section 3.4 describes the implementation of those designs as well as the motivation behind the more influential implementation choices.

- **Chapter 4** reviews the evaluation of the project based on the methodology outlined in chapter 1.

- **Chapter 5** reflects on the results and the evidence gathered from chapter 4 as well as describes the future work that could be done regarding the application.
This chapter provides the needed background to understand the theoretical basis of the thesis and the new implementation of the application. It briefly describes the basics of wireless energy consumption as well as how the original tool operates.

2.1 Wireless Communication Energy Consumption

This section provides the basic theoretical background regarding wireless energy consumption at the handset end for both 3G and Wi-Fi wireless communication systems.

2.1.1 Energy consumption of 3G

The energy consumption of 3G communication at the handset level is mostly influenced by the Radio Resource Control (RRC) protocol. Once the user equipment (UE) establishes a connection to the 3G network it switches from the Idle state to the RRC connected state, which has three substates: Paging Channel (PCH), Forward Access Channel (FACH), and Dedicated Channel (DCH). Transitions between these three states are governed by parameters for inactivity timers and buffer thresholds for state demotion and promotion respectively [3].

The PCH state is the one using the lowest power level. In this state no radio resource is yet allocated, but the UE can still check for incoming downlink packets.

FACH. The UE is allocated a default common or shared transport channel in the uplink and continuously monitors the downlink in FACH. Practically, this means that the achieved data rates are low.
**DCH.** A dedicated channel is allocated to the UE for both uplink and downlink. This state has higher speeds than FACH, but also consumes more energy [4].

Demotion, which is a transition to a lower energy consumption state in the UE, occurs when a timeout period for that specific transition has elapsed. There are separate timers for DCH to FACH and FACH to PCH transitions which differ depending on both the network and the operator of that network [5].

Promotions, which are transition to a higher energy state, are handled using a set of Radio Link Control (RLC) protocol buffers in the UE (PCH to FACH and FACH to DCH) separate for uplink and downlink. When filled, it triggers a promotion to a higher state. These buffers are monitored by the operator and thus the threshold values may vary between them.

### 2.1.2 Energy consumption of Wi-Fi

When a device is connected to an access point (AP) and is sending or receiving data, it is considered to be in the Constant Awake Mode (CAM). To improve energy efficiency the IEEE 802.11 standard provides a Power Saving Mode (PSM) that must be enabled by the AP. In this state the UE wakes up after set intervals and checks if the AP is sending beacons that indicate data for the device, that has been buffered while the device slept [6]. Some UE devices also offer a mode where the wake-up time is dynamic and changes depending on heuristics such as the number of packets, traffic inactivity period or screen activity called Adaptive Power Saving Mode [7].

### 2.2 Original EnergyBox

The original EnergyBox is built to be a tool for energy consumption analysis, that models the state transitions of the UE based on packet traces. By using a previously recorded trace along with network and device parameters that correspond to real devices and operators, the tool uses a simplified finite state machines to determine the power state of the handset. This way the tool recreates the states that the handset was in over time and calculates the power the trace consumed.

The tool comes with two modes: 3G and Wi-Fi. Each has its own engine that loops through the packet list and outputs state and power information over time using a state machine with specific transition criteria. Both engines are described in detail below.

#### 2.2.1 3G Engine

The state machine that is used for modelling is simplified by combining Idle and PCH in to one state, as they consume a similar amount of energy compared to FACH and DCH. This leaves the state machine with only 3 states: Idle/PCH, FACH and DCH as seen in Figure 2.1 (a).

The demotions are determined by the difference in time between the current packet and the previous one. If the difference is greater than the inactivity timer
value for that state, a demotion occurs. Two inactivity timers are modelled - DCH to FACH and FACH to PCH. A low activity mechanism is also implemented by tracking the throughput in DCH in a given time window. If the throughput is lower than the threshold (which is a network parameter) a demotion to FACH is triggered.

The promotions depend on the buffer threshold values. EnergyBox simulates the occupancy of the buffers over time. For every packet in the trace, its length is added to the buffer. For each promotion type there are two buffers: one for uplink and one for downlink. When the engine checks the packet for promotions it adds the packet’s length to the buffer to see if the threshold was exceeded. The clearing of the buffer is modelled using a time value that indicates how long it takes for the buffer to empty, which depends on the buffer’s occupancy and the transmission rate. If the time is shorter than the time between packets, the next packet will be added to an empty buffer since the data in the buffer is already sent.

When the engine reaches the last packet, a set of demotion are modelled if the trace ends with the current state as FACH or DCH. This ensures an equal number of promotions and demotions.

