Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Computer Science
2020 | LIU-IDA/LITH-EX-A–20/003–SE

Synthetic Generation of Realistic Network Traffic

Syntetisk generering av realistisk nätverkstrafik

Emil Gustafsson

Supervisor : Patrick Lambrix
Examiner : Niklas Carlsson

External supervisor : Jonathan Jogenfors

Linköpings universitet
SE--581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

The industry shows a clear need for synthetically generated realistic network traffic. As a possible solution, this thesis proposes a method for generating such data in an automatic and controllable manner. This thesis first examines the characteristics of real network traffic and analyzes the length of ON/OFF periods. The theory that network traffic exhibits self-similarity and high variability is once again tested and proven, thereby also the fact that the ON/OFF periods of real network traffic comes from a heavy-tailed distribution. Thereafter, the thesis proposes a way to simulate user interaction with real world applications by using a UI testing framework called WinAppDriver. This tool is then used to synthetically generate network traffic, of which the characteristics are analyzed and compared to that of real network traffic. The results show that the generated network traffic is indeed statistically similar to real network traffic. Finally, everything is combined by setting up a whole network of virtual machines with simulated users.

Acknowledgments

I would like to show my gratitude to Sectra, for giving me the opportunity to do my thesis work at your organization. I would also like to thank Jonathan Jogenfors and Niklas Carlsson for providing feedback and guidance during my work. Lastly, I would like to thank my opponent and future colleague Silas Lenz for your thorough feedback and our many interesting discussions during the thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The outline of this chapter is as follows: In Section 1.1 the research problem and its motivation are introduced. After that, the purpose and goals of the thesis are described in Section 1.2. These goals are stated in concrete terms in Section 1.3. Section 1.4 states the contributions that the thesis provides. Section 1.5 describes the scope of this thesis. Finally, Section 1.6 describes the structure of the rest of the report.

1.1 Motivation

Network engineers and security analysts need methods to synthetically generating network traffic in order to test themselves, the network, and the tools that they use. In that context, the characteristics of the generated network traffic is very important. To generate network traffic is a trivial task, but to synthetically generate traffic that is convincing to professionals is much more difficult. Therefore, a lot of research has been conducted on analysis and characterization of network traffic in order to understand how real network traffic behaves. Unfortunately, the majority of this research has been done on a larger scale [1, 2, 3, 4, 5], such as on the network traffic at the Internet Service Provider (ISP) level, thereby consisting of thousands of individual clients. In addition to this, many tools have been developed with the purpose of imitating this type of network traffic, both tools that replay recorded traffic and network throughput estimation tools.

This research of course has its applications, but it does a poor job of generating convincing network traffic from a small network or a single client. It might be possible to increase this quality, to a level where it is difficult for an experienced network engineer to determine if the network traffic is synthetically generated or not, by simulating real users.

1.2 Aim

This thesis aims to bridge this gap by designing and evaluating a method of synthetically generating *realistic* network traffic in an *automatic* but *controllable* manner.

In order to later be able to evaluate this method, the mentioned attributes must first be defined, i.e. what do they mean in the scope of this thesis, why are they important and how can they be measured.

1.2.1 Realistic

According to the Oxford dictionary Lexico.com[1], realism can be defined as "the quality or fact of representing a person or thing in a way that is accurate and true to life". In the scope of this thesis, this definition is interpreted and applied by stating that the realism of the resulting

[1]Source: `https://www.lexico.com/en/definition/realism`, accessed on December 10th, 2019.

network traffic is measured by the statistical characteristics *variability* and *self-similarity* which are explained in detail in Section 3.1.

1.2.2 Automatic

The resulting method of this thesis should be automatic, meaning that after the initial setup and configuration. The system should be able to run and generate network traffic virtually indefinitely without administration.

1.2.3 Controllable

The method should also be controllable in the sense of giving the administrator the ability to add, remove and alter the circumstances of the nodes in the network at runtime. The system's current behavior should always be transparent to the administrator by offering the possibility of inspecting what every node is doing at every point in time.

1.3 Research Questions

This thesis seeks to answers the following two research questions:

- **RQ 1:** How to simulate the network traffic of multiple users, in a network of virtual machines?
- **RQ 2:** How realistic is the generated network traffic, in terms of variability and self-similarity?

1.4 Contributions

This thesis contributes to the industry by explaining and evaluating a method for synthetically generating realistic network traffic, by simulating users and network devices. The advantages of this method, and what it can be used for, is further described in Chapter 2.

1.5 Delimitations

To further understand what the purpose of this thesis is, we must define what it is not. This thesis aims to simulate realistic network traffic as described above, but the effort of studying the exact behavior of people in the chosen scenario will not be performed. Instead, examples of how people realistically *could* behave will be reproduced. This thesis is also not about trying to create a workload generator with the purpose of generating as *much* network traffic as possible, even if the tool could be tweaked to work in that way. Furthermore, the tool presented in this thesis for simulating users is currently limited to use on the Windows 10 operating system because of its use of Microsoft's WinAppDriver.

1.6 Structure of the Report

Chapter 2 mentions related work in the field and describes how it differs from the method presented in this thesis. Chapter 3 covers the theory and techniques used to develop the presented method. Chapter 4 describes how the method was developed and evaluated. The result of this evaluation is presented in Chapter 5. Chapter 6 discusses the methodology and the possible implications of the thesis. Finally, Chapter 7 presents the conclusions and answers the research questions.

Chapter 2

Related Work

This section lists and discusses related research in the field. No research was found on generating realistic network traffic with ON/OFF periods (see Section 3.1.1) by simulating users. Therefore, this chapter will discuss what has been achieved, how it may be utilized and how it differs from this thesis. The mentioned research is divided into the following parts: network traffic characterization, network traffic generation, simulated network devices, and lastly, simulated users.

2.1 Network Traffic Characterization

Understanding a network and the habits of its users is a crucial step for network design and dimensioning for any serious actor. Network traffic in its turn, is a combination of application mechanisms and user behavior. To characterize network traffic, it must first be recorded. This can be performed at many points of a the Open Systems Interconnection [6] (OSI) model (see Figure 2.1), each with its own advantages and disadvantages. Recording user behavior might be valuable in some cases, but because of the mechanics at the layers below, it will not directly correlate to the network traffic measured at the physical, or even the transport layer. The physical location of the measurement also matters, since some nodes, such as routers, might add, remove or manipulate layers of the network traffic along the way. Some packets, such as local network file transfers, will hopefully not be recordable by an ISP. Equally, encrypted packets might only be legible at the sender and the receiver, but not at any other point along the way. Because it is not feasible to measure at every point in a large network, most studies has focused on either a specific location [7, 2, 8, 9, 10] or application/protocol [11, 12] depending on what is desired to characterize. Advancements in the research on network traffic characterization has among other things led to the understanding of how large network flows can be modeled. Models such as the pareto distribution process or ON/OFF model has been studied and used when developing software to synthetically generate resembling network traffic [13].

2.2 Network Traffic Generation

This section covers some tools that have been developed with the purpose of generating network traffic.

2.2.1 Traffic Replay Tools

Traffic replay tools are software that either records the traffic on the network by itself, or takes captured data as input, in order to then replay the same exact traffic onto the network. Even though tools such as TCPreplay[1] offer the ability to modify the packets content and inter-

[1]Source: `https://tcpreplay.appneta.com/`, accessed on December 10th, 2019.

7	Application Layer
6	Presentation Layer
5	Session Layer
4	Transport Layer
3	Network Layer
2	Datalink Layer
1	Physical Layer

Figure 2.1: A visualization of the OSI model.

arrival time, they can not for example generate any new traffic or wait for server responses. Traffic replay tools can also be used as a workload generator for network test benches such as WoTbench [14] in order to evaluate protocols and applications. Traffic replay tools therefore have a purpose in network and penetration testing, but they are not good at generating an unlimited amount of realistic network traffic.

2.2.2 Network Throughput Estimation Tools

Network throughput estimation tools such as IPerf[2] and Netperf[3] have also earned their place in the network engineer's toolbox. They can simply be described as tools that measure the performance of a network. However, these share few similarities with the tool developed in this thesis since they focus on quantity instead of quality of the generated network traffic.

2.2.3 Realistic Network Traffic Generators

Vishwanath et al. [15] developed a network traffic generator called Swing that generates network traffic that is "statistically similar" to the original traffic. Statistically similar in this sense means that the generated traffic is similar in protocols used, number of bytes sent/received, number of connections and timing of these packets. The major disadvantage of this solution is that Swing must first be trained on the network traffic trace of each application to support, and also that Swing will not adapt well to a changing network environment. Another tool developed with the purpose is SourcesOnOff by Varet et al. [16]. SourcesOnOff is similar to the method proposed in this thesis in that it also generates network traffic timed by the ON/OFF distribution (explained in Section 3.1.1) proposed by Willinger et al. [1], the difference is mainly in how this network traffic is generated.

[2]Source: `https://iperf.fr`, accessed on December 10th, 2019.
[3]Source: `https://github.com/HewlettPackard/netperf`, accessed on December 10th, 2019.

2.3 Simulated Network Devices

2.3.1 Honeypots

A honeypot is a closely monitored network decoy used to detect and deflect attacks on a network. It can either consist of a physical machine or virtual machine (VM), such as the one provided by the Honeyd project by Provos et al. [17]. Regardless the implementation, it must appear to be a legitimate piece of the network so that an adversary would find some value in attacking it, and it should never introduce a risk for the real network. However, since the honeypot has no real production value, all interactions with it can be seen as attacks. It can then be used to deflect the attack by letting the adversary think that he/she has broken in to the real system, and thereby also learning the techniques used by the adversary. Using this knowledge, a network administrator can make sure that the real parts of the network are sound against those kinds of attacks. Honeypots can be classified by their level of interaction:

- A *low-interaction* honeypot is a dummy node that only simulates services that are frequently targeted by attackers, such as an SSH server. What the attacker do not know is that there is no real SSH service, instead all login attempts are denied and monitored. The advantage of low-interaction honeypots is the simplicity of the software and hardware needed to implement them. The previously mentioned Honeyd is a low-interaction honeypot.
- A *high-interaction* honeypot simulates all aspects of an operating system. Instead of just responding to a specific type of service requests, high-interaction honeypots are real systems. This means that high-interaction honeypots can be fully compromised and taken over by an attacker. The extended realism comes with the price of being more time-consuming and expensive to set up and maintain.

