

HERIOT-WATT UNIVERSITY

Master Thesis

COMPUTER NETWORK ANALYSIS AND PERFORMANCE MEASUREMENT

Supervisor:

Idris Skloul Ibrahim

Second Reader:

Jessica Chen-Burger

Author:

Sidi Sun

*A thesis submitted in fulfilment of the requirements
for the degree of MSc. Data Science
in the*

School of Mathematical and Computer Sciences

August 2018



Abstract

NS2 (Network Simulator 2) is a cross-platform discrete-event computer network simulators that has been primarily used in research and teaching. NS2 provides support for simulation of TCP, this includes multicast protocols such as AODV, DSDV and etc. over wired and wireless network.

Despite the rich features NS2 provides, its trace file tends to be very large, complex and non-human-readable. Moreover, there are not many good tools available to help analyze the trace file. Most of the available tools are poorly designed and have flaws, such as low performance, low portability and lack of graphical user interface.

In this master's thesis, a cross-platform GUI application that uses multithreading technology to help effectively analyze multiple large data files is designed and implemented. The aim of this application is to analyze NS2 trace files (old and new format) effectively in multiple platforms: Windows, Mac and Linux. The application is also designed as a teaching tool for third year network students.

NOTE:

The following are a few chapters to show you how to download and use this tool cross different platforms (e.g. Window, MAC and Linus)

Literature Review

This chapter starts with an example of the trace file, illustrates the defect of existed analysis tools. Finally, there will be a discussion on the technologies used for this project.

2.1 Trace File Format

This application is designed for analyzing both new and old format of trace file over wired and wireless network. The trace files for wire and wireless network are the same, however, the old format trace file is different from the new one. One of the objectives to distinguish the trace file format, and analyze both formats of the trace files.

2.1.1 Old Trace File Format

Figure 2.1 shows an example of the old format trace file. This line represents the following information:

1. “s” stands for a send event
2. The send event occurs at 21.500275000 second
3. This packet is sent from node “_0_”
4. This event happens on the “_MAC_” layer
5. The packet id is 0
6. The packet type is “AODV”
7. The packet size is “106” bytes
8. The rest of the line are the MAC address and IP address of the source and destination, which are not essential for calculating the analysis result

```
s 21.500275000 _0_ MAC --- 0 AODV 106 [0 ffffffff 0 800] ----- [0:255 -1:255 30 0]
```

Figure 2.1 Old Format Trace File Example

The old trace file format has 12 mandatory fields and some other optional fields based on the type of protocols [3]. In order to generate an analysis result of the old trace file format, 6 of the 12 fields are required, which are event, time, trace level, packet id, packet size, and packet type.

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id
-------	------	--------------	------------	-------------	-------------	-------	-----	-------------	-------------	------------	-----------

Figure 2.2 Mandatory Fields for Old Trace File

2.1.2 New Trace File Format

The major difference between the old format and the new format is that the new format has more fields and contains more details [3]. However, for the analyzing purpose, the 6 same fields are required, and the rest of the fields can be ignored. Since new format has more fields on each line, the position of the 6 fields is different from the old format. To identify a trace file's format, the application only has to check the first and second field in the line. If the second field of the line is a time flag "-t", then it's the new format. If the first field is a "+", then the file is written in tagged format, which is not covered by this project. If it doesn't meet the previous conditions, then the file has an old format.

2.1.3 Tagged Format

Even though this application is not designed for analyzing tagged format trace file, it's important to know the difference from the tagged trace file and the new, old trace files. Tagged trace file always starts a line with a "+" or "-" sign [3]. By checking the first character of each line, tagged trace file can be easily filtered out.

2.1.4 Processing Big Data

Besides the different formats of the trace files, another common problem is the size of the trace file can be really large, which means that the processing time can be extremely long if the analyzer is not implemented by using multithreading technology. In fact, the old NsGTFA is able to process 60MB file in 51 seconds, and 1.2GB file in 4.1 minutes [6]. This will set a standard for the new version of NsGTFA. To implements multithreading in this application, the main thread will create a worker thread for each trace file. The analysis of all trace files should start at approximately the same time [22]. Larger trace file will take long time to analyze, but this is not going to block the analysis process of other trace files. More details about multithreading will be discussed in the Design section and Implementation section.

Implementation

7.1 Views

The view module has 4 Java classes and a readme file and it has a file structure shown as follow:

```
Views
|—— BarChart_AWT.java
|—— GraphFactory.java
|—— MainFrame.java
|—— PieChart_AWT.java
|—— readme.html
```

MainFrame.java is responsible for rendering all the windows. GraphFactory.java, BarChart.java and PieChart.java are responsible for generating analysis graphs.

7.1.1 Main Panel

One of the requirements is to develop a simple and intuitive user interface for this application. The first design of the GUI was inspired by the previous versions of NsGTFA.

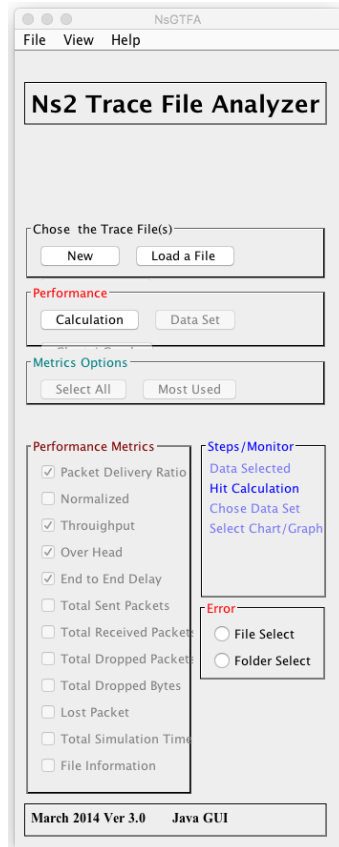


Figure 7.2 GUI Version 1.0

The problem of this design is obvious. The GUI is too narrow; there is not enough space to show all the buttons and labels. The text of the labels is shown crossed the subpanel border, and buttons are squeezed into random places. The second problem is that there is no place to show the analysis result and the progress of the analyzing process.

To tackle these problems, GUI version 2.0 was designed. On the left side of the GUI, it remains the same layout but has a wider width. On the right side of the GUI there are two new subpanels: Analysis Result Panel and File Information Panel. The result of each trace file will be shown as a single tab in the Analysis Result Panel. File Information Panel will show the text line while analyzing the trace file. To let the user see the progress of the analysis, on the bottom of the GUI, a progress bar is added.

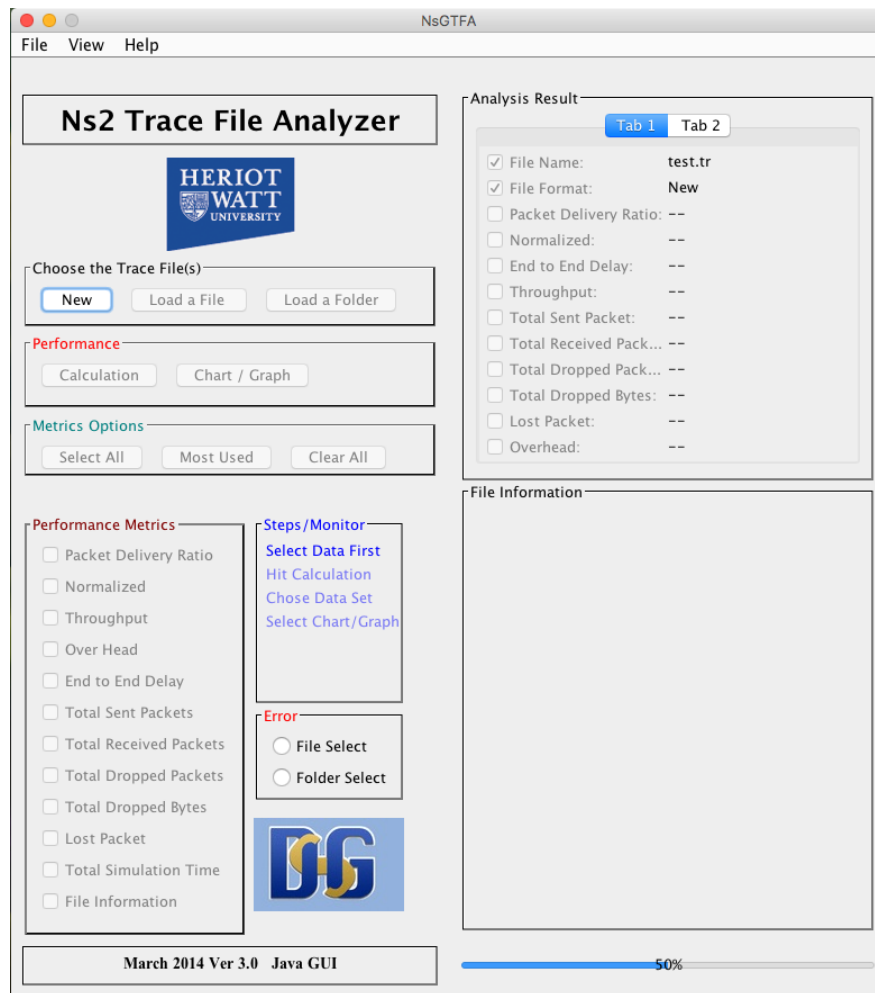


Figure 7.3 GUI Version 2.0

The second design of the GUI meet all the requirements, however, there was an issue when running the program on laptops that are smaller than 15-inch. On the vertical level, the GUI is too high. Laptops that are smaller than 15-inch usually don't have the screen size to show the entire GUI. Although the GUI is resizable, resizing will hide some of the components and cause inconvenience while using the application.