### 2.2.2 Wi-Fi Engine

Along with PSM and CAM, the state machine includes two more states that let the engine model energy consumption more accurately. The PSM-TX state models the transmission of packets in PSM and the CAM-H state represents a high energy consumption in the CAM state caused by high throughput. These state and their
relations can be seen in Figure 2.1 (b). It has to be noted that the CAM-H state is implemented externally using Matlab. The state graph produced by the tool which only models two states (PSM and CAM) is saved as a set of time and state values. These values are then fed in to Matlab which processes the graph again and changes the state from CAM to CAM-H in places where CAM-H is suppose to occur.

The promotions from PSM to PSM-TX or CAM are affected by the packets per second count. If the number exceeds a predefined parameter, the state switches to CAM. Otherwise the PSM-TX state is set. The transitions between CAM and CAM-H are dependant on a data rate threshold that, when exceeded, would trigger a promotion. The data rate over a fix period of time is calculated and compared to a threshold. If the data rate is higher than the threshold for the period of time, the state is moved to CAM-H for the period duration.

The demotions are handled in a similar way. If the data rate falls below the threshold, a demotion is triggered from CAM-H to CAM. The same happens if the packets per second count falls below the limit - the state is demoted from CAM to PSM.
This chapter describes the design choices made to accomplish the goals set in chapter 1 as well as the practical implementation of these designs and how they influence the revised application functionally and aesthetically.

3.1 Design Choices

For the current implementation of the application Java is selected as the programming language. This provides the means to achieve the goal of cross platform portability and the standard libraries that are provided in the Java Development Kit (JDK) solve the following problems:

- **Graphical User Interface toolkit with which layout and functionality can be separate**: JavaFX is a set of graphics and media packages for creating cross platform applications with intricate UI elements. It has been a part of the JDK since Java 8 and provides the previously mentioned features [8]. Using the JavaFX markup language (FXML) the layout can be designed independently and later linked to functions written in Java. An alternative to this is Swing, which has been the de facto GUI toolkit for Java, but is set to be replaced by JavaFX in the future [9].

- **Import and export of configuration files**: `util.Properties` is a class from the standard library that represents a persistent set of properties. The Properties can be saved to a file or loaded from a file. Each key and its corresponding value in the property list is a string [10]. This lets the user alter configuration files in a structured way without the need to alter the source code.
3.1.1 Model-View-Controller

Since with JavaFX the graphical representation and functionality of the GUI are separated specifically by different languages, design patterns that leverage this divide work more naturally with this framework.

Model-view-controller (MVC) is an architecture design pattern, that separates the system into three distinct parts in order to facilitate loose coupling and easier reuse of modules [11].

- **Model**: responsible for changing the state of the system. Provides information to the view.
- **View**: a visual representation of the model. Sends user input to the model and reflects changes received. In JavaFX a view is referred to as a form.
- **Controller**: dictates how the view communicates to the model. Allows changing the way a view responds to user input without changing the visual representation.

Even though following the MVC design is not a goal for the new implementation, the inherit separation of visual presentation and functionality that comes from JavaFX, allows more modularity.

3.1.2 JNetPcap

For packet trace file reading and import in Java there are two prominent libraries: JNetPcap and JPcap. Both use the libpcap C/C++ library for network packet capture. JNetPcap is used in this implementation due to its more detailed documentation.

3.2 Graphical User Interface

With the need to compare multiple packet traces or the same trace with different configurations, a two part GUI design was devised: the main form, where the user can select the trace as well as the configuration files (see Figure 3.1), and a second form for results, which would display the results provided by the engine (see Figure 3.2).

The results form is split into three tabs to make the data visualization easier to view and navigate. Each tab provides elements relevant to its context.

- **Packet Trace**: Figure 3.2a shows the design for the Packet trace tab. This tab shows information regarding packets such as the pie chart of the distribution between uplink (UL) and downlink (DL) packet count, the throughput chart, and details about every packet individually (such as time, length, protocol, etc.) in the table of packets. None of the information in this tab uses data from modelled states.
3.2 Graphical User Interface

**Figure 3.1:** Design for the main menu.

**Figure 3.2:** Design for the (a) Packet trace, (b) State, and (c) Power tabs for the results form.

- **States:** Figure 3.2b shows the design for the States tab. This tab focuses on states and their distribution over time. Along with the graph of states over time it has the packet graph from the previous tab for easier use as well as the pie chart of the distribution of time between states.

- **Power:** Figure 3.2c shows the design for the Power tab. The main parts of this tab are the power chart that displays the power consumed over time and the statistics table that contains general analytical statistics such as the total power consumed during the trace.