2.3.2 Honey Nets

A honey net is simply a network of two or more honeypots. The size of a honey net is only restricted by the administrator's resources and could therefore be used as a decoy not only for a server, but for the whole network. This thesis does not only build on the knowledge of honeypots and honey nets, but also introduces one higher level of interaction. If the first level is to simulate a service, the second is to simulate the whole network stack, then the third one is to simulate the whole system and the users interacting with it. An attacker would then see not only see a passive network of computers, but a network with active users doing actual work.

2.3.3 CRATE

Cyber Range And Training Environment (CRATE)[4] is a project developed and maintained by the Swedish Defense Research Agency (FOI) in Linköping. CRATE is able to simulate a whole city's network infrastructure. Since this requires thousands of VMs, CRATE is running on a supercomputer consisting of some 750 servers [18]. CRATE is nowadays mainly used for research and courses in IT security. The main difference between this thesis and CRATE is scale. While the massive scale of CRATE might be useful for research purposes and for teaching security awareness to companies, it is not possible or necessary for most individuals or small businesses to host thousands of VMs. While CRATE focuses on mimicking the network traffic of a whole city, the method developed in this thesis focuses on generating realistic network traffic from each individual device in a smaller network.

[4]Source: `https://www.foi.se/en/foi/resources/crate---cyber-range-and-training-environment.html`, accessed on December 10th, 2019.

2.4 User Simulation

This section covers some tools for recording and simulating user interaction.

2.4.1 Appmonitor

To simulate something, be that a machine or a user, its behaviors must first be analyzed. Alexander et al. [19] presented a tool called AppMonitor in 2008 that is able to log user activity in unmodified Windows applications. AppMonitor does this by using the Windows SDK library to record both low-level user interactions, such as key presses, and high-level logical actions, such as menu selections. The major drawbacks of AppMonitor compared to the method developed in this thesis are that:

- AppMonitor is not modular and must be extended for each new application it should support.
- AppMonitor can only log user activity and has no replay feature.

2.4.2 LARIAT

LARIAT is a computer security test bench proposed by Rossey et al. [20, 21], that uses statistical models of user behaviour together with an emulation of a wide area network to generate workload. This tool is similar to the one developed in this thesis in that is uses a finite-state machine to simulate user activity on real world applications. The research does not, however, examine the realism of the generated network traffic.

Chapter 3

Background

This chapter describes some techniques used that the reader needs to know about in order to understand the work presented in this thesis. The first section describes network characteristics of real network traffic. How user activity can be simulated by using UI testing frameworks and automatas is described in Sections 3.2 to 3.5. Section 3.6 explains some basic concepts about probability distribution fitting that is used in this thesis. Lastly, Section 3.7 describes the software used to emulate network devices.

3.1 Network Traffic Characteristics

Studies show that network traffic streams are highly-variable and that the aggregation of these streams is self-similar [1, 22]. In order to be able to evaluate the network traffic generated by the method proposed in this thesis, one must first understand what these attributes are and how they are measured.

3.1.1 Variability

When analyzing small sets of network traffic, it is easy to see that network traffic is bursty (variable) in short time scales (see Figure 3.1). A logical assumption would be that the aggregation of shorter network streams would lead to network traffic that gets smoother at larger time scales. Many studies show, however, that real network traffic exhibits high variability, meaning that traffic streams are bursty over a wide range of time scales, and when multiple streams are combined [1]. Crovella et al. [22] showed that bad models of network traffic (such as the Poisson process), however, becomes uniform when a large set of streams are aggregated. This makes variability one good measure of realistic network traffic. High variability in network traffic can be characterized as infinite mathematical variance of the burst ON/OFF periods, where sudden irregularities can always occur. This would suggest that these periods follow some heavy-tailed distribution with infinite variance. ON/OFF periods are simply durations when the network traffic stream is either active or inactive.

3.1.2 Self-Similarity

Another trait that is a common trend among real network traffic is self-similarity. Self-similarity is the property we associate with fractals — objects that appear the same, regardless of the scale at which they are viewed. A common example of self-similar data is the Mandelbrot set, seen in Figure 3.2. In the case of stochastic objects such as time series, self-similarity means that the object's distribution remains the same when viewed at different time scales [22]. Self-similarity is also a cause of a Long Range Dependency (LRD) characteristic, i.e. values at any moment in time are correlated to values at all future moments [22, 23, 24, 25]. Network traffic can be seen to exhibit LRD when $0.5 < H < 1$ and Short Range Dependence (SRD)

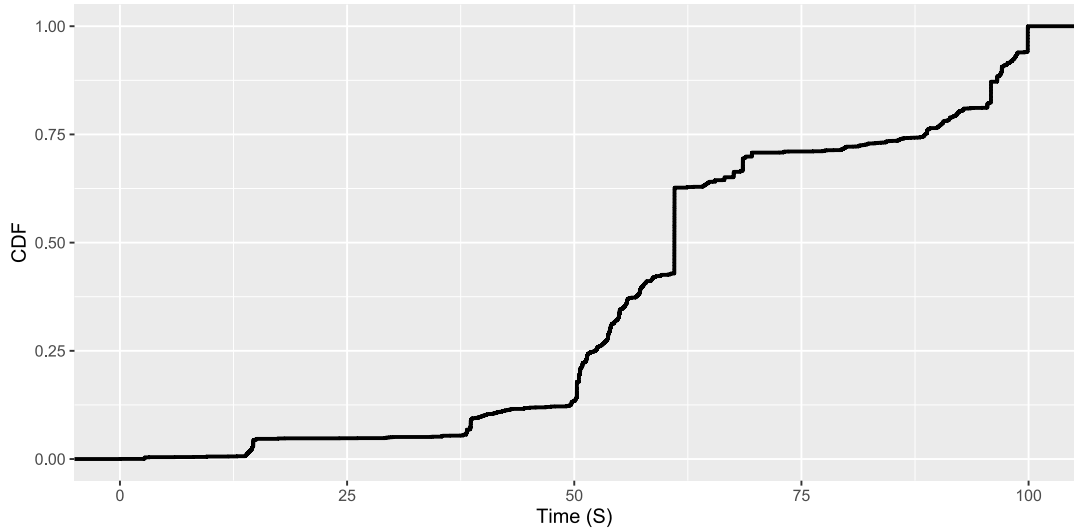


Figure 3.1: Example of 100 s of bursty network traffic.

when $0 < H < 0.5$, where H is the so called Hurst exponent [1, 22]. The Hurst exponent [1] is calculated by

$$\mathrm{E}\left[\frac{R(n)}{S(n)}\right] = Cn^H \text{ as } n \to \infty\,, \tag{3.1}$$

where

$R(n)$ = range of the first n cumulative deviations from the mean,
$S(n)$ = standard deviation,
$E[x]$ = expected value,
n = number of data points in the series, and
C = a constant.

The closer to 1 that H is, the stronger the LRD and the closer to 0 that H is, the stronger the SRD. Therefore, the Hurst exponent is widely used to determine the intensity of the dependence of the network traffic and therefore also its self-similarity. Willinger et al. [1] found that using ON/OFF sources with heavy-tailed distributions caused traffic stream to be highly variable, and an aggregation of these traffic streams to be both highly variable and self-similar.

3.1.3 Time Scale of ON/OFF Periods

Many studies on ON/OFF periods have focused on very short time scales, from a few microseconds up to a few seconds, where self-similarity and high variability have already been identified and reproduced [16]. Because of the self-similarity of network traffic, one can assume that this trait would also exist at longer ON/OFF periods, from a few seconds up to a few minutes, which is the focus in this thesis.

3.2 User Simulation

As mentioned in Section 2.2, there are multiple ways of generating network traffic for the ON periods. This thesis aims to do it by simulating user activity, which can be also be accomplished in multiple ways, at different layers of the OSI model. This section describes two of them, one at either end of the model.

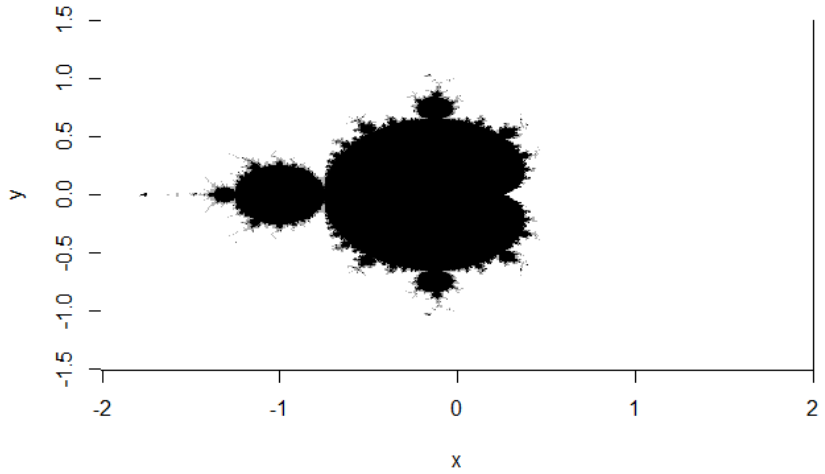


Figure 3.2: Visualization of the Mandelbrot set.

3.2.1 Recording Network Traffic

The first option is recording the network traffic that is produced on a client computer when used by a human. This will capture all output of the client computer, at the chosen layer, but will have no knowledge of what behavior leads to what output. It will also be very difficult to distinguish what output was produced by an action performed by the user and what output that was just a product of the operating system (OS). Therefore, replaying this traffic on a VM could lead to duplicating the OS network traffic.

3.2.2 Recording User Behavior

Another option is to record the input of the user instead of the output. This could also be achieved in several ways. On the lowest level all system calls done by the user could be recorded, and on the other end of the spectrum is recording the high level tasks, which for example could be performed by video recording the user's display [19]. There is, however, a recently developed tool called WinAppDriver UIRecorder[1]. UIRecorder is able to identify which and how Windows UI elements are interacted with by the user in real time. UIRecorder could, since it is open source, be altered to save this information and therefore be used to record user behavior.