Therefore, a third version of GUI was made. Since most of the users will only pay attention to the analysis result, File Information Panel was removed from the GUI. By removing the File Information Panel, it saves more space for other components and cut the height of the window. This revised GUI window can be successfully displayed on most of the modern computers.

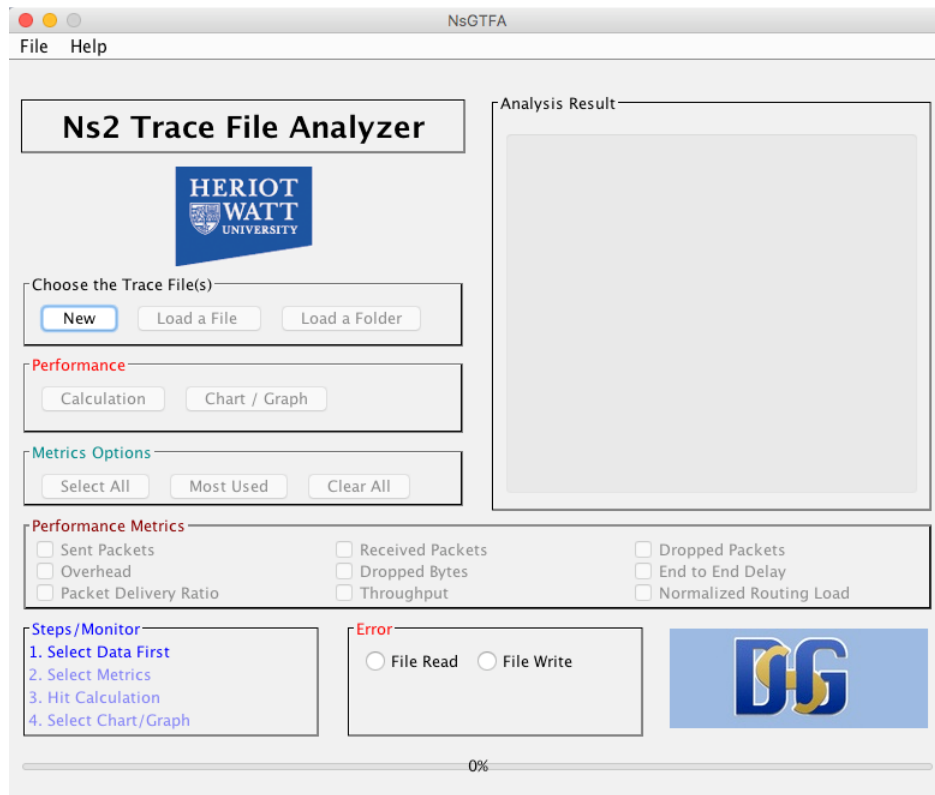


Figure 7.4 GUI Version 3.0

7.1.2 Graph Window

When user clicks on the “Chart/Graph” button, `MainFrame.java` will trigger the `setupGraphPanel()` method to generate all the graphs. There are three Java classes that are involved in the generation of graphs. `GprahFactory.java` will feed the data into `BarChart.java` and `PieChart.java`. These two Java files use the `JFreeChart` library to help generate the bar chart and pie chart.

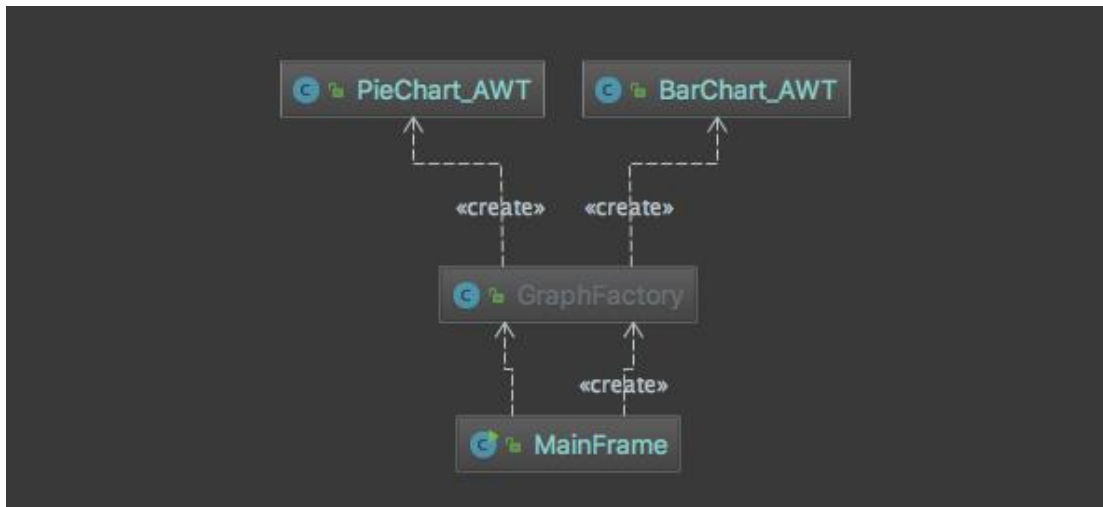


Figure 7.5 Class Diagram of the GraphFactory Class

By selecting from a dropdown list, user can decide to see the bar chart or pie chart.