Independently from the tabs a field on the top right corner shows information about the inputs (such as the name of the trace file, network configuration file, etc.). The menu bar also provides an option to export the power chart to a Comma Separated Values (CSV) file.
3.2.1 Command Line Interface

The Command Line Interface (CLI) is designed to execute the same methods as the form controllers, without the use of the GUI. This means that the inputs are all either received as command line parameters when the application is called or the user inputs them one by one with a back-and-forth dialogue.

Similarly to the usage of the GUI, the CLI of the revised application requires three main inputs: the packet trace, network configuration, and device configuration files. The CLI would take these inputs and process them the same way as the GUI. The user can specify an optional parameter to output the states to a file along with receiving the total energy consumption in Joules.

3.3 Recording Device IP Detection

When a packet trace is selected, in the original EnergyBox, the user needed to input the IP of the device that performed the trace recording, so that the engine can differentiate the uplink and downlink packets. To streamline the use of the application, a prediction system was designed to help detect the IP automatically based on a set of rules to select the source IP (the recording device’s IP) out of the IP addresses present in the packet trace. These rules are based on the assumption that at least one of the IPs in the trace belongs to the recording device (a smartphone). The following rules are considered for identifying the source IP of the device, which are executed in order:

- **Domain Name System (DNS) packets** – the device is not hosting a DNS server so any requests that are received with this protocol are going to have the source IP as the device’s IP.

- **Hypertext Transfer Protocol (HTTP) packets** – the device is not hosting a web server so any requests that are received with this protocol are going to have the source IP as the device’s IP.

- **IP address occurrence frequency** – this assumes that the recording device will communicate with more than one other location, which means that from those two conversations the recording device’s IP is the one used most frequently. Some overhearing is possible when it comes to Wi-Fi and protocols like the Address Resolution Protocol (ARP).

- **First mentioned** – the worst case scenario is if there are no DNS or HTTP packets and the packets go back and forth from the recording device to only one external IP address, which means that the occurrence frequency is going to be the same for both IP addresses. In this case the system takes the first mentioned IP. Most of the times that is the source IP of the first packet.

If the previously mentioned system does not provide the correct recording device IP address the user can repeat the modelling process with a manually provided recording device IP address as this is an optional feature and is not evaluated in this thesis.
3.4 Implementation

This section explains the realisation of the design based on decisions described in the previous section. It also provides the motivation behind implementation decisions that significantly influence the main goals of the project as well as architectural alternatives to those decisions. It describes the way the forms are structured and later covers how the functionality of those forms is implemented.

Figure 3.3 shows the connections of the modules and the application data flow. The configuration files and the packet trace are selected using the main form. Its controller determines which engine (3G or WiFi) to call and provides the parsed packet trace and the source IP address to it. The engine provides the modelled data (states and power) to the results form controller which either displays them or outputs them to a file. All of the modules are described below.

![Diagram of application design](image)

**Figure 3.3: Planned application design.**

3.4.1 Configuration files

The import and export of the property files is handled using the built-in class `util.Properties`. The files consist of lines following the notation:

```
PROPERTY_NAME=<value>
```

Every line is a new property (except lines that start with # indicating a comment line) and all of the values are imported as Strings. The properties in the default configuration files match the original implementation's parameters as described in earlier work [2].

The first property, i.e. type, is the same for all of the configuration files. This property indicates what kind of configuration file it is and is used to match the property file with its appropriate properties class.
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>Associated with the Properties3G class, linked to the 3G engine type</td>
</tr>
<tr>
<td>Device3G</td>
<td>Associated with the PropertiesDevice3G class</td>
</tr>
<tr>
<td>Wifi</td>
<td>Associated with the PropertiesWifi class, linked to the Wi-Fi engine type</td>
</tr>
<tr>
<td>DeviceWifi</td>
<td>Associated with the PropertiesDeviceWifi class</td>
</tr>
</tbody>
</table>

Table 3.1: Configuration types and their descriptions

3.4.2 Forms and Controllers

JavaFX provides a way to separate the layout of a GUI from its functionality by using FXML, which is a declarative XML-based mark-up language used to define the layout elements for the form. The functionality is implemented using event triggers that are linked to the FXML elements. This way the layout and the functionality can be systematically separated. Having the layout be in its own FXML file also provides the possibility to replace it with a different layout without altering the Java code. It also allows linking a Cascading Style Sheet (CSS) to the FXML mark up to change the form's appearance, which furthers the goal of the code base being easy to modify and maintain.

The Main Form seen in Figure 3.4 consists of the FXML file and a Java source file that contains the MainFormController class. Each of the three input buttons opens a JavaFX FileChooser window which is specific to each platform. The "Source IP" field overrides the automatic IP address detection and lets the user input it manually. After choosing the appropriate input files, the "EnergyBox" button can be used to open a Results Form.