3.3 Windows UI Automation

To simulate a user's interaction with the Windows User Interface (UI), three things must be answered: *what*, *how* and *when?* I.e., what element should be interacted with, how should it be interacted with, and when should it occur? This section covers how these questions can be answered.

[1]Source: `https://blogs.windows.com/windowsdeveloper/2018/06/20/introducing-winappdriver-ui-recorder/`, accessed on December 16th, 2019.

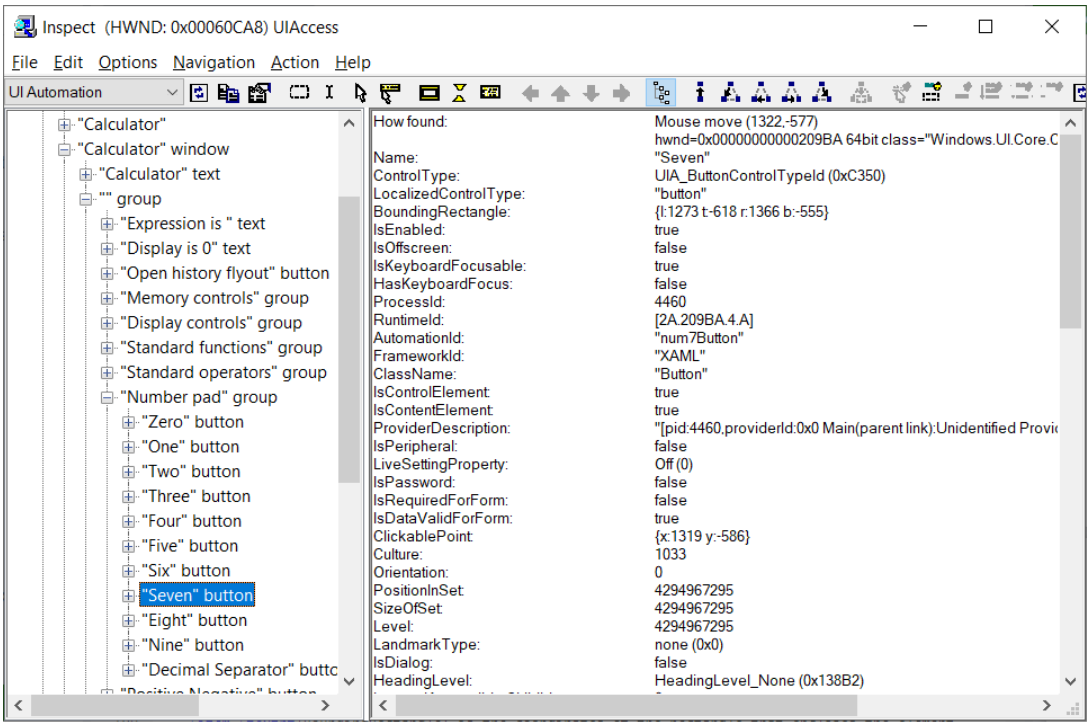


Figure 3.3: The Windows UI element inspection tool *(Inspect.exe)*.

3.3.1 Identifying UI Elements

The first step to user simulation is giving the simulation software some knowledge about its surroundings. In later versions of Windows, all UI elements have a set of attributes that can be used for identification. These can be viewed in tools such as Microsoft's own Inspect (see Figure 3.3, which comes with the Windows SDK. There are a few options to choose from when identifying a graphical element, here are a few of them, using the Windows Calculator (see Figure 3.4) as an example:

- **Name** is the name of the button, this is often the same as the text of the button, in this case `Seven`. This does not guarantee any kind of uniqueness and might be different depending on local language of the OS.
- **AutomationId** is an identifier that is unique in that level of the UI tree, and is the same for every instance of the application. It does not have to be unique throughout the whole UI tree, and since it is set by the developer of the software, it is not always available. For example, if two calculators where running at the same time, two elements would have the same AutomationId. In this example the AutomationId of the button is `num7button`.
- **RuntimeId** is a unique identifier of the UI element, but it is only guarantied to not change during the lifetime of the element. If the window is closed and opened again, the same element might have a new RuntimeId. The `Seven` button can for example have the RuntimeId of `[2A.209BA.4.A]` at one instance, and `[2A.203B6.4.A]` the next time the calculator is opened.
- **ControlType** is a class that identifies the type of the element. This lets the developer know what kind of actions that are available for the specified element. It, however, only works if the element is a standard Microsoft UI control. The example element has the ControlType `UIA_ButtonControlTypeId`.

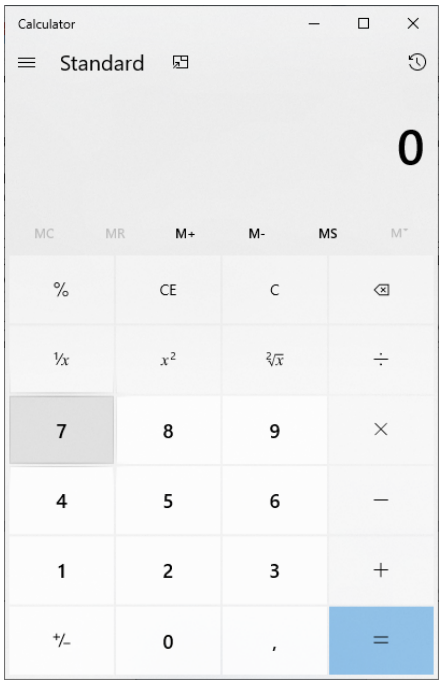


Figure 3.4: The Windows 10 calculator *(calc.exe)*.

- **ClassName** is similar to ControlType, but also works for non-standard UI controls. It instead returns the class name of the element that is assigned by the control developer. Therefore, it will always return some value although it might not be of standard format. The example element has the ClassName `Button`.
- **BoundingRectangle** is the coordinates of the rectangle that encloses the element. Although some simple automation tools uses coordinates to locate elements, this is very unstable. The desired element would not be found if for example the window was moved, or if another window was in front of it. The rectangle can also contain points that are not clickable, if the element is not exactly rectangular. An example of a BoundingRectangle might be `l:1273 t:-618 r:1366 b:-555`.
- **ClickablePoint** is a point of the BoundingRectangle that is clickable. It has the same drawbacks as the BoundingRectangle but will at least work on non-rectangular shaped elements. In this example it is equal to `x:1319 y:-586`.

3.3.2 XPath

Even though UI elements has a lot of properties, none of them can act as a system wide unique identifier that will remain the same though different sessions. Therefore, one such identifier must be created. This is where the XML Path Language (XPath) comes in. XPath is described by the World Wide Web Consortium [26] as "a language for addressing parts of an XML document". It is commonly used to identify an object in HTML, but since the Windows UI tree can also be described by a XML document, it can be used here too. To do this, XPath uses a combination of the properties mentioned in Section 3.3.1 to create an absolute (starting at the system root) and unique path to the element. Because XPath is absolute, it can be used to find an element even if it is hidden behind e.g. another window. The example below represents the XPath to a UI element. Worth to mention is that since the XPath contains the Name property, the XPath to an element might be different on two machines with different local languages.

```
/Pane[@ClassName=\"#32769\"][@Name=\"Desktop 1\"]
/Window[@ClassName=\"ApplicationFrameWindow\"][@Name=\"Calculator\"]
/Window[@ClassName=\"Windows.UI.Core.CoreWindow\"][@Name=\"Calculator\"]
/Group[@ClassName=\"LandmarkTarget\"]
/Group[@Name=\"Numeric keypad\"][@AutomationId=\"NumberPad\"]
/Button[@Name=\"Seven\"][@AutomationId=\"num7Button\"]
```

3.3.3 Interaction Types

The next task is to answer the question of *how* the element should be interacted with. Windows has a standard set of interaction types when using a keyboard and/or a mouse that can be classified as follows:

- **KeyboardInput** is the combination of KeyDown and KeyUp events.
- **MouseHover** is the event that occurs when the mouse cursor is placed on top of an element.
- **LeftClick** left mouse button down, followed by left mouse button up.
- **RightClick** right mouse button down, followed by right mouse button up.
- **LeftDblClick** consists of two LeftClick events, usually within 500 milliseconds of each other[2].
- **Drag** is when the left mouse button is pressed followed by mouse movement.
- **DragStop** is the opposite to Drag, i.e. mouse movement followed by letting go of the left mouse button.
- **MouseWheel** is the event that is triggered when the mouse wheel is rotated, also called a scroll.

3.3.4 Interaction Timing

The last thing to decide is *when* the simulation software should trigger events on the UI elements. This could either be performed as fast as possible, after a specified time or at the same speed as the recording. The problem with trying to trigger events too quickly is that it might not always be possible and/or might lead to unrealistic interactions.

3.4 Windows Automation Frameworks

Multiple Windows automation frameworks exist that makes it possible to interact with the OS. This section covers the two of them that were considered in this thesis.

3.4.1 WinAppDriver

WinAppDriver[3], short for Windows Application Driver, is a software developed by Microsoft that utilizes Selenium's WebDriver protocol[4] for UI testing of Windows applications. It supports *Universal Windows Platform (UWP), Windows Forms (WinForms), Windows Presentation Foundation (WPF)*, and *Classic Windows (Win32)* apps on Windows 10 PCs. In order

[2]Source `https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setdoubleclicktime`, accessed on December 10th, 2019.

[3]Source: `https://github.com/microsoft/WinAppDriver`,
accessed on December 10th, 2019.

[4]Source: `https://selenium.dev/documentation/en/webdriver/`, accessed on December 10th, 2019.

for WinAppDriver to work, the machine must run in Developer Mode[5]. WinAppDriver then acts as a server, listening for WebDriver requests on port 4723. WinAppDriver uses XPath and the interaction types mentioned above in order to interact with the OS.

3.4.2 AutoIT

AutoIT[6] is a scripting language used to automate the Windows Graphical User Interface (GUI) [27]. AutoIT was considered, but dissmissed in favor of WinAppDriver because it is developed by Microsoft, whom also developed the OS, instead of a third party, and that this software can be interacted with using any programming language.