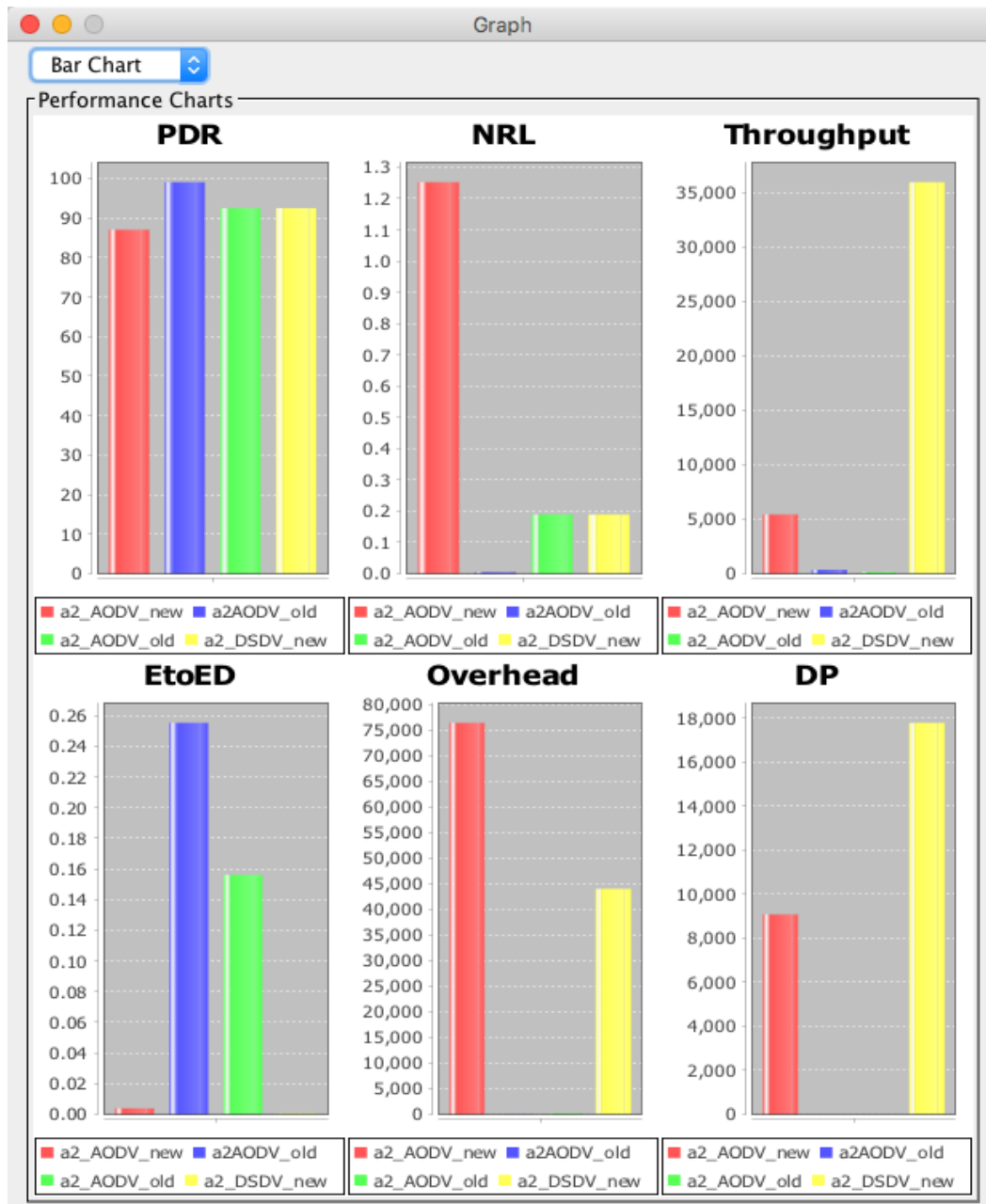


Figure 7.6 Bar Chart

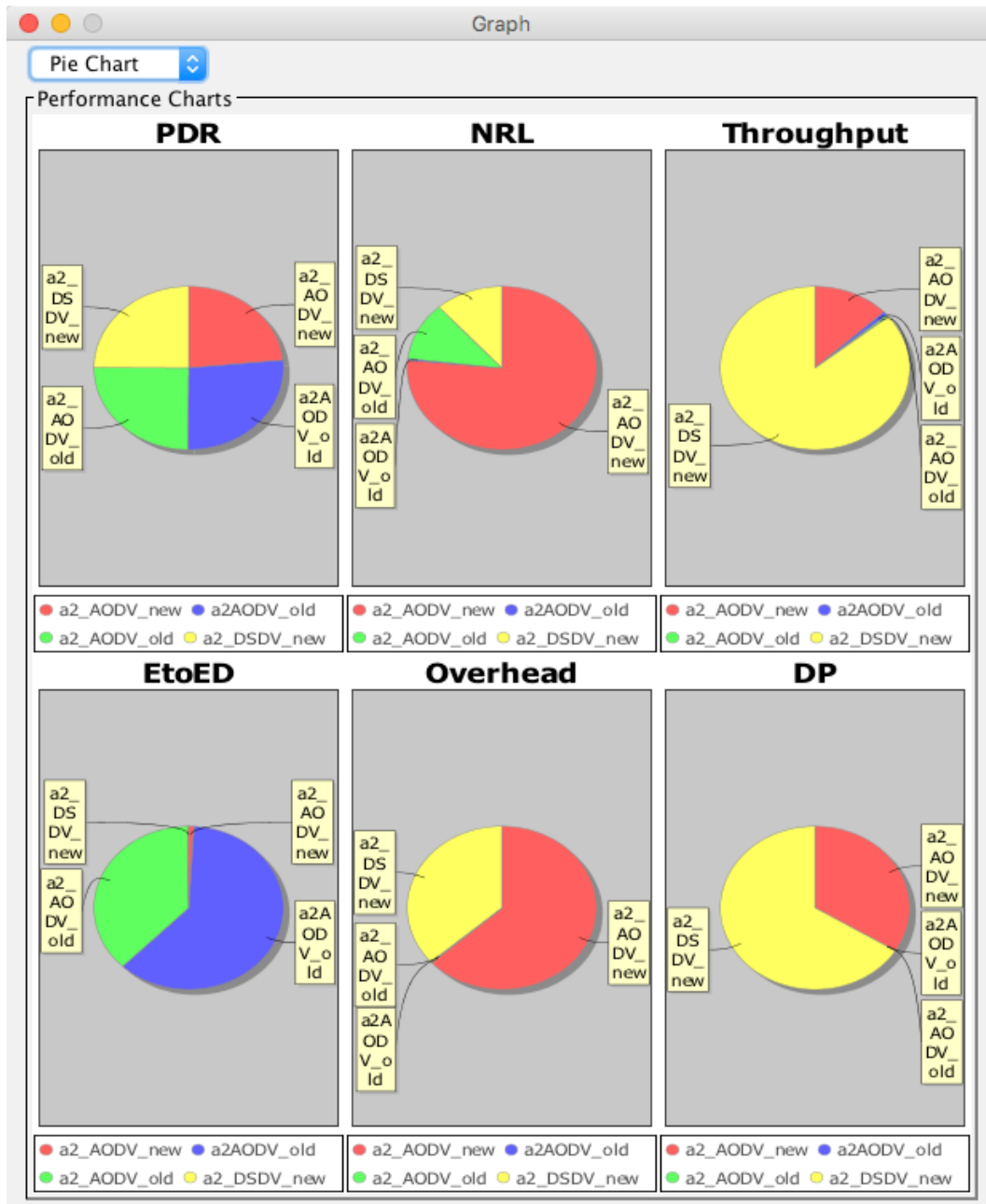


Figure 7.7 Pie Chart

7.1.3 User Manual Window

When developing a tool to help user do the analysis, it's important to make tool easy to pick up by the user. One way to do this, as already mentioned, is to make the GUI as simple as possible. Another way is to create a user manual to describe the running steps in details.

User can find the user manual by click the “Help” button on the menu bar, and a User Manual Window will show up.

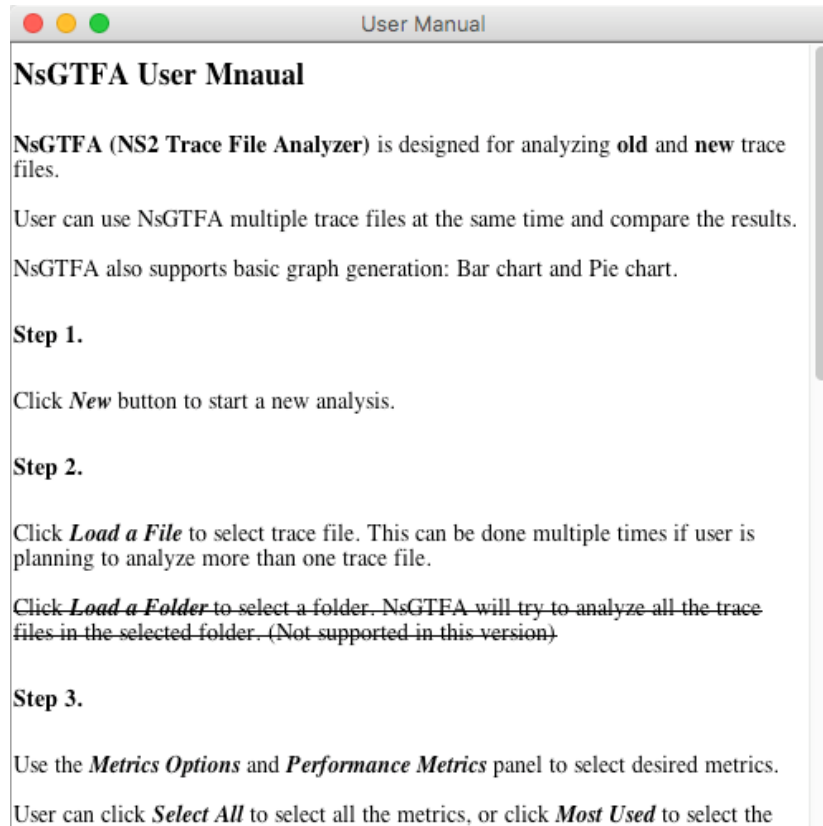


Figure 7.8 User Manual Window

The user manual is shown in a *JEditorPane* instead of a *JTextArea*, and the original text is saved in the *readme.html* file. The reason to use a *JEditorPane* is that it can read a HTML file and display the HTML tag in a proper format. For example, the crossed line in the user manual is created by using the `<strike>` tag:

```
<p>
  <strike>Click <b><i>>Load a Folder</i></b> to select a folder. NsGTFA will try to analyze all
the trace files in the selected folder. (Not supported in this version)
</strike>
</p>
```

7.2 Models

Models are the core files for this application that help manipulate data in the desired way, this includes reading data from files, analyzing data, formatting data and exporting data.

Underneath the model folder, there are 4 Java files:

```
Models
├── FileAnalyser.java
├── FileLoader.java
├── Helper.java
└── ModelAnalysisResult.java
```

Following is the class diagram of the model files:

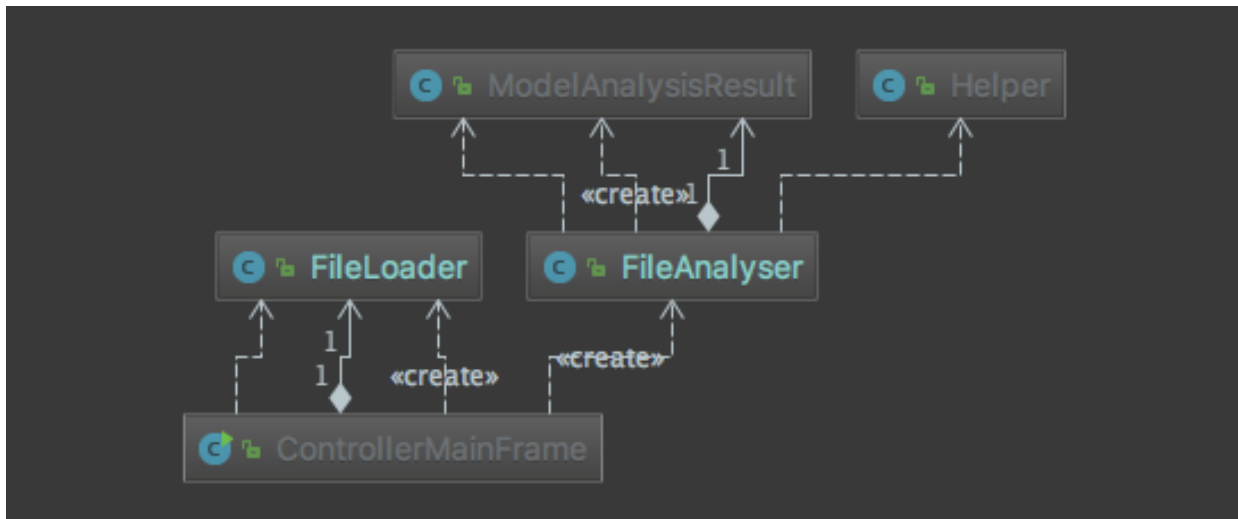


Figure 7.9 Class Diagram of the Models

7.2.1 Loading Files into Application

FileLoader.java implements *JFileChooser* object to open a file selection window. The default file format is filtered by using a *FileNameExtensionFilter*, which only allows the user to select files with “.tr” extension:

```
JFileChooser fileChooser = new JFileChooser(defaultPath);
fileChooser.setAcceptAllFileFilterUsed(false);
FileNameExtensionFilter filter = new
FileNameExtensionFilter("No Trace Files Only", "tr");
fileChooser.addChoosableFileFilter(filter);
```

The default path of the file chooser is the user's root directory. After selecting a trace file, the default path will be set to that trace file's parent directory. This will help the user to select multiple file in the same folder. The selected trace file will then be returned as a File object, and saved in an ArrayList. In this way, the user can select multiple trace files and be ready to do the analysis.