Figure 3.4: Implementation of the Main Form in Windows 8.
The Main form controller contains the code for the dependency resolution for JNetPcap, its implementation, as well as the recording device IP address detection (explained later in section 3.3).

The Results Form. To implement an easy way to compare different trace and configuration combinations the results are displayed on a separate window. This way the user can have multiple Results form windows side by side for comparison. These windows are generated based on the configuration files chosen in the Main form. There are two architecturally different ways to achieve this:

- A single form and controller that has elements for every type of configuration, be it 3G, Wi-Fi or otherwise. The elements are marked with tags that dictate with which engine the element would be drawn. This makes the code base simple at first but has the potential for the Results Form to get cluttered and slow.

- Separate forms for every engine, that contain the elements for only the specific configuration. This means that the form is chosen before drawing and all of its elements are displayed which requires less computation but also creates redundancy as there are elements that must be displayed for every kind of configuration (such as the packet table or the packet chart). These elements would have to be in every form separately.

The current version works using the latter option since it is less time consuming to implement and does not cause complications with a low number of result form types. This is shown in Figure 3.5.

Which form gets displayed is dependent on the type of network configuration file provided in the Main Form. Each of the Results form controllers has a custom initiation method that passes the appropriate engine object to the controller. Separating the controllers allows the object that is passed to be the specific en-
engine class instead of a superclass that covers every engine type. The input configuration types are linked with their appropriate forms, controllers and engine through a `switch` statement within the Main form controller.

Both forms have the option to export the state graph to a CSV file, which is structured the same way as the original tool’s CSV export file to make it easier to compare both results in chapter 4.

### 3.4.3 JNetPCap Implementation and Packet Class

The packet trace import is implemented within the MainFormController classes `handleButtonAction` method. The full list of the classes methods can be found in appendix A. The JNetPcap’s Pcap object is created and its `openOffline` and `loop` methods open and loop through the previously selected packet trace file. During this loop a list gets populated with instances of the Packet class. Packet is a custom class that contains information about a single packet in the trace. The constructor for this class takes values from the JNetPcap header class objects that represent the headers of the current packet in the loop. This custom class was used to make it easier to bridge the gap between JNetPcap and JavaFX, while still keeping JNetPcap code easily replaceable. The Packet class uses a Façade design pattern [11] to decouple the packet capture import system from the rest of the modules. All of the Packet class fields are wrapped in a property type, which adds additional usability if the data is displayed using JavaFX elements.

The loop also identifies indicators for the recording device IP (described in the section below), that are later processed after the loop.

### 3.4.4 Engines

The current version of the application includes two types of engines: `Engine3G` and `EngineWifi` (see Figure 3.6). Both are written to be semantically identical to the original tool’s implementation.

To facilitate maintainability the engine classes were designed to be extensions of a parent class, that includes the methods that apply for every engine type. The full list of the classes methods can be found in Table B.3. The classes that extend the Engine class are required to override the `modelStates` and `calculatePower` methods. They must also have their own constructor that has the appropriate `properties.device` and `properties.network` classes as parameters. Figure 3.6 shows this relationship between the engine superclass, its subclasses and their respective property classes.

Comparing to the original implementation, there are two major differences in terms of semantics.

1. **Drawing graphs**: Since the original tool’s engines output plot data and use it for calculation, differences in the way the new application draws graphs introduce differences in calculation. `Gnuplot`, the original tool’s command-line driven graphing utility, could only plot continuous functions, thus when a state transition occurs the two plot points that are required to plot the discrete transition
3.4 Implementation

from one state to the other could not have the same time value. The original tool works around this problem by using a time value called shortT which is equal to one microsecond (see Figure 3.7). This value is either added to the second point’s time value \( t(P_{i+1}) \) for demotions or subtracted from the first point when it comes to promotions. The reimplementation allows these points to be drawn without changing the time values, thus creating small discrepancies in power calculation.

This would not change anything if the points were used only for display as the difference is so minuscule that a line between the two points would still seem vertical, but for the original tool the total power value is also calculated using the same points. The power is calculated taking the time difference between two points and multiplying it by the power value of the latter point’s state. The same power calculation algorithm was used in the new application.

**Figure 3.7:** The difference in point placement upon state promotion/demotion.

2. 3G state modelling algorithm: The original tool’s algorithm is written assuming that the packet trace is recorded in the context of some given network parameters. The packets recorded in a real trace experience different delays due to the RRC protocol (e.g., state transitions). Thus, when a packet trace is altered (e.g., to isolate some transitions of interest) or the configuration parameters do not match the ones with which the trace was recorded, the behaviour can be unpredictable.

The revised tool handles the transitions in a way that is more consistent for the rare cases where this occurs, by checking whether the packet trace is plausible.
3.5 Dependencies and Platform Specifics

There are two things that bring their own dependencies to the project: Java and JNetPcap.