3.5 Automatas

The automation frameworks mentioned in Section 3.4 require some logical structure that decide in which order to interact with the UI elements. One option is to use an automata [28]. Automatas are abstract models of machines that acts upon input in order to move around a state-space. These automatas are divided into four classes, making up the Chomsky Hierarchy [29]. They are divided in order of complexity, where the Finite-State Machine (FSM) is the simplest, succeeded by the Pushdown Automata (PDA), Linear-Bounded Automata (LBA) and lastly, the Turing Machine (TM). It is important to notice that a more powerful automata can do everything that a simpler automata can, but the extra complexity is not always needed. The automaton used to simulate user activity in this thesis can be classified as a (non-deterministic) FSM, described in detail below.

3.5.1 Finite-State Machines

Finite-state machines can be described by a quintuple [Σ, S, δ, s_0, F] where:

- Σ is the (finite, non-empty) input alphabet.
- S is the (finite, non-empty) set of states.
- δ is state transition function, $\delta : S \times \Sigma \rightarrow \mathcal{P}(S)$. $\mathcal{P}(S)$ is a set of states, on a Deterministic Finite-state Automata (DFA) this would only be one state.
- s_0 is the start state, an element of S.
- F is a (possibly empty) set of final states, a subset of S.

3.5.2 Determinism

A FSM can either be a deterministic finite automata (DFA) or non-deterministic (NFA). A deterministic automaton has exactly one transition for each possible input. An input in a non-deterministic automaton can lead to one, multiple, or no transitions at all. One can clearly see that the automata in Figure 3.5 is an NFA because q_2 can go to both q_1 and q_3 on input 0. Since NFAs and DFAs has the same complexity, a NFA can always be converted into an equal DFA as shown in Figure 3.6.

[5]Source: `https://docs.microsoft.com/en-us/windows/uwp/get-started/enable-your-device-for-development`, accessed on December 10th, 2019.

[6]Source: `https://www.autoitscript.com/site/`, accessed on December 10th, 2019.

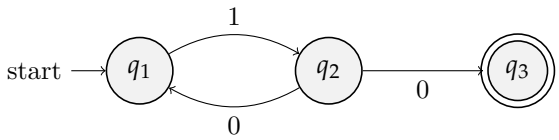


Figure 3.5: A basic NFA.

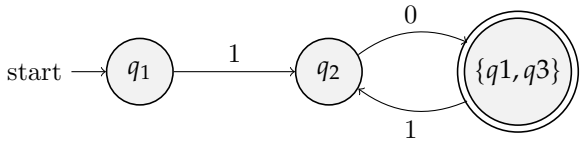


Figure 3.6: A basic DFA equivalent to Figure 3.5.

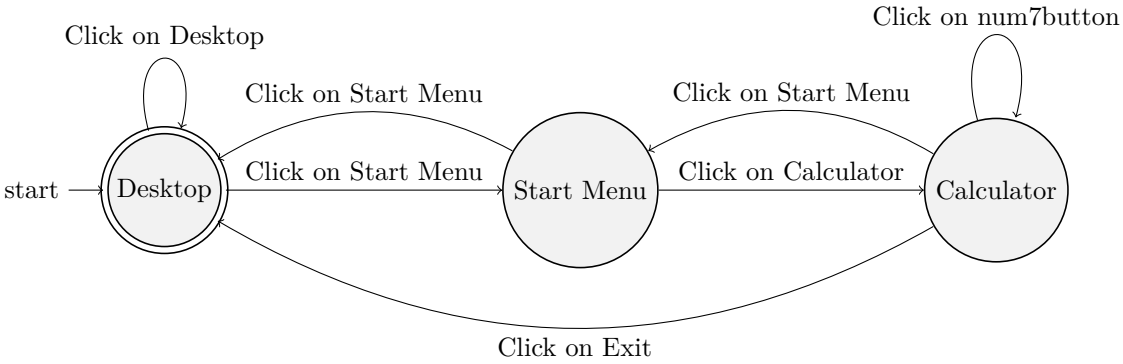


Figure 3.7: A FSM describing a basic task in Windows.

3.5.3 Defining Windows UI Navigation as a FSM

Windows user interface navigation can be modeled by a non-deterministic FSM where:

- Σ is defined by possible user input.
- The states in S are defined by the different windows presented on the screen.
- δ is the mapping between a state, an input and which states it can lead to.
- s_0 is the first state that the user is presented to.
- F can be any state in S.

Defining the whole Windows UI state-space would result in an infinitely large FSM, but smaller UI traversals can be described quite easily, Figure 3.7 is an example of this. The reason why the FSM in this thesis needs to be non-deterministic, is that the same input at the same start state might lead to different end states depending on external factors such as time, network connection or available disk space. These factors could be included in the states, in order to make the FSM deterministic, but that would probably increase the complexity significantly.

3.5.4 Traversing the FSM State-Space

When a large enough set of states and transitions is obtained, the FMS state-space can be traversed, either at random or by a set of rules, in order interact with the environment and simulate user activity. This state-space traversal is visualized in Figure 3.8.

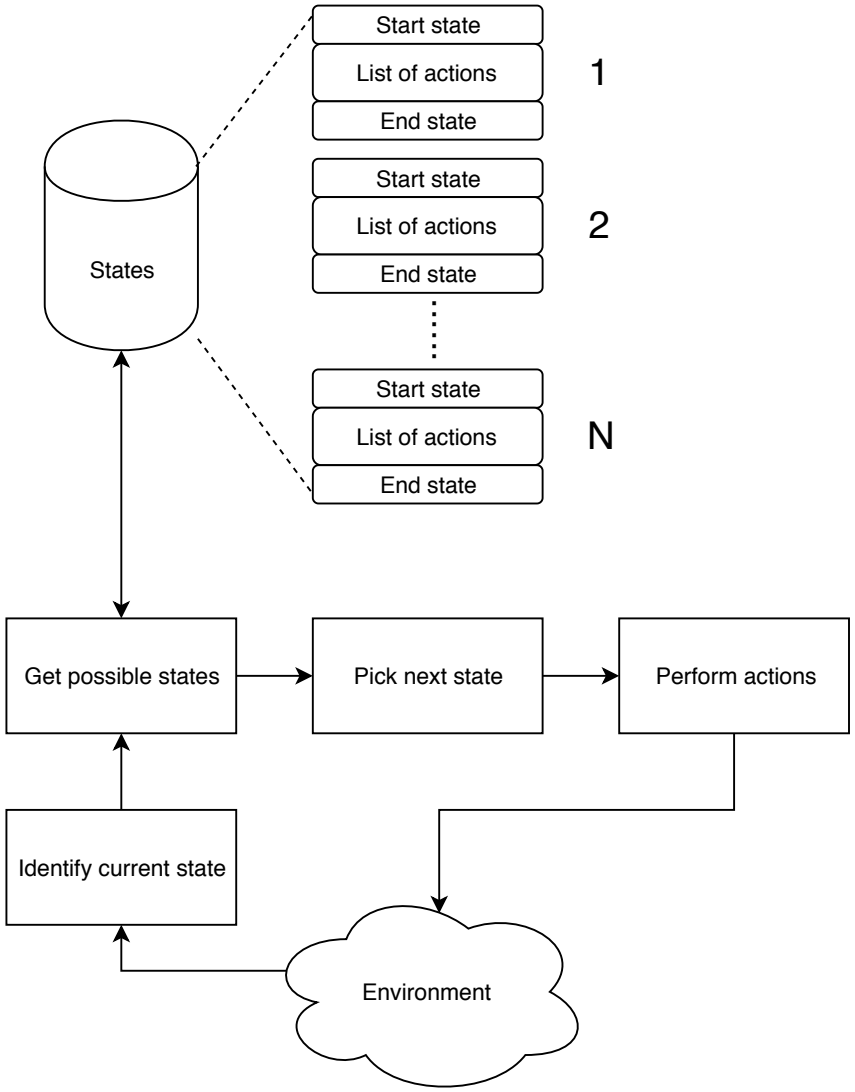


Figure 3.8: User activity simulation by FSM state-space traversal.

3.6 Probability Distribution Fitting

The goal of probability distribution fitting is to identify the population that is most likely to have generated the observed data sample [30]. When trying to fit a probability distribution to a series of observed data, a common practice is to start by looking at the measured data's symmetry in order to determine which distribution it most closely correlate to. After the probability distribution type is determined, one can use a parametric method in order to estimate what the parameters of the chosen distribution should be set to in order to get a maximum fit to the observed values. This process is described in greater detail below.

3.6.1 Visual Observation

Probability distributions can be divided into categories depending on their characteristics. The characteristic that is in focus in this thesis is mainly the skewness of the probability distribution. The data can either be symmetrical, positively skewed or negatively skewed. If the data in a distribution for example is positively skewed, it means that the large values are far away from the mean, and the mean is to the right of the median. Figure 3.9 shows a

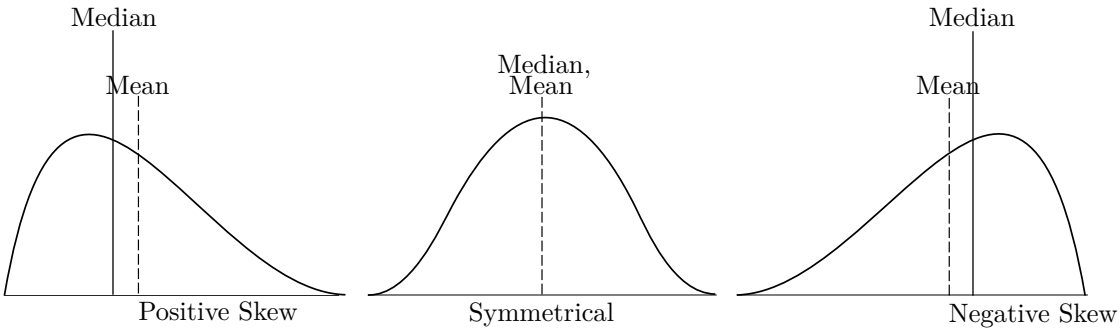


Figure 3.9: Visual comparison between distribution symmetry.

comparison between these three categories. These categories then map to a number of specific distributions, which can further be distinguished by some other attributes such as if they are discrete or continuous. The Exponential, Gamma and Weibull distributions are all examples of positive skewed distributions. ON/OFF periods are expected to come from positively skewed distributions with a lower limit of 0, since time can not be negative.