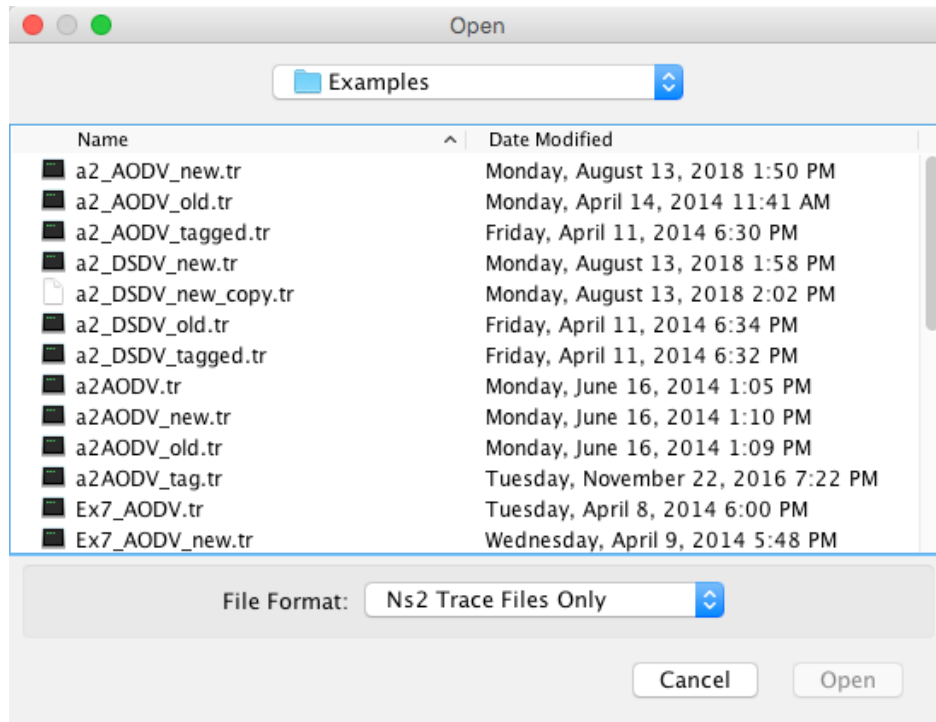


Figure 7.10 File Selection Window

7.2.2 Analyzing the File

The core functionality of this application is to analyze NS2 trace files. This analyzing process is fully handled by the *FileAnalyser* class. *FileAnalyser* extends the abstract *SwingWorker* class, and implements the following methods:

```
public class FileAnalyser extends SwingWorker<Model AnalysisResult, Integer> {
    @Override
    protected Model AnalysisResult doInBackground() throws Exception {
        while ((thisLine = br.readLine()) != null) {
            // Analyze the line...
            publish((int) Math.round(readLength / fileSize * 100));
        }
    }
    @Override
```



```

protected void done() {
    try {
        get();
    }
    catch (ExecutionException e) {
        e.printStackTrace();
    }
}

@Override
protected void process(List<Integer> chunks) {
    int i = chunks.get(chunks.size() - 1);
    progressBar.setValue(i);
}
}

```

The *doInBackground* method executes in a worker thread. This method handles all the analyzing work. First, the trace file format has been determined. According to the format, old or new, each line of the trace file is tokenized into pieces. The corresponding token value that is involved in the analyzing process is then save into memory, and will be used to analyze the file. The *doInBackground* method will also output the analysis result to a text file when the calculation is done.

The *publish* method inside the *doInBackground* method updates the intermediate result to keep track of the analyzing progress. Every time the application reads a single line, the line size will be summed up to calculated the total size of file that has been already analyzed. Then the application can use this value to divide the total size of the trace file to obtain the progress of the analysis.

The *process* method is used to get the intermediate result that is passed in by the *publish* method. Every time the *publish* method updates a result, it will push the result into the chunk list. In order to get the latest result, the process method can simply get the last element in the list and then update the progress bar value.

When all the calculation is done, the final result can be obtained by calling the *get* function inside the *done* method, and this result is ready to be updated.

7.2.2.1 Error Handling

Besides analyzing the trace file, *FileAnalyser* will also handle two types of error: Format Error and Export Error. When user trying to analyze a trace file that doesn't follow the old or new format, the application will throw an exception and a pop-up window will appear to notify the user.

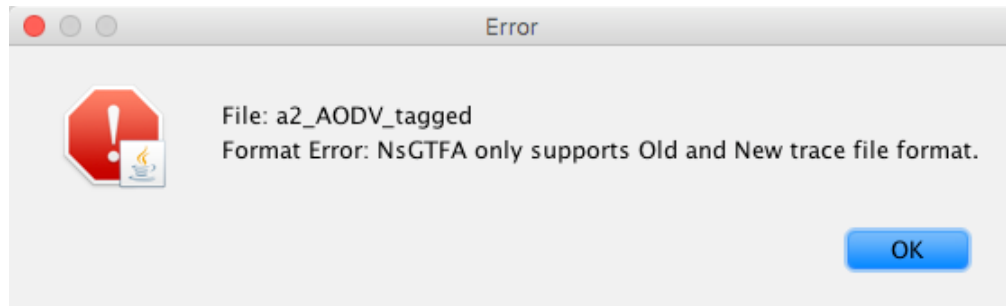


Figure 7.11 Format Error

The second type of error, Export Error, is caused by unsuccessfully writing analysis result into a text file. Similar to the Format Error, a pop-up window will show a warning to the user.

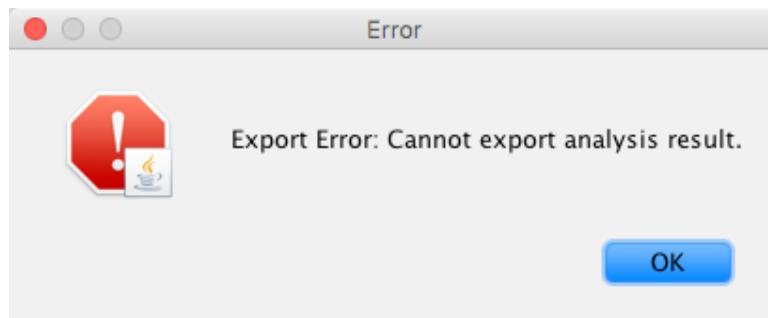


Figure 7.12 Export Error

7.2.3 Saving Analysis Result

The analysis result is a combination of different types of variables; it includes *Strings*, *Integers* and *Floats*. Therefore, there is no good data structure to store all these values together. This is why it's necessary to create a special object to hold all the results. *ModelAnalysisResult.java* is designed to hold all three types of variables into their corresponding *HashMap*, and easy to format and export these results.

```
private LinkedHashMap<String, String> stringMap;  
private HashMap<String, Integer> intMap;  
private HashMap<String, Float> floatMap;
```

7.2.4 Helper Functions

The Helper class is a utility class that helps to measure the execution time of each analysis. There are two static functions inside this class:

```
public static void startPerformanceTest() {
    startTi me = System.current Ti me Milli s();
    System.out.println("Start at: " + startTi me);
}
public static void endPerformanceTest() {
    long endTi me = System.current Ti me Milli s();
    long execTi me = (endTi me - startTi me) / 1000;
    System.out.println("End at: " + endTi me);
    System.out.println("Performance: " + execTi me + "s");
}
```

Static function is a good approach when there is no instance object required, and they can be easily used in any other class to serve as a helper function.

7.3 Controller

ControllerMainFrame.java is the main class of this application. It initiate the program by invoking *SwingUtilities.invokeLater()*:

```
SwingUtilities.invokeLater(() ->{
    try {
        MainFrame v = new MainFrame();
        FileLoader f = new FileLoader();
        Controller MainFrame c = new Controller MainFrame(v, f);
        c.actionListener();
    } catch (Exception e) {
        e.printStackTrace();
    }
});
```

SwingUtilities.invokeLater() will properly schedule the tasks in event dispatch thread.

The controller then can add event listener to the GUI components:

```
public void actionListener() {
    view.getBtnNew().addActionListener(e -> newAnalyse());
    view.getBtnReset().addActionListener(e -> reset());
    // More components to add event listener
    // ...
}
```

Every time an event is triggered by the user's gesture, the event will be executed on the event dispatch thread. If the event is time-consuming, in this case analyzing the trace file, the event dispatch thread will create a worker thread to do this task in the background:

```
// The following code creates a worker thread for // each file path in the ArrayList<String>
for (String filePath : filePaths) {
    // FileAnalyser extends Swing Worker class
    new FileAnalyser(view, filePath, tmp).execute();
}
```

Each worker thread should start to execute at the time it's created. Therefore, when analyzing multiple trace files at the same time, all trace files should get analyzed at the same time. Based on the size of each trace file, the execution time of the worker thread varies differently.