- **Java.** To run the application the system needs to have a Java Virtual Machine (JVM) set up. There are a number of them available for multiple platforms and the installation process varies between the JVM and platform.

- **JNetPcap.** The set up for the current version differs depending on the platform:
  - **Windows.** The WinPcap packet must be installed and the jnetpcap.dll file must be within one of the folders in the JVM's PATH variable.
  - **Linux.** The libpcap-dev package must be installed and the jnetpcap.so file must be within one of the folders in the JVM's CLASSPATH variable.
  - **OS X.** Officially JNetPcap 1.4 does not support OS X, but the library is still in active development and unofficial patches are available to extend its functionality. OS X support is not achieved in this thesis.

Since the variable differs between platforms, operating system detection is required. This is accomplished with OSTools, which is a custom class with static methods. In the MainFormController's `initialise` method the `getOS` and `addDirectory` methods from OSTools are called to detect the platform in which the application is running and to add the JAR files location to the PATH or CLASSPATH variable, depending on the platform. This is done so that the user would not have to manually copy the DLL or SO file to a location that the JVM recognises (such as Windows/system32 for Windows or /usr/lib/ for Linux).
This chapter describes the setup of the evaluation procedure to assess the accuracy of the IP detection feature and to determine the application’s power estimations’ correspondence to the original tool. That is, how well the results match between the implementations running the same test data. It also provides analysis of the results.

4.1 Evaluation Setup

The results for the original EnergyBox tool were compared to physical measurements that were taking place during the recording of the packet trace. The evaluation of the original tool showed it to be 98% accurate compared to physically measured energy [2].

The engines in the new tool were designed to be semantically as similar to the original tool’s implementation as possible, while still addressing the issues described in 3.4.4. This means that with the same inputs (network and device configurations and the same trace) both should produce very similar results.

Two metrics are chosen to compare the results between the two implementation for a number of packet traces:

- **State graph.** Both the tool and the application have the ability to export the state graph in time. The CSV file consists of state and time pairs.

- **Total power in Joules.** Both the tool and the application calculate the total power consumed over the whole time of the trace the same way, which means that the only differentiating factor could be the fact that the power is calculated by the state graph points which differ between the two versions.
• **Correspondence.** In this case accuracy is calculated for the total power in Joules using the following formula:

\[
\text{Correspondence} = 1 - \frac{\text{OriginalPower} - \text{NewPower}}{\text{OriginalPower}}
\]

where \(\text{OriginalPower}\) is the total power in Joules calculated by the original tool and \(\text{NewPower}\) is the total power calculated by the new EnergyBox.

The evaluation procedure is done in five steps for both engines:

1. **Determine recording device IP.** Every input trace is analysed to determine manually the recording device IP address and this is used as a baseline to assess the rule-based detection.

2. **Run the packet traces.** The chosen traces are run with the original tool using the previously obtained IP addresses. The same is done for the application, but it is allowed to detect the IP addresses automatically.

3. **Check detected IP addresses.** The detected recording device IP addresses is compared to the ones determined in step 1. If the IP addresses match, proceed to next step.

4. **Compare the state graph points and total power.** The state graph CSV outputs are compared between the tool and the application.

5. **Compare total power.** The total power value is compared between the tool and the application.

### 4.1.1 Trace selection

For primary correspondence testing the chosen traces are the same ones used in the original tool’s accuracy evaluation, which were recorded during specific use cases like using the Youtube Android application or during a Skype call. Each trace is 5 minutes long.

Additional testing was performed on the 3G engine specifically using a greater number of traces to further test the correspondence. The traces that were chosen vary in size, time between the first and last packet, and the number of packets. The type and frequency of protocols also differs. 10 out of the 30 traces use different device power configuration parameters which diversifies the test cases even more. All of the traces are from real device network communication using messager applications such as Gtalk, Viber and Skype. These traces have been studied using the original EnergyBox for other published studies [12].

### 4.2 Evaluation Parameters

This section provides the parameters used for the correspondence evaluation. Every trace’s corresponding parameters were matched in both the original tool and the new implementation.
4.2.1 3G Parameters