3.6.2 Parameter Estimation

Once the visual observation is complete and one or a few distributions have been chosen, one is in a position to evaluate their goodness of fit, meaning how well the distribution fit the observed data. The method for parameter estimation conducted in this thesis is called Maximum Likelihood Estimation (MLE) [30]. In MLE, the parameters are found such that they maximize the likelihood that the process described by the model produced the observed values. This is accomplished by first observing the Probability Density Function (PDF) of the model. Each distribution has its own equation for calculating the PDF. The normal distribution for example has two parameters, the mean μ and the variance $\sigma^2 > 0$. The probability density of observing a single data point x generated by a normal distribution is given by the likelihood function

$$P(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

The total probability density for *all* observed values $x_1, \ldots, x_n$ is then given by

$$P(x_1, \ldots, x_n \mid \mu, \sigma^2) = \prod_{i=1}^{n} \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)\right).$$

The maximum likelihood is then estimated by figuring out which parameters μ and σ^2 that maximizes $P(x_1, \ldots, x_n \mid \mu, \sigma^2)$. This is solved by simply differentiating the function and extracting the maximum value. Since this function is often difficult to differentiate, it is often simplified by differentiating the natural logarithm of the function instead, which is okay since the natural logarithm is a monotonically increasing function, resulting in the same maximum likelihood.

3.6.3 Fit Evaluation

When the parameters for the chosen models are estimated, one can compare the error between the different models to determine which model fits the observed data the best. This can be achieved by calculating their Root Mean Square Error (RMSE), calculated as follows

$$RMSE = \sqrt{(\frac{1}{n}) \sum_{i=1}^{n} (y_i - x_i)^2}, \tag{3.2}$$

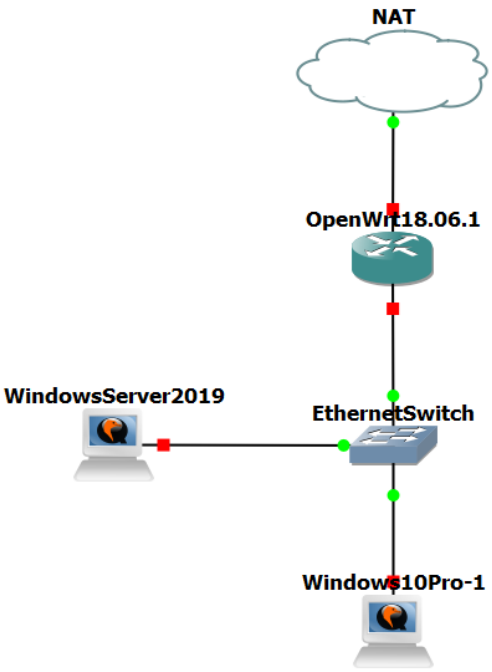


Figure 3.10: A sample project in GNS3.

where

n = sample size,
x_i = observed values (known results), and
y_i = forecasts (expected values).

The RMSE is the statistical distance between the observed values and the model, therefore a lower RMSE is better. It is important to note that RMSE is scale dependent, meaning that it is only useful for comparing errors between different models of one data set, and not between data sets. RMSE is also very sensitive to outliers since the effect of each error is proportional to the size of the squared error. [31]

3.7 GNS3

Graphical Network Simulator 3[7] (GNS3) is a network software emulator tool that lets its users "drag and drop" network appliances into a grid and connect them with virtual network cables. Figure 3.10 shows a small sample project in GNS3 containing some basic appliances. A model of a small network can be designed, created and tested out in minutes. It also makes the environment easy to contain, since all network appliances run inside a VM. This VM can easily be run remotely on more powerful hardware, and resetting the environment to a saved state is easy, which is good for testing purposes, or if the network is to be used as a honey net. GNS3 also offers the ability to clone devices in a project. This means e.g. that a machine can be set up with all the necessary software installed and configured and then one could clone it into 10 machines, and not have to go through the setup process again.

[7]Source: `https://www.gns3.com/`, accessed on December 12th, 2019.

Chapter 4

Methodology

4.1 Analysis of Real Network Traffic

Before any network traffic could be generated, some real network traffic had to be gathered in order to have something to reproduce synthetically. All data sets found available online were either not large enough or had a too specific use case (such as only a particular protocol recorded) to be used in this thesis, therefore the decision was made to record some network traffic.

4.1.1 Recording Network Traffic

The network traffic model was created by capturing all the incoming and outgoing traffic of a laptop during a typical work week, from around 8am to 5pm, but including interruptions such as lunch breaks. The reason for not including the long intervals between each work day was because several-hour long off periods where not wanted in the simulations. These large capture files was then stripped down to only contain information such as Internet Protocol (IP) source address, IP destination address, timestamp, length (in bytes) and protocol, since no other information was needed. These files where then concatenated after each other, such that it would appear as if the first packet in the second capture file would arrive just after the last one in the first file.

4.1.2 Determining ON/OFF Periods

Since the traffic throughput of a computer on a network is almost never at zero (one experiment showed that only opening and using the Windows Calculator for 60 s generated around half a megabyte of network traffic), a throughput threshold must be chosen in order to determine the length of the ON/OFF periods. The threshold was set at the median amount of transmitted packets per minute. This way the threshold scales based on the background noise of the OS. Every minute of the network traffic was then categorized as ON if its throughput were higher than the threshold and as OFF if it was equal to or lower than it. Figure 4.1 shows a section of the categorized traffic, where green dots represents values above the threshold and red dots represents values at or below the threshold. Then the length of each ON/OFF period was determined by splitting the data where it went from ON to OFF and vice versa, as can be seen in Figure 4.2. This then resulted on two sets of observed distributions, one for the ON periods and one for the OFF periods, which can be seen in Figure 4.3.

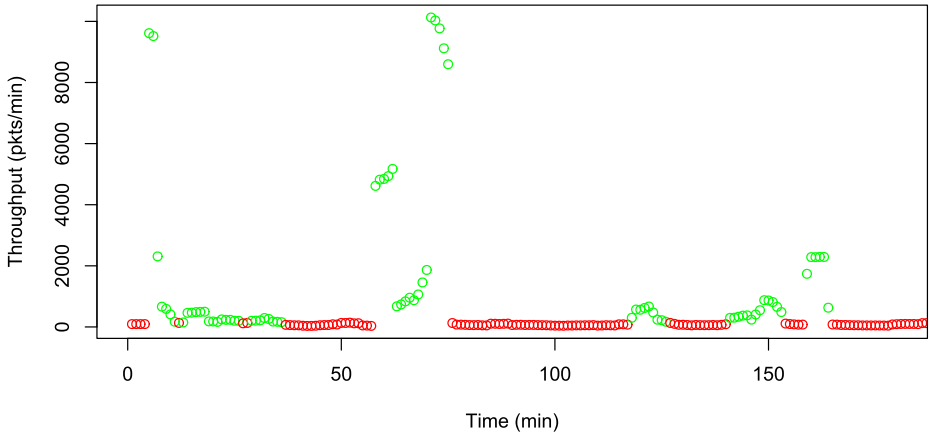


Figure 4.1: Network throughput thresholding.

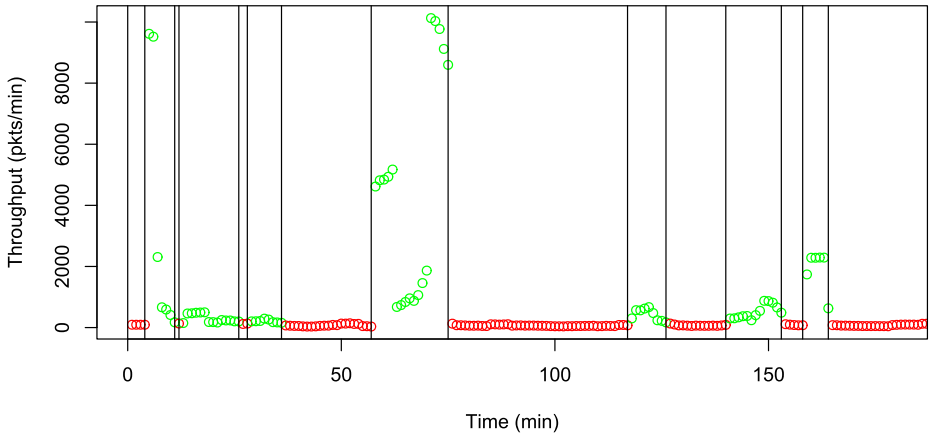
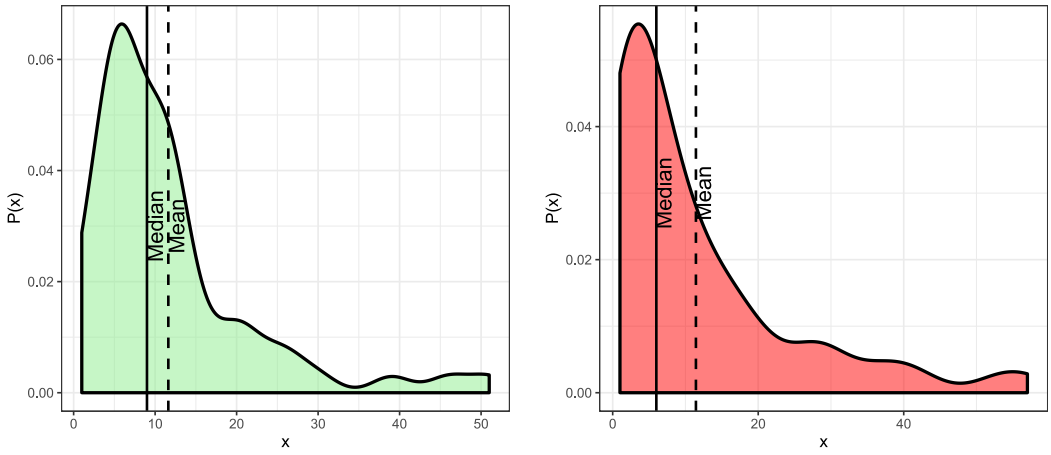


Figure 4.2: Captured traffic divided into ON/OFF periods.