The following example shows the analysis of two trace files with different sizes:

1. *a2_AODV_new*: 350MB
2. *a2_DSDV_new*: 1GB

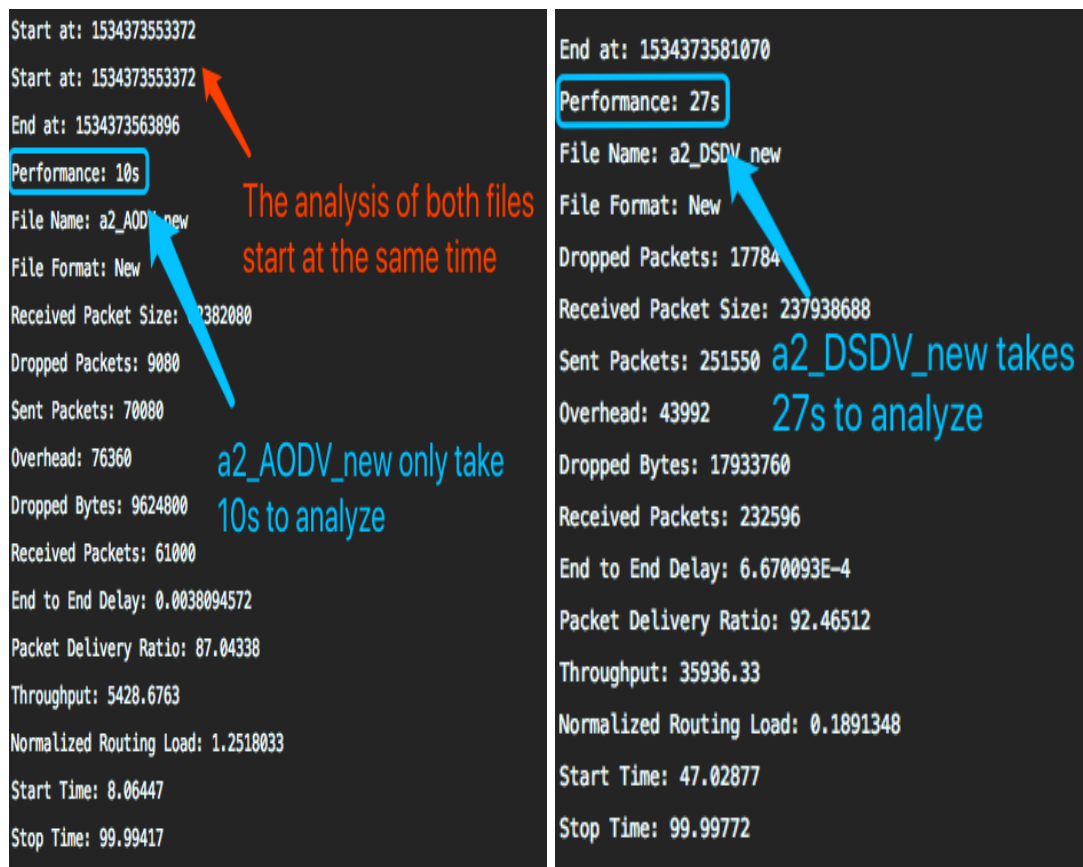


Figure 7.13 Example of Multithreading Analysis

8.3 Result Accuracy

One of the most important aspects of this application is to produce an accurate precise result of the analyzed trace file. To do this, the same trace file is analyzed by using three different tools or scripts, and the results are compared and verified.

8.3.1 New Format

The new format trace file analysis is conducted by using the older version of NsGTFA and the new version of NsGTFA.

Start at: 1534404370066	**StartTime .. 12.022363
End at: 1534404370369	**StopTime .. 49.981655
Performance: 0.303s	
File Name: a2A0DV_new	**Send Packets .. 1562
File Format: New	Receive Packets .. 1547
Received Packet Size: 1581056	Lot Packets .. 15
Sent Packets: 1562	No. of dropped data (packets)0
Overhead: 8	No. of dropped data (bytes) 0
Received Packets: 1547	Routing packets (OverHead) 8
End to End Delay: 0.25516537	Normalized routing load 0.0051712994
Packet Delivery Ratio: 99.03969	Packet delivery Fraction PDF ..99.03969
Throughput: 333.21085	Avg End-End delay 0.25516537
Normalized Routing Load: 0.0051712994	Avg End-End delay (ms)255.16537
Start Time: 12.022363	Average Throughput[kbps] 333.21085
Stop Time: 49.981655	Elapsed Time = 0.37748736000000005
Result from new version of NsGTFA	Result of older version of NsGTFA
	recved size =1581056
	end_to_end_delay 394.74084

Figure 8.2 Comparing Analysis Results of New Format Trace File

8.3.2 Old Format

The old format trace file analysis is conducted by using an AWK script (see Appendix) and the new version of NsGTFA.

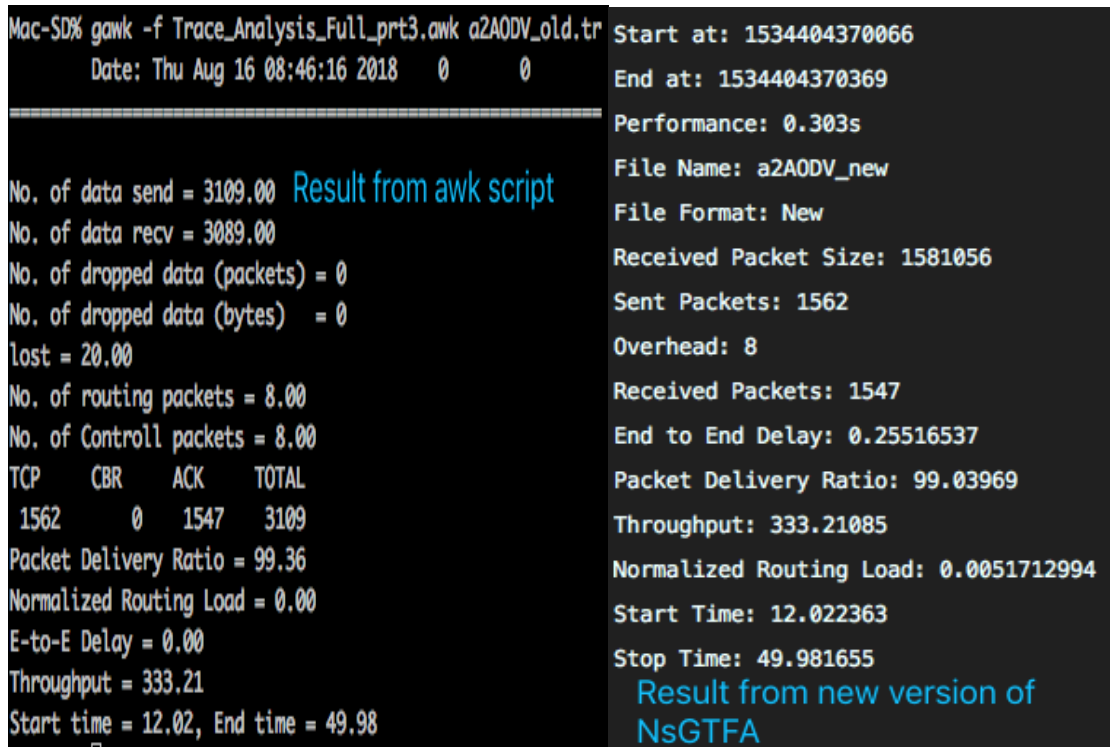


Figure 8.3 Comparing Analysis Results of Old Format Trace File

Despite the format difference, these two trace files, *a2AODV_new.tr* and *a2AODV_old.tr* are generated by the same scenario. The analysis results of these two files should be the same. By comparing the images above, the new version of NsGTFA produces a very accurate analysis result.

8.4 Cross-platform Portability

To run the application on different platforms, the easy way is to build a jar file and then run that jar file on the desired computer. Before running the application, it's important to make sure the system already installed the Java Runtime Environment (JRE). For this application, JRE 1.8.0 is required.

As shown in the following image, the *JavaNs2.jar* file is only 2MB. Nothing else is required besides the JRE to run this application. The small size of the application makes it easy to download and transport.


Name	Date Modified	Size	Kind
 JavaNs2.jar	Yesterday at 1:14 PM	2 MB	Java JAR file

Figure 8.4 JavaNs2.jar

8.4.1 Mac OS

To run NsGTFA on the Mac OS, simply import the JavaNs2.jar to the system, and double click on it to run.