Testing for the 3G engines is done using the parameters values in Tables 4.1 and 4.2, which are either provided in the configuration files for the application or changed in the code for the original tool. The network properties correspond with the TeliaSonera network and the device parameters model the Ericsson F337 broadband module (as described in [2]).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCH_FACH_INACTIVITY_TIME</td>
<td>4100</td>
<td>ms</td>
</tr>
<tr>
<td>FACH_IDLE_INACTIVITY_TIME</td>
<td>5600</td>
<td>ms</td>
</tr>
<tr>
<td>DCH_LOW_ACTIVITY_TIME</td>
<td>4000</td>
<td>ms</td>
</tr>
<tr>
<td>DATA_THRESHOLD</td>
<td>1000</td>
<td>bytes</td>
</tr>
<tr>
<td>UPLINK_BUFFER_IDLE_TO_FACH_OR_DCH</td>
<td>1000</td>
<td>bytes</td>
</tr>
<tr>
<td>DOWNLINK_BUFFER_IDLE_TO_FACH_OR_DCH</td>
<td>515</td>
<td>bytes</td>
</tr>
<tr>
<td>UPLINK_BUFFER_FACH_TO_DCH</td>
<td>294</td>
<td>bytes</td>
</tr>
<tr>
<td>DOWNLINK_BUFFER_FACH_TO_DCH</td>
<td>515</td>
<td>bytes</td>
</tr>
<tr>
<td>UPLINK_BUFFER_EMPTY_TIME</td>
<td>1.2</td>
<td>ms</td>
</tr>
<tr>
<td>DOWNLINK_BUFFER_EMPTY_TIME</td>
<td>30</td>
<td>ms</td>
</tr>
<tr>
<td>IDLE_TO_FACH_TRANSITION_TIME</td>
<td>430</td>
<td>ms</td>
</tr>
<tr>
<td>IDLE_TO_DCH_TRANSITION_TIME</td>
<td>1700</td>
<td>ms</td>
</tr>
<tr>
<td>FACH_TO_DCH_TRANSITION_TIME</td>
<td>650</td>
<td>ms</td>
</tr>
<tr>
<td>DCH_TO_FACH_TRANSITION_TIME</td>
<td>700</td>
<td>ms</td>
</tr>
<tr>
<td>FACH_TO_IDLE_TRANSITION_TIME</td>
<td>300</td>
<td>ms</td>
</tr>
</tbody>
</table>

*Table 4.1: Network configuration parameters for 3G testing*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>POWER_IN_IDLE</td>
<td>0.2</td>
<td>W</td>
</tr>
<tr>
<td>POWER_IN_FACH</td>
<td>0.5</td>
<td>W</td>
</tr>
<tr>
<td>POWER_IN_DCH</td>
<td>1.3</td>
<td>W</td>
</tr>
</tbody>
</table>

*Table 4.2: Device configuration parameters for 3G testing*

4.2.2 Wi-Fi Parameters

Testing for the Wi-Fi engines is done using the parameters values in Tables 4.3 and 4.4, which are either provided in the configuration files for the application or changed in the code for the original tool, which also includes changing values in the Matlab scripts responsible for modelling CAM-H. The device parameters correspond to a Samsung Galaxy SII.

4.3 Evaluation Results

This section provides analysis of the results gathered following the procedures described in the previous section.
### Correspondence Evaluation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSM_TO_CAM_THRESHOLD</td>
<td>1</td>
<td>packets/s</td>
</tr>
<tr>
<td>CAM_PSM_INACTIVITY_TIME</td>
<td>220</td>
<td>ms</td>
</tr>
<tr>
<td>CAM_TIME_WINDOW</td>
<td>50</td>
<td>ms</td>
</tr>
<tr>
<td>WINDOW_DATA_RATE_THRESHOLD</td>
<td>3000</td>
<td>bytes</td>
</tr>
</tbody>
</table>

*Table 4.3: Network configuration parameters for Wi-Fi testing*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>POWER_IN_PSM</td>
<td>0.03</td>
<td>W</td>
</tr>
<tr>
<td>POWER_IN_CAM</td>
<td>0.25</td>
<td>W</td>
</tr>
<tr>
<td>POWER_IN_CAMH</td>
<td>0.45</td>
<td>W</td>
</tr>
</tbody>
</table>

*Table 4.4: Device configuration parameters for Wi-Fi testing*

#### 4.3.1 3G Validation

3G validation focuses on the dataset of original traces captured from devices in realistic conditions. Figure 4.1 shows the total energy consumption measured with both the original tool and the reimplemented application. Between all of the traces tested the average correspondence of the new EnergyBox is 99.94% comparing to the original tool.

![Energy consumption comparison](image.png)

*Figure 4.1: Total energy consumption for 3G measured with the original tool and the new EnergyBox.*

The discrepancy is caused by the differences in the new 3G Engine implementation discussed in 3.4.4. The difference can be seen in Figure 4.2, where the promotion from PCH to DCH is handled differently by the New EnergyBox, because the packet following the start of the promotion arrives before the transition is supposed to end. This situation is handled as an exception by the new EnergyBox.