(a) ON periods.

(b) OFF periods.

Figure 4.3: Probability density of ON/OFF period lengths.

4.1.3 Probability Distribution Fitting of ON/OFF Periods

Visual Comparison

As mentioned in Section 3.6, the first step towards determining the theoretical distribution closest to the observed values is a visual comparison. One can clearly see in Figure 4.3a and Figure 4.3b that the observed distributions are slightly different, but that both have a positive skew. Because of this, symmetrical and negative skewed distributions are ignored in the selection. The following theoretical distributions where tested: **Weibull**, **Gamma**, **Exponential**, **Geometric** and **Poisson**. The **Pareto** distribution was initially also viewed as an option, but it resulted in extremely large values which made it unusable. The Pareto distribution is known for placing a large portion of the probability mass in the tail, when the sample size is small [22].

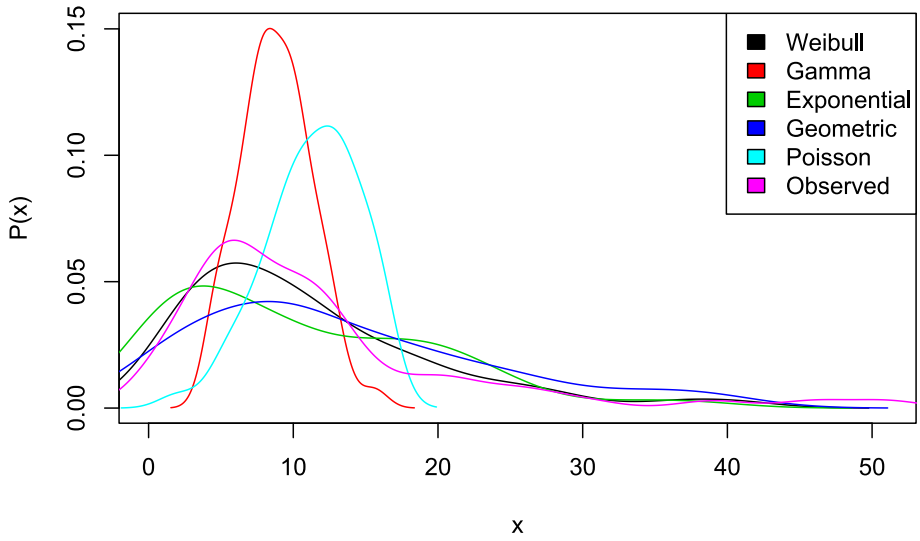
Parameter Estimation

The parameters of the different theoretical distributions where then estimated using the fitdistr[1] function in the programming language R. Fitdistr uses the MLE method in order to estimate the different parameters of the theoretical distribution. Figure 4.4a and Figure 4.4b shows the fitted theoretical distributions compared with the observed values.

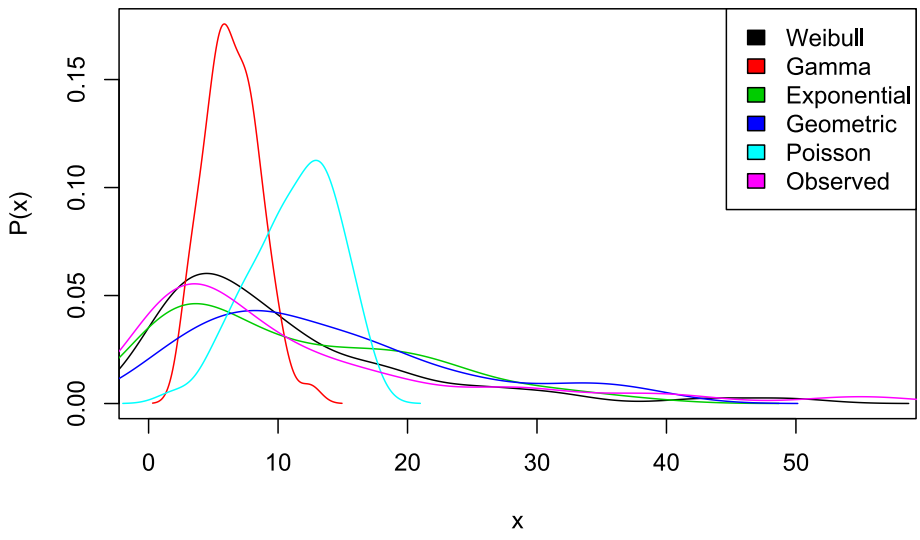
Fit Evaluation

To evaluate the distributions more precisely, their RMSE in relation to the observed distribution where calculated and compared. As mentioned in Section 3.6, a lower RMSE means that the distribution fits better to the observed values. By looking at Figure 4.5, one can see that both the ON and OFF values are best represented by the Weibull distribution (with different parameters, see Table 4.1). The quantile-quantile plots (Q-Q plots) in Figure 4.6 shows a comparison between the observed values (dots) and the fitted Weibull distribution (line). Only a quick visual observation is needed to see that most of the values intersect, which was considered "good enough" for the purpose of this thesis.

[1]Source: `https://www.rdocumentation.org/packages/MASS/versions/7.3-51.4/topics/fitdistr`, accessed on December 12th, 2019.

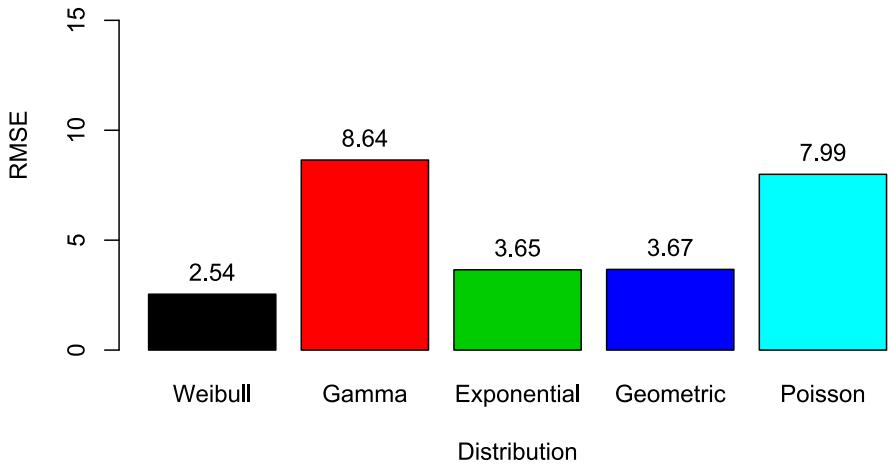


(a) ON periods.

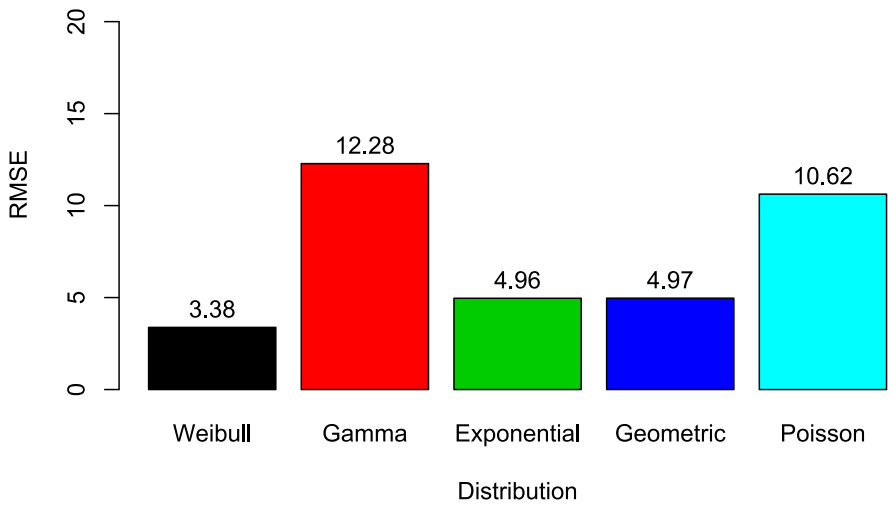


(b) OFF periods.

Figure 4.4: Probability density of ON/OFF period lengths compared with fitted theoretical distributions.



(a) ON periods.

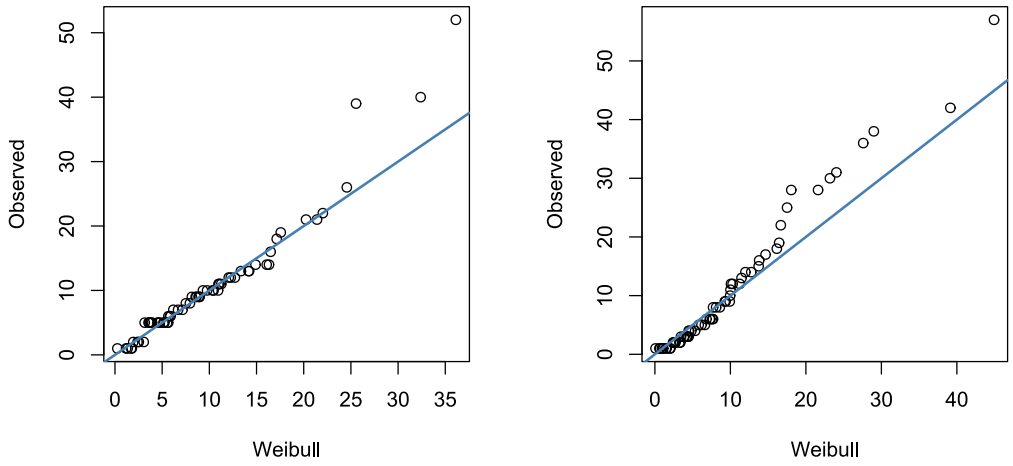


(b) OFF periods.

Figure 4.5: RMSE comparison of fitted theoretical distributions (lower is better).

	Shape	**Scale**
ON	1.26	11.36
OFF	1.00	10.42

Table 4.1: Estimated parameters of ON/OFF Weibull distributions.