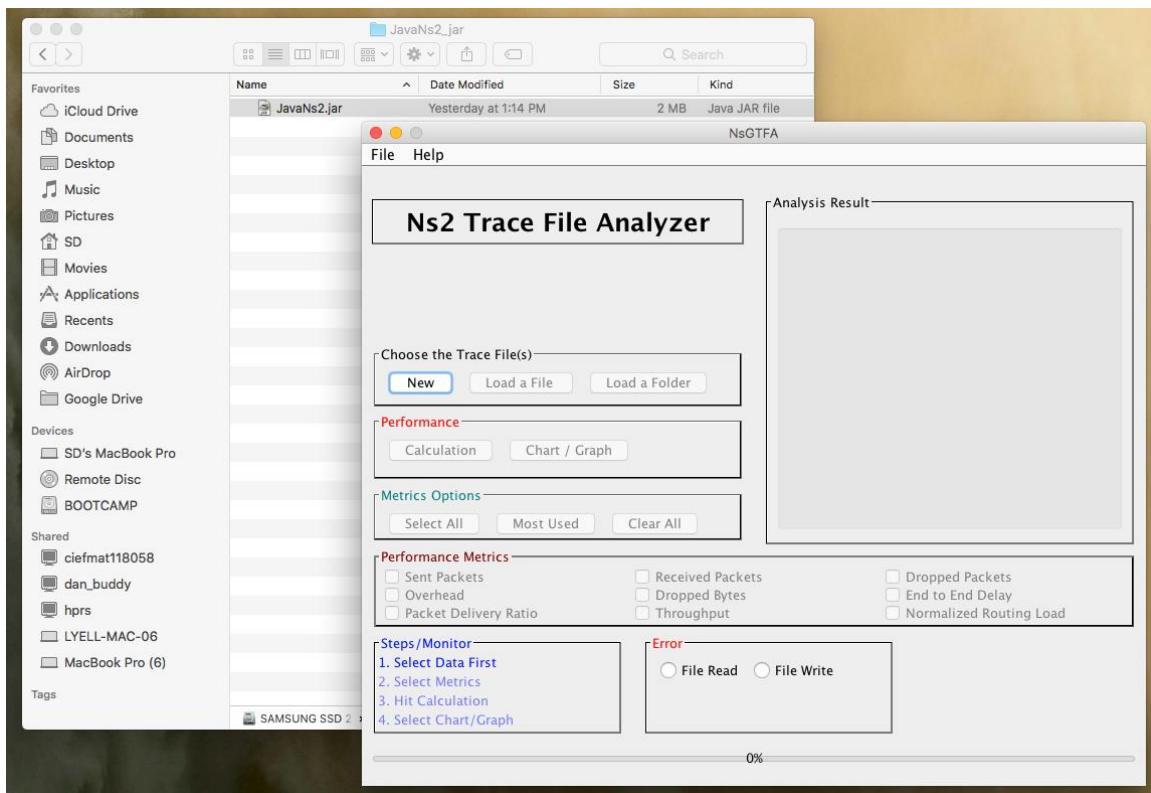


Figure 8.5 Running NsGTFA on Mac OS

Analyzing trace files on Mac OS:

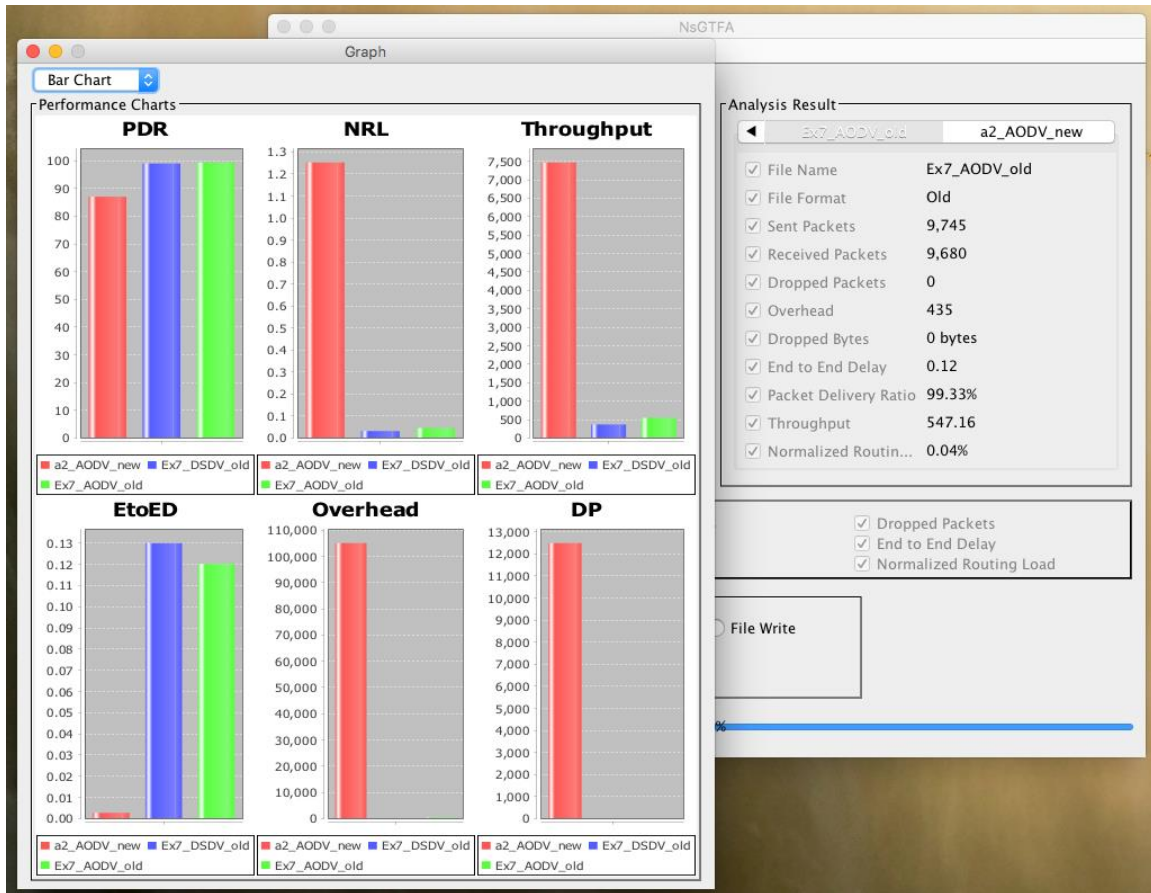


Figure 8.6 Analyzing Trace Files on Mac OS

Exporting result text files on Mac OS:

Result_a2_AODV_new.txt												
	Send	Receive	Drop	O/H	[kbps]	PDR	NRL	EED	startTime	stopTime	Date / Time	Filename
1	96362	83875	12485	105024	7464.431	87.04	1.25	0.00	8.06	99.99	2018/08/16 09:22:10	a2_AODV_new

Result_Ex7_DSDV_old.txt												
	Send	Receive	Drop	O/H	[kbps]	PDR	NRL	EED	startTime	stopTime	Date / Time	Filename
1	3911	3872	0	123	369.310	99.00	0.03	0.13	64.13	149.99	2018/08/16 09:22:03	Ex7_DSDV_old

Result_Ex7_AODV_old.txt												
	Send	Receive	Drop	O/H	[kbps]	PDR	NRL	EED	startTime	stopTime	Date / Time	Filename
1	9745	9680	0	435	547.163	99.33	0.04	0.12	5.03	149.95	2018/08/16 09:22:03	Ex7_AODV_old

Figure 8.7 Exporting Analysis Results on Mac OS

8.4.2 Windows

Running NsGTFA on a Windows computer is similar to running it on the Mac OS. Importing the *JavaNs2.jar* file into the system, and double click on it. Following images show an example of using NsGTFA to analyze trace files on Windows 10.

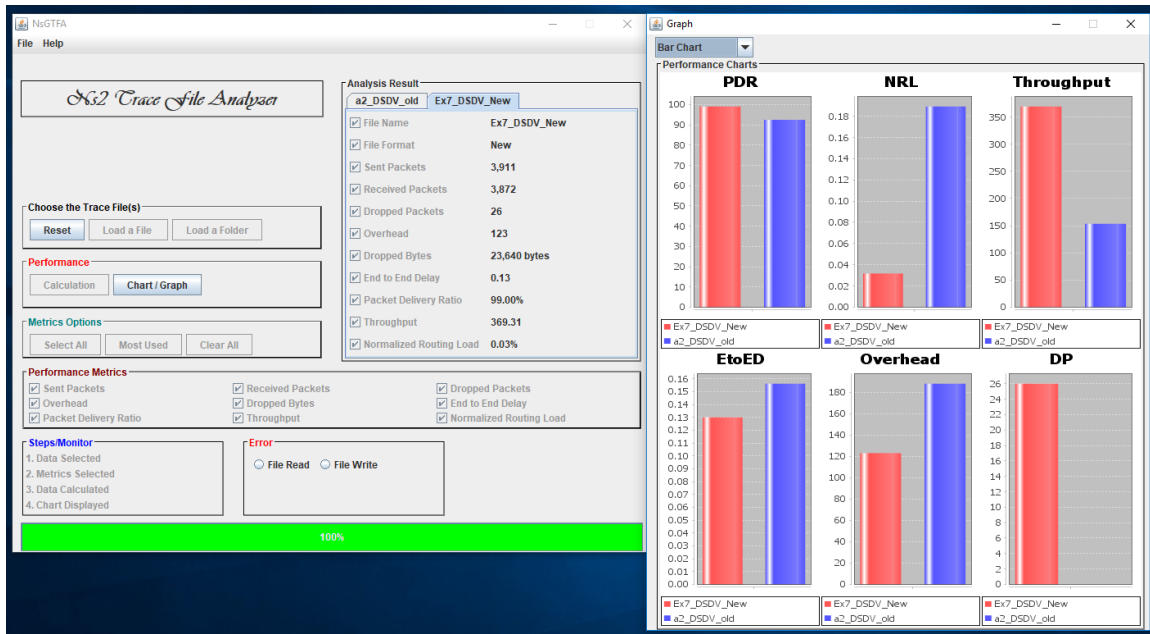


Figure 8.8 Running NsGTFA on Windows 10

Exporting result to a text file:

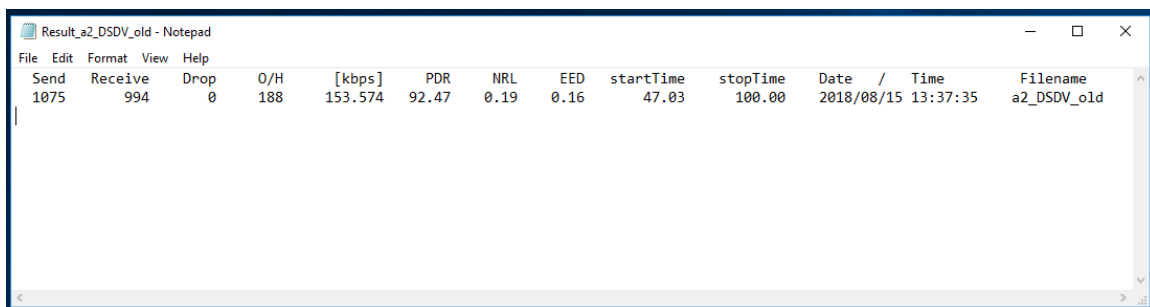


Figure 8.9 Exporting Analysis Result on Windows 10

8.4.3 Linux (Centos 7.5)

To run NsGTFA on a Linux machine, follow the steps shown below:

1. Import the JavaNs2.jar file into the desired directory
2. Open Terminal and go to JavaNs2.jar's parent directory by using change directory command *cd*
3. Execute the jar file by using the following command, and this should open the application window:

```
java -jar JavaNs2.jar
```

```
bash-4.2$ cd '/home/msc/ss244/Desktop/NS2/'
bash-4.2$ java -jar JavaNs2.jar
```

Figure 8.10 Terminal Command Line

Application window shows up after entering the Java command:

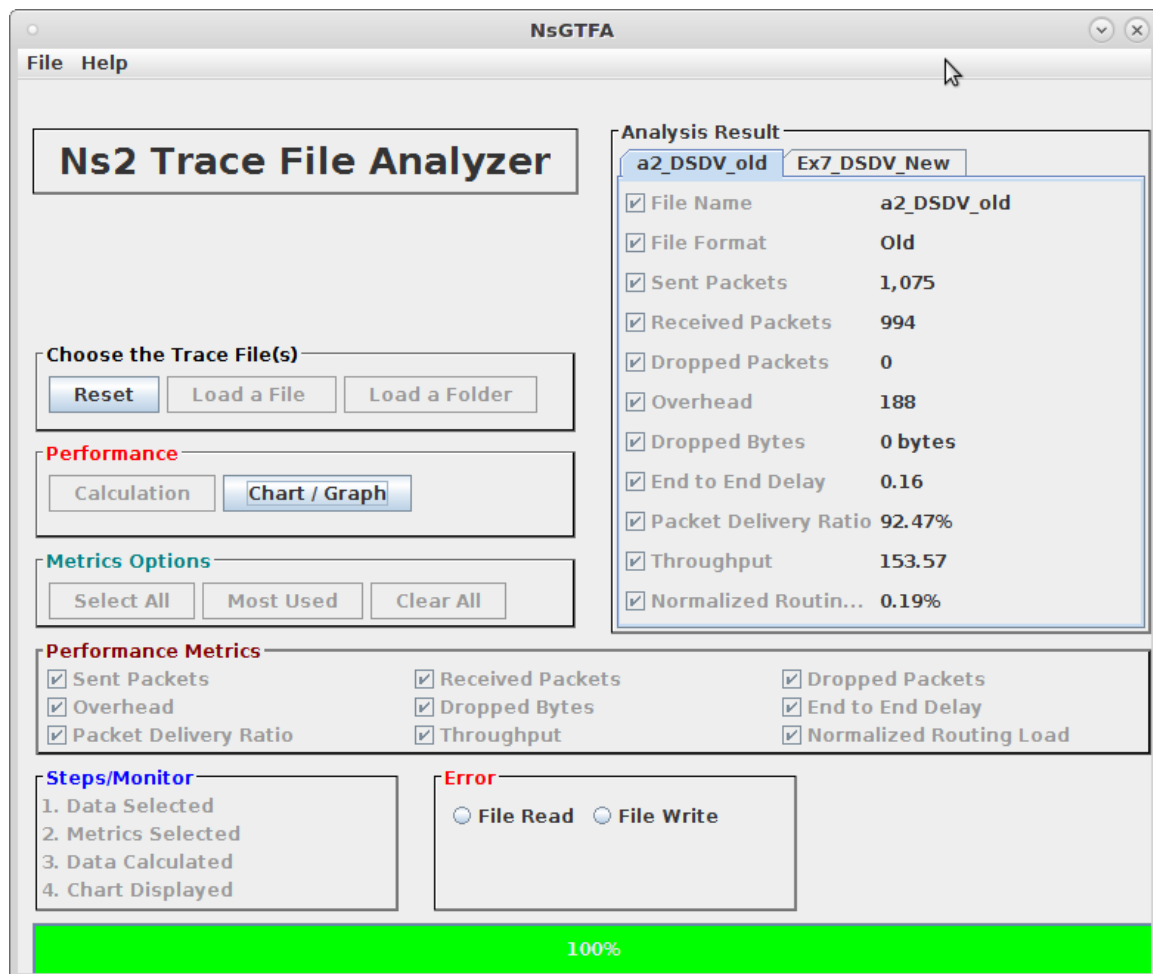


Figure 8.11 Analyzing Trace Files on Linux

Generating graphs for the analysis result:

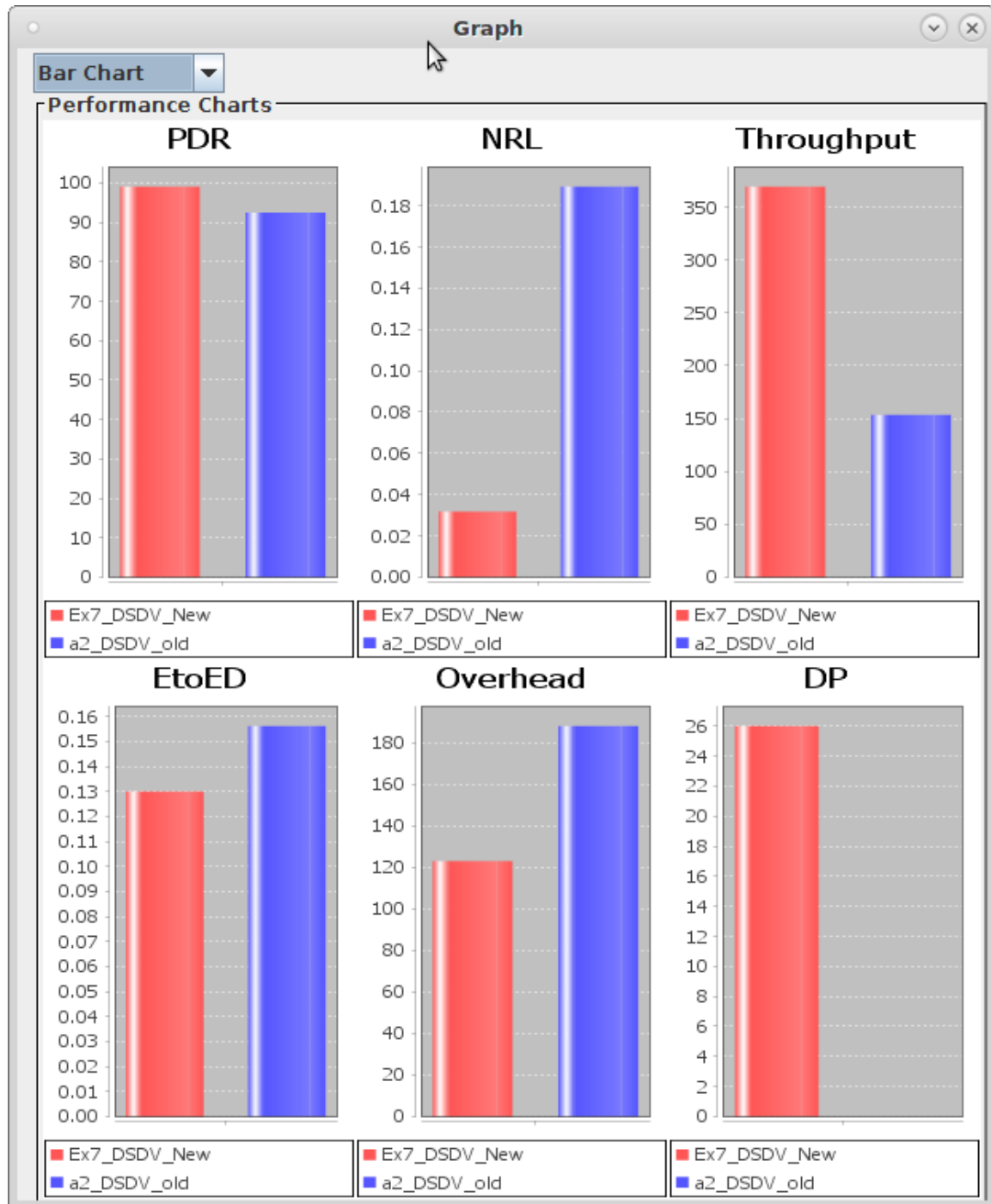


Figure 8.12 Graph Result on Linux

Exporting analysis result to a text file:

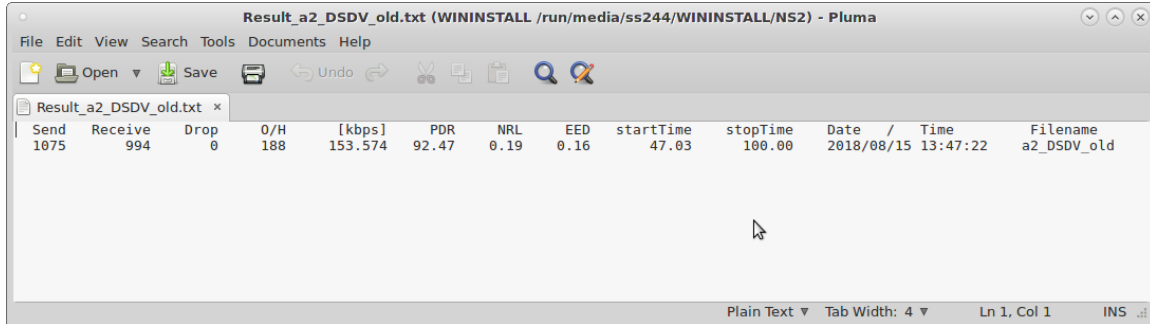


Figure 8.13 Result Text File on Linux

Overall this application works smoothly on all three platforms. Based on the different operating system, the GUI displays a little bit different too, but this is not going to affect the functionality of the application.

However, there is a flaw when compile the application into a jar file. On all three platforms, the main GUI doesn't display the HW and DS logo. This is because the paths of these images are not coded correct when the application is compiled into a jar file.

References:

- [1] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. Boston, MA: Springer US, 2012, pp. 1-40.
- [2] "The Network Simulator - ns-2", Isi.edu, 2018. [Online]. Available: <https://www.isi.edu/nsnam/ns/>. [Accessed: 12- Apr- 2018].
- [3] "NS-2 Trace Formats - nsnam", *Nsnam.sourceforge.net*, 2018. [Online]. Available: http://nsnam.sourceforge.net/wiki/index.php/NS-2_Trace_Formats. [Accessed: 12- Apr- 2018].
- [4] "NS by Example", Nile.wpi.edu, 2018. [Online]. Available: <http://nile.wpi.edu/NS/>. [Accessed: 12- Apr- 2018].
- [5] "6.Trace file Definition", *NETWORK SIMULATOR 2*, 2018. [Online]. Available: <https://cloudns2.wordpress.com/trace-file-definition/>. [Accessed: 12- Apr- 2018].
- [6] I. Ibrahim, P. King and H. Loidl, "NsGTFA: A GUI Tool to Easily Measure Network Performance through the Ns2 Trace File", *Journal of Intelligent Systems*, vol. 24, no. 4, 2015.
- [7] "ns2trana", *Sites.google.com*, 2018. [Online]. Available: <https://sites.google.com/site/ns2trana/>. [Accessed: 12- Apr- 2018].
- [8] C. Bouras, S. Charalambides, M. Drakoulelis, G. Kioumourtzis and K. Stamos, "A tool for automating network simulation and processing tracing data files", *Simulation Modelling Practice and Theory*, vol. 30, pp. 90-110, 2013.
- [9] "Network Simulations | Research Unit 6", Ru6.cti.gr, 2018. [Online]. Available: <http://ru6.cti.gr/ru6/research-areas/network-simulations#TRAFIL>. [Accessed: 12- Apr- 2018].
- [10] "AWT vs. Swing", Northstar-dartmouth.edu, 2018. [Online]. Available: http://northstar-www.dartmouth.edu/doc/idl/html_6.2/AWT_vs._Swing.html. [Accessed: 12- Apr- 2018].
- [11] "www.jfree.org", *Jfree.org*, 2018. [Online]. Available: <http://www.jfree.org/>. [Accessed: 12- Apr- 2018].
- [12] K. Adam and H. Dorota, "Automated Defect Prevention: Best Practices in Software Management", *Wiley.com*, 2018. [Online]. Available:

- <http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470042125.html>. [Accessed: 12- Apr- 2018].
- [13] "Buffered Streams (The Java™ Tutorials > Essential Classes > Basic I/O)", *Docs.oracle.com*, 2018. [Online]. Available: <https://docs.oracle.com/javase/tutorial/essential/io/buffers.html>. [Accessed: 12- Apr- 2018].
- [14] "Comparison computational complexity", *Upload.wikimedia.org*, 2018. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/7/7e/Comparison_computational_complexity.svg. [Accessed: 12- Apr- 2018].
- [15] "Google Java Style Guide", *Google.github.io*, 2018. [Online]. Available: <https://google.github.io/styleguide/javaguide.html>. [Accessed: 12- Apr- 2018].
- [16] "The MIT License | Open Source Initiative", *Opensource.org*, 2018. [Online]. Available: <https://opensource.org/licenses/MIT>. [Accessed: 12- Apr- 2018].
- [17] Moran, A. (2014). *Agile Risk Management*. Springer Verlag. ISBN 3319050079.
- [18] A. Jacobs, "The Pathologies of Big Data", *Queue*, vol. 7, no. 6, p. 10, 2009.
- [19] "Byte Streams (The Java™ Tutorials > Essential Classes > Basic I/O)", *Docs.oracle.com*, 2018. [Online]. Available: <https://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html>. [Accessed: 16- Aug- 2018].
- [20] "Buffered Streams (The Java™ Tutorials > Essential Classes > Basic I/O)", *Docs.oracle.com*, 2018. [Online]. Available: <https://docs.oracle.com/javase/tutorial/essential/io/buffers.html>. [Accessed: 16- Aug- 2018].
- [21] M. Sami, "The Waterfall Model, a different perspective - Mohamed Sami", *Mohamed Sami*, 2018. [Online]. Available: <https://melsatar.blog/2018/02/16/the-waterfall-model-a-different-perspective/>. [Accessed: 16- Aug- 2018].
- [22] "Lesson: Concurrency in Swing (The Java™ Tutorials > Creating a GUI With JFC/Swing)", *Docs.oracle.com*, 2018. [Online]. Available: <https://docs.oracle.com/javase/tutorial/uiswing/concurrency/index.html>. [Accessed: 16- Aug- 2018].

- [23] Georgia Tech, "Using MVC with Swing Components", 2018.
- [24] A. Fowler, "A Swing Architecture Overview", Oracle.com, 2018. [Online]. Available: <https://www.oracle.com/technetwork/java/architecture-142923.html>. [Accessed: 16- Aug- 2018].
- [25] E. Freeman, E. Robson, B. Bates and K. Sierra, Head First Design Patterns. Sebastopol: O'Reilly Media, Inc., 2008.

Appendices

Appendix A: User Manual

NsGTFA is designed for analyzing **old** and **new** trace files.

User can use NsGTFA multiple trace files at the same time and compare the results.

NsGTFA also supports basic graph generation: Bar chart and Pie chart.

Step 1.

Click *New* button to start a new analysis.

Step 2.

Click *Load a File* to select trace file. This can be done multiple times if user is planning to analyze more than one trace file.

~~Click *Load a Folder* to select a folder. NsGTFA will try to analyze all the trace files in the selected folder. (Not supported in this version)~~

Step 3.

Use the *Metrics Options* and *Performance Metrics* panel to select desired metrics.

User can click *Select All* to select all the metrics, or click *Most Used* to select the most used metrics.

To clear all the selections, click *Clear* button.

Step 4.

Click *Calculation* to analyze the selected trace files.

The progress bar will show the progress percentage of each trace file's analysis.

The analysis result of each trace file will displays in the *Analysis Result* panel. User can use left, right arrow to check each result.

If one of the selected trace file doesn't have the correct format (**old, or new**), a pop-up window will show up and the *Error* panel will indicate that there is a *File Read* error.

Any trace file that doesn't have a correct format will be skipped, and the rest of the selected trace files will continue their progress until finish.

Each trace file that has been successfully analyzed will generate a corresponding result file, which is named as "*Result_originalFileName.txt*".

User is supposed to have all their trace files saved in the same folder. This will help export the result file in an easy and organized way, since all the result files will be located in the same folder of the first selected trace file.

If NsGTFA is not able to export the result file, a pop-up window will show up and the *Error* panel will indicate that there is a *File Write* error.

Step 5.

To view the graphs, user can click *Chart/Graph* button to show the Graph Window. At the top-left of the Graph Window, user can select the graph type, either a Bar chart or Pie chart.

To start a new analysis, click the *Reset* button and repeat Step 1-5.

Appendix B: Project Source File and AWK Script

This project can be found on Github at: <https://github.com/sunsidi/NsGTFA>

The AWK Script used to analyze old trace file:

<https://github.com/sunsidi/NsGTFA/blob/master/src/JavaNs2/Extra/Analysis.awk>