Disregarding the specific case, discrepancy in correspondence ranged from $-2 \times 10^{-7}$ to $2 \times 10^{-7}$ which is caused by the difference in rounding.
4.3 Evaluation Results

4.3.2 Extended 3G Volume Validation

Since the 3G engine is more complicated from a modelling stand point extended 3G testing with a larger number of traces has been added.

Figure 4.3 shows the correspondence difference of every trace in percent. Deviations are caused by the rounding accuracy. The original tool rounds to 4 decimal places where as the new implementation takes 14. This also explain why the deviations are both positive and negative. It must be noted that the highest deviation is still only \(4.9 \times 10^{-7}\).

4.3.3 Wi-Fi Validation

Fig. 4.4 shows the total energy consumption measured with both the original tool and the reimplemented application. The average correspondence comparing to the original tool over all of the traces is 99.97%. The offset is caused by the way CAM-H is modelled by the original tool (see 2.2.2). This is further illustrated by
Figure 4.5 which shows the deviation of the total power calculated using only two states between the two implementations. The largest deviation is only $1.5 \times 10^{-7}$ which is caused by the same rounding difference as for the 3G traces.

**Figure 4.4:** Total energy consumption for Wi-Fi measured with the original tool and the new EnergyBox.

**Figure 4.5:** Deviation of total power using two states in Wi-Fi.
Conclusions and Future Work

This chapter summarizes on the gathered results as well as describes the future work that could be done regarding the new application.

5.1 Conclusions

The main goal of the project has been to reimplement the EnergyBox tool in a modifiable, maintainable way that provides a graphical user interface while still estimating the same power values as the original tool.

- **Hard coded inputs.** The new implementation manages the inputs for the different engines using a system that does not require changes to the code for each new setup.

- **Modular design and platform independence.** The tool is implemented in Java which allows the structure of the class hierarchy to be easily modifiable, which in turn helps with extending the application with new engine types. It also ensures seamless cross platform support.

- **GUI.** JavaFX was used to implement a graphical user interface that looks and functions the same on every platform supported by JavaFX. This enables uniform way of analysing the results regardless of the platform.

- **Correspondence and correctness.** The correspondence of the new EnergyBox is maintained by implementing the modelling engines semantically as close as possible to the original tool’s engines, but also correcting some state demotion/promotion detection aspects that were discovered to be not desirable in the original implementation. For 3G these defects proved to influence the results by 0,06% leading to 99,94% correspondence comparing
to the original tool. For the Wi-Fi engine the defects influenced the results by 0.082%, so overall the application's performance has a more than 99.9% similarity to the original implementation.

5.2 Future Work

The work may continue to develop new configuration types and engines to extend the functionality of EnergyBox. Support for new technologies (such as Bluetooth or LTE) can be implemented by creating new forms and property classes.

5.2.1 Code Base

With these new engines, classes, and forms a new system will need to be implemented to structure them.

- **Results Forms.** The previously mentioned unified results form system may be the better option, where there is only one FXML file and one controller for the results form and the elements are displayed based on tags associated to them and their appropriate engines.

- **Engine.** A system to model an engine without explicitly coding it may be the subject of future development. Designs for a tool that lets the user create a state machine for an engine visually by connecting states with transitions using a GUI and a mouse may have a potential.

- **Links between configuration types and property classes.** In the current version the links are managed using switch statements, where for every configuration type there is a property class assigned to it. A more scalable option would be to implement some sort of property class generation system, a stored register of configuration type and property class pairs, or a universal approach to configuration files.

- **OS X support.** Future versions are planned to support OS X either using a custom JNetPcap compilation or an alternative way to import the packet capture file such as Tshark.

5.2.2 Optimization

The focus of the thesis was functionality of the system. Now that it is achieved, further improvements can be made not only by extending the feature set but also by optimizing the current features. Many specific changes may increase performance of the system, but most of these changes require benchmarking and testing to determine whether they would be viable to implement. Some may increase performance but compromise the modifiability and maintainability of the code base.

Some of these changes are:

- **Alternative data structures.** Storing packets and modelling data may be done in a more efficient way using other data structures provided by the
5.2 Future Work

- **Multithreading.** Some function may be offloaded to separate threads or handled concurrently to both increase the calculation speed as well as the responsiveness of the GUI.

- **Graph rendering.** With a large number of points being drawn in the charts, some are close enough to where they merge, given the limitations of the screen and window size. These points are visually redundant and could be skipped.

### 5.2.3 Features

Many of the UI features that slightly increase usability or extend functionality that were classified as *optional* were not implemented. These can be implemented some time in the future.

One of the more prominent features that may be implemented in the future is the support for EnergyBox as a web application. JavaFX provides ways to create a port that uses the same code base, looks and functions the same way as the current application. This would allow the user to access the features available in the current version through a web browser.