(a) ON periods.

(b) OFF periods.

Figure 4.6: Q-Q plots for fitted ON/OFF distributions. If both sets of quantiles came from the same distribution, the points would form a line that is roughly straight, meaning that the distributions in these plots are very similar.

4.2 Simulations

This section explains in detail how the tools used for the simulations worked and how the simulations where executed.

4.2.1 User Recording Tool

The tool used to record user activity was a modified version of the WinAppDriver UI Recorder. The modifications made to this tool was the ability to save recorded user interactions to a JavaScript Object Notation (.json) file representing a FSM. This made it possible to record and save scenarios which could then be executed over and over again in another application, described in Section 4.2.2. Recorded scenarios consisted of typical user tasks such as browsing the web and sending e-mail. The true realism of these scenarios where, however, not taken in to consideration since it was not the purpose of the thesis, but rather a whole other study in itself. The recording tool also saved the time between steps, so that it could be replayed at the same speed, which was important in order to take things such as website buffering and app startup times into account. Below is a representation of states and actions of one of the recorded scenarios.

1. [**Desktop**]: LeftClick on StartMenuButton
2. [**Start Menu**]: Type in “mail”
3. [**Start Menu**]: LeftClick on Mail App
4. [**Windows Mail App**]: LeftClick on NewEmailButton
5. [**New E-mail Window**]: Type “example@example.com” in ReceiverForm
6. [**New E-mail Window**]: Type “test” in SubjectForm

7. [**New E-mail Window**]: Type “testing testing” in MessageForm
8. [**New E-mail Window**]: LeftClick on SendButton
9. [**Windows Mail App**]: LeftClick on CloseButton

4.2.2 User Replay Tool

The program for executing scenarios then took a set of .json files and a seed for the Pseudo Random Number Generator (PRNG) as input. A brief outline of the main loop of the program can be seen in Listing 1. The ON and OFF distributions where two separate Weibull distributions with the parameters identified in Section 4.1.3. The OFF method simply consisted of a sleep instruction, during which the VM only generated background OS traffic. The ON method choose scenarios from the .json files at random (based on the seed) and executed them over and over again until timeout. A snapshot were created on the VM before the simulations in order to make the test environment as similar as possible between sessions. The simulations consisted of four five-hour-long sessions with different seeds. One additional five-hour-long session with no simulation was also conducted for comparison. The enumeration below lists a brief outline of the performed simulations.

1. Preparation
    a) Revert VM to snapshot
    b) Start simulation software
    c) Load scenario files and set PRNG seed
2. Execution
    a) Send *start* packet
    b) Let simulation run for 5 hours
    c) Send *stop* packet
3. Result collection
    a) Turn off VM
    b) Collect generated .pcap file
    c) Remove all packets before *start* packet and after *stop* packet

4.2.3 Network Traffic Filtering

The VirtualBox VMs where configured to generate a .pcap file of all incoming and outgoing network traffic, which was then stored on the host machine after each session. This made it possible to exclude all network traffic from the host that occurred during the sessions. The Start and Stop packet mentioned in the enumeration above, were packets that could easily be identified in the .pcap file. This made it possible to remove all traffic that might have been transmitted before and after the simulations.

4.3 Measurements

The results in this thesis were only based on the outgoing network traffic. This was justified by the fact that the amount of ingoing and outgoing packets were strongly correlated, probably because of the request/response nature of the Transmission Control Protocol, which accounted for around 98 % of the generated network traffic. The incoming traffic was examined but it will not be presented in this thesis, since there was no significant difference compared to the outgoing network traffic.

```
1   void main(){
2       while(true){
3           onTime = onDistribution.next();
4           offTime = offDistribution.next();
5
6           on(onTime);
7           off(offTime);
8       }
9   }
10
11  void on(onTime){
12      while(elapsedTime < onTime){
13          runRandomTask();
14      }
15  }
16
17  void off(offTime){
18      Sleep(offTime);
19  }
```

Listing 1: Psuedo code for ON/OFF loop.

4.3.1 Variability

The variability of the generated traffic ON/OFF periods where evaluated by comparing the generated ON/OFF period distribution to the observed ON/OFF period distribution. If the generated and observed distributions where of the same type, their parameters could also be compared.

4.3.2 Self-Similarity

As mentioned in Chapter 3, self-similarity occurs when there are multiple network streams generating traffic with high variability at the same time. The simulations where, however, run one at a time after each other for performance reasons. So in order to measure the self similarity, all capture files where first cut into the same length, and then every N seconds of the capture where set to 1 if was an ON period, and to 0 if it was an OFF period. Figure 4.7 show how the output of the streams where aggregated, and Figure 4.8 show an example of the output from this.

4.4 Putting It All Together

When the characteristics of the generated network traffic had been examined and evaluated, the last step was to put everything together in order to answer **RQ 1**. This was achieved using the tool GNS3 described in Section 3.7. The VM used for the earlier experiments was imported into GNS3 and cloned into multiple clients. One could then give them different input, in order to make them behave dissimilar. More devices, such as a Windows Active Directory (AD) server and a OpenWRT router, was then added with the purpose of creating a more realistic network topology. A snapshot was then created which enabled the possibility of reverting the simulated network to a saved state. The full workflow of synthetically generating realistic network traffic would then be:

1. Record user activity with the user recording tool, and save this information to scenario files.

Stream 1	Stream 2	Stream 3	Stream 4		Aggregated Stream
1	0	1	1	=	4
1	0	0	1	=	2
0	1	0	1	=	2
0	1	1	0	=	2
0	1	1	0	=	2
1	0	1	1	=	3
0	1	0	1	=	2
1	1	1	1	=	4
1	0	0	0	=	1
1	0	1	0	=	2
.	.	.	.		
.	.	.	.		
.	.	.	.		

Figure 4.7: Example of how the aggregated stream was calculated.

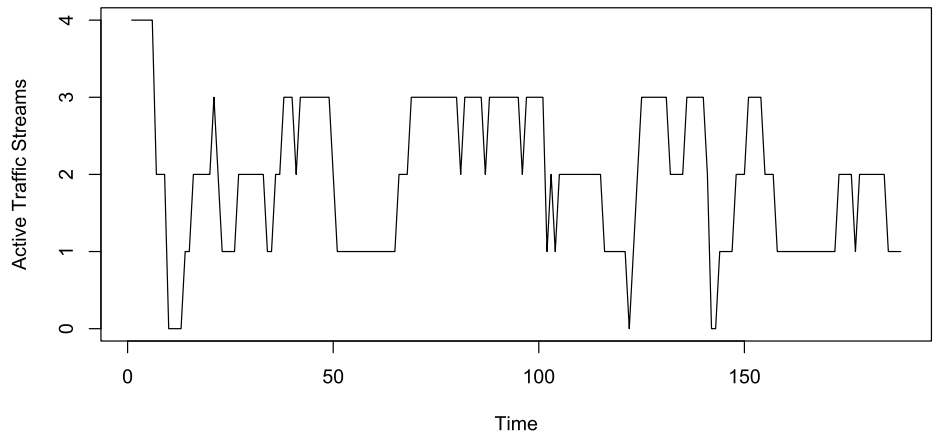


Figure 4.8: Visualization of the number of active streams at any time, in a network with 4 nodes.

2. Design and build a network of simulated devices in GNS3.
3. Load the scenario files onto the Windows machines, set different PRNG seeds for each machine and start the user replay tool.
4. Choose where in the simulated network topology to tap into the network traffic, and record it using Wireshark.

Chapter 5

Result

This chapter evaluates the method presented in this thesis by reconnecting to the aim of generating network *realistic* traffic in an *automatic* and *controllable* manner. This chapter thereby also seeks to answer the research questions presented in Chapter 1.

5.1 Realism

The generated network traffic exhibits both self-similarity and high variance, even though the simulations in total was only half as long as the observations of real network traffic. These two characteristics are evaluated in detail below.

5.1.1 Variance

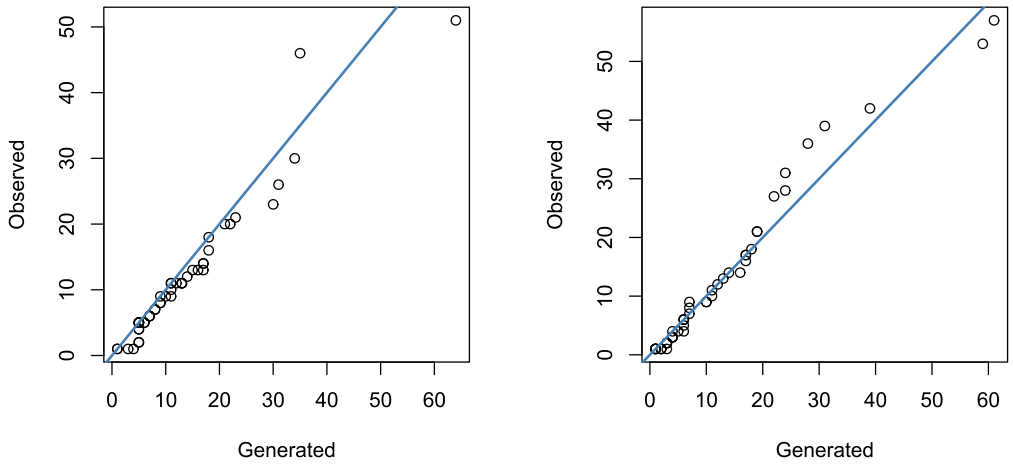
The generated network traffic ON/OFF periods successfully follows a Weibull distribution similar to the one generated by the observed network traffic, with a RMSE of 3.39 for the ON period distribution and 2.82 for the OFF period distribution. Tables 5.1 and 5.2 show the difference of the two Weibull parameters. The generated Shape parameter is very close (within 5.26 % of the observed Shape), while the generated Scale parameter is a bit off, at most 20.56 % from the observed one, meaning that the slope of the PDF is slightly stretched out, generating a few more long ON/OFF periods. These outliers can be seen in Figure 5.1. The generated network traffic has a mean of 13.06 min for the ON periods and 12.31 min for the OFF periods, and a median of 10.50 min and 7.00 min, respectively (see Figure 5.2).