Ongoing development will continue, extending the open source code base available at https://github.com/rtslab/EnergyBox.
Appendix
The screen shots below are from the results form. They display how the graphical user interface is structured and what information is available in all three tabs.

All of the information displayed in the screen shots is taken from results provided by the rewritten engines.
Figure A.1: Implementation of the Packet trace tab for the results form in Windows 8.
Figure A.2: Implementation of the State tab for the results form in Windows 8.
Figure A.3: Implementation of the Power tab for the results form in Windows 8.
The tables seen in this appendix reflect the state of the class structure as it currently is. Every table provides a list of methods and their descriptions for every class or set of classes that have similar functions.
<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Name and Description</th>
</tr>
</thead>
</table>
| public void      | initialize(URL url, ResourceBundle rb)  
Identifies the OS to change the JVM PATH variable, which is different for every OS, to include the folder that the JAR file is in. |
| public void      | handleButtonAction(ActionEvent event)  
Called when the "EnergyBox" button is pressed. Handles the JNetPcap loop, that imports the packet list and checks for signs of the recording device's IP address. |
| public void      | handleDeviceButton(ActionEvent event)  
Called when the "Device Model" button is pressed. Opens a FileChooser, imports a property file, and chooses the appropriate properties.device class for the file. |
| public void      | handleNetworkButton(ActionEvent event)  
Called when the "Network Model" button is pressed. Opens a FileChooser, imports a property file, and chooses the appropriate properties.network class for the file. |
| public void      | handleTraceButton(ActionEvent event)  
Called when the "Trace" button is pressed. Opens a FileChooser and saves the trace files path. |
| public Stage     | showResultsForm3G(Engine3G engine)  
Opens a ResultsForm3G window and passes the Engine3G object to the controller. |
| public Stage     | showResultsFormWifi(EngineWifi engine)  
Opens a ResultsFormWifi window and passes the EngineWifi object to the controller. |
| public Properties| pathToProperties(String path)  
Takes the path to a property file and returns a Properties object. |

*Table B.1: MainFormController method summary*
<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Name and Description</th>
</tr>
</thead>
</table>
| public void       | `initialize(URL url, ResourceBundle rb)`
|                   | Identifies the OS to change the JVM PATH variable, which is different for every OS, to include the folder that the JAR file is in. |
| private void      | `handleButtonAction(ActionEvent event)`
|                   | Called when the "EnergyBox" button is pressed. Handles the JNetPcap loop, that imports the packet list and checks for signs of the recording device’s IP address. |
| public void       | `handleDeviceButton(ActionEvent event)`
|                   | Called when the "Device Model" button is pressed. Opens a FileChooser, imports a property file, and chooses the appropriate properties.device class for the file. |
| public void       | `handleNetworkButton(ActionEvent event)`
|                   | Called when the "Network Model" button is pressed. Opens a FileChooser, imports a property file, and chooses the appropriate properties.network class for the file. |
| public void       | `handleTraceButton(ActionEvent event)`
|                   | Called when the "Trace" button is pressed. Opens a FileChooser and saves the trace files path. |

*Table B.2: ResultsForm3GController and ResultsFormWifiController method summary*
<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Name and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ObservableList <code>&lt;Packet&gt;</code></td>
<td>sortUplinkDownlink(ObservableList&lt;Packet&gt;, String sourceIP) Marks packets as either Uplink or Downlink.</td>
</tr>
<tr>
<td>public XYChart.Series <code>&lt;Long, Long&gt;</code></td>
<td>getUplinkThroughput(double chunkSize) Calculates the uplink throughput for the length of the trace using the given chunk size.</td>
</tr>
<tr>
<td>public XYChart.Series <code>&lt;Long, Long&gt;</code></td>
<td>getDownlinkThroughput(double chunkSize) Calculates the downlink throughput for the length of the trace using the given chunk size.</td>
</tr>
<tr>
<td>abstract XYChart.Series <code>&lt;Double, Integer&gt;</code></td>
<td>modelStates() Loops through the packetList and calculates the state chart, packet chart, distribution pie chart. Implemented for every engine type separately.</td>
</tr>
<tr>
<td>abstract void</td>
<td>calculatePower() Calculates the time spent in each state as well as the total power used from the power chart.</td>
</tr>
<tr>
<td>public void</td>
<td>packetChartEntry(Packet packet) Utility method for populating the packet chart. Used in the modelStates() implementations.</td>
</tr>
<tr>
<td>public void</td>
<td>drawState(Long time, int state) Utility method to populate the power chart. Used in the modelStates() implementations.</td>
</tr>
</tbody>
</table>

*Table B.3: Engine method summary (not including getters)*


Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagens hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Rihards Polis