	On (observed)	**On (generated)**	**Difference**
Shape	1.26	1.33	5.26 %
Scale	11.36	14.30	20.56 %

Table 5.1: Weibull parameters comparison of ON distribution.

	Off (observed)	**Off (generated)**	**Difference**
Shape	1.00	1.01	0.99 %
Scale	10.42	12.35	15.63 %

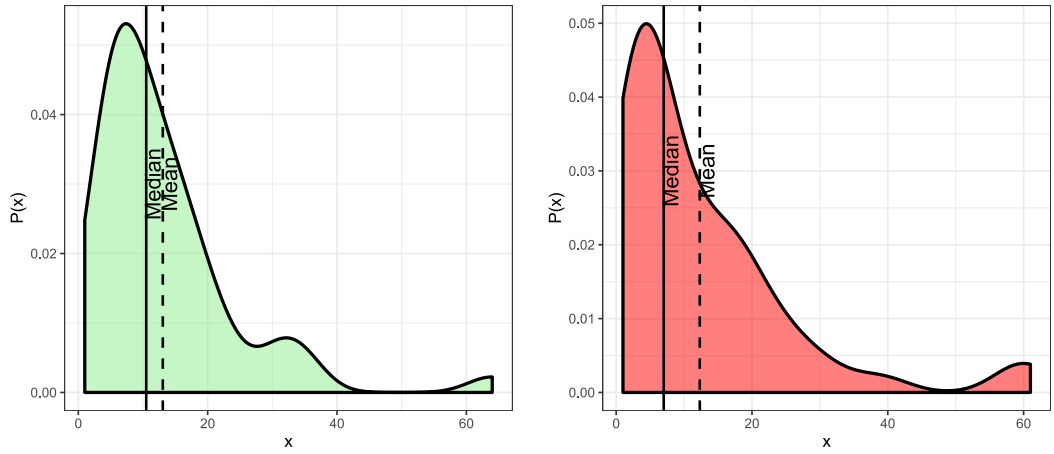
Table 5.2: Weibull parameters comparison of OFF distribution.



(a) ON distribution fit.

(b) OFF distribution fit.

Figure 5.1: QQPlot for observed vs. generated ON/OFF values.



(a) Generated ON periods.

(b) Generated OFF periods.

Figure 5.2: Probability density of generated ON/OFF period lengths.

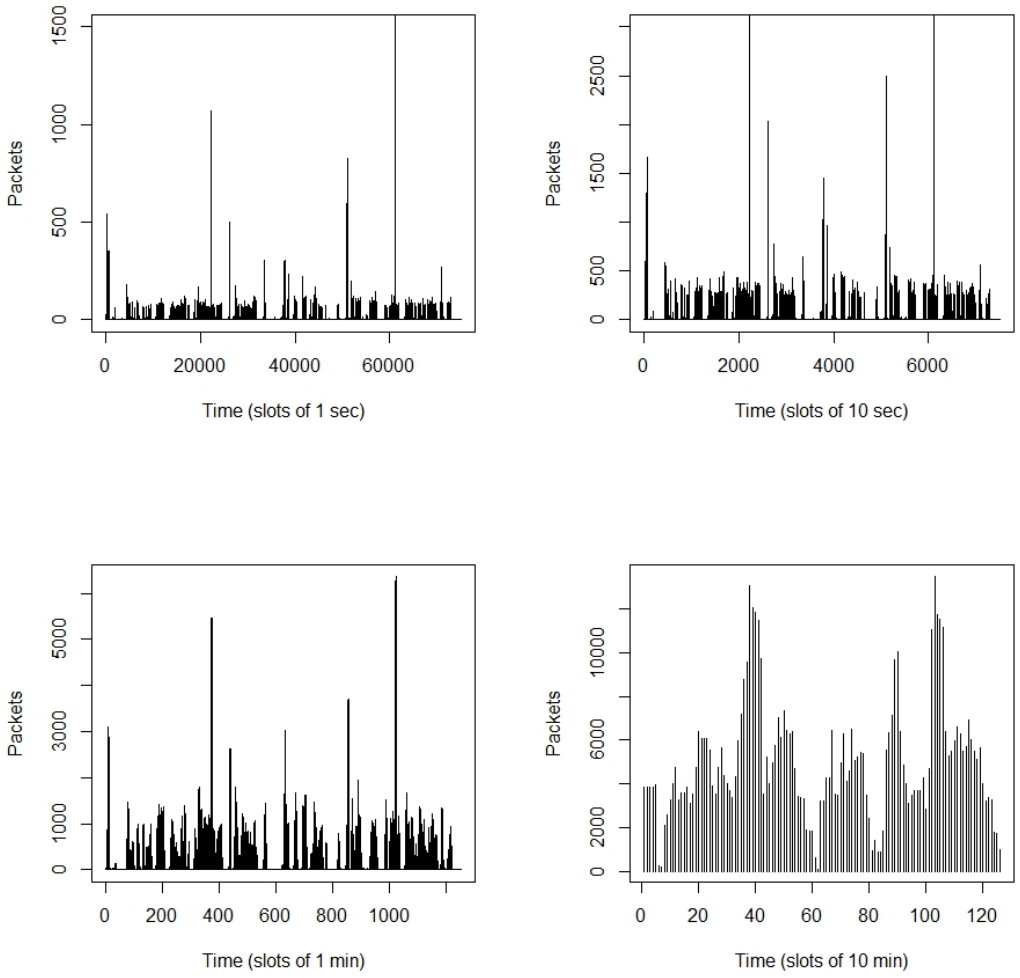


Figure 5.3: Traffic bursts over four orders of magnitude from 1 s to 10 min.

5.1.2 Self-Similarity

As can be seen in Figure 5.3, the generated network traffic is still bursty over a wide range of time spans. This burstiness starts to degrade first at sample lengths of 10 min, probably because the large sample lengths leads to a small data set. As mentioned in Section 3.1.2, network traffic exhibits LRD and therefore also self-similarity when $0.5 < H < 1$, where H is the Hurst exponent. This is the case for all test results (see Table 5.3), but again, it starts to become slightly less bursty at longer sample durations. The median number of active streams is half of them, which is expected since the difference in time between total ON and OFF periods of the streams is $< 6\,\%$.

5.2 Automatic & Controllable

The VMs in GNS3 can be set to run the user simulation automatically at startup, with a preconfigured state-space for the Windows UI traversal. The ON/OFF values will be drawn at random from the predetermined distributions. Therefore, the method presented in this thesis will generate network traffic automatically once the configuration is complete. The

Sample duration for throughput measurements	H Observed	H Generated	Difference
1 s	0.79	0.77	2.53 %
10 s	0.80	0.71	11.25 %
1 min	0.70	0.66	5.71 %
10 min	0.70	0.51	27.14 %

Table 5.3: Hurst exponent estimations (H).

VMs in GNS3 can also be easily controlled from the GNS3 interface. VMs can be added and removed at runtime and all network traffic going to and from can be inspected live by an administrator.

Chapter 6

Discussion

6.1 Method

This chapter discusses the methodology and the possible implications of this thesis.

6.1.1 Replicability, Reliability and Validity

The simulation software used in this thesis is not publicly available, however, the detail in which it is described should make the method highly replicable. Note that the observed ON/OFF periods will not be exactly the same for everyone, but they should at least show a similar distribution to the one described in this thesis. WinAppDriver version 1.1.1 is used in this thesis, but since the development of the simulation software, at least one new version has come out. Windows 10 Pro is the OS used to run the simulations. The realism of network traffic can not be measured by variance and self-similarity alone, these are however, the properties examined in this thesis. If one would try to generate network traffic that is realistic in every aspect, one would have to include other factors.

6.1.2 System Requirements

While the focus of this thesis was not on building massive networks of simulated clients, some might be interested in the hardware requirements of the system demonstrated in this thesis. The short answer is that it depends on what kind of network that is to be simulated. Taking the network shown in Figure 3.10 as an example. The Windows machines where given 2 GB of memory each, and the router and NAT 128 MB each, and they where all able to run simultaneously on a Intel i7-7500U processor. An additional 1 GB was allocated to the GNS3 VM in which all VMs where contained. The requirements of WinAppDriver and the simulation software was insignificant enough to be neglected in this context. The requirements of the applications that the simulated user interacts with should, however, be taken into account when designing a simulated network. The network connection of the machine running the GNS3 must be good enough to not limit the amount of ingoing and outgoing traffic of the simulated network. This is of course dependent on the types of simulated scenarios run on the VMs.

6.1.3 Privacy

One of the advantages of synthetically generating realistic network traffic is that one does not have to worry about obfuscating or removing sensitive data, such as passwords. One does, however, need to make sure that this information does not end up in the .pcap-files to begin with. All personal information in the packets send to/from the simulated machines will end up in the generated .pcap-files. One should therefore be careful when simulating scenarios that includes signing into online services.

6.2 The Work in a Wider Context

In this thesis the hope is to make synthetically generated realistic network traffic more accessible to the industry. Widely available data sets of this kind might make a difference in the quality of networking products, the training of network security specialists and to a greater understanding of the Internet.

Chapter 7

Conclusion

WinAppDriver can successfully be used to simulate user activity on a Windows machine. A aggregation of multiple such simulated users in GNS3, with statistically correct ON/OFF periods can be used to synthetically generate network traffic, that is realistic in terms of variability and self-similarity.

7.1 Future Work

This thesis only presents a method for how to synthetically generate network traffic that is statistically similar to that of some observed data. To really make the generated traffic truly realistic though, it is important to make the scenarios as realistic as possible. This was not taken into consideration in this thesis, but studying and modeling real user interaction could be a valuable addition to the work. Another interesting addition to this thesis could be to research how difficult it would be for a human or machine to differentiate between real and generated traffic. Would it be easier with a large enough data set and so on. One could also explore if longer and more simulations would, in fact, lead to a larger Hurst exponent or not.

Bibliography

[1] Walter Willinger, Will E Leland, Murad S Taqqu, and Daniel V Wilson. "On the self-similar nature of Ethernet traffic (extended version)". In: *IEEE/ACM Transactions on Networking* 2.1 (1994), pp. 1–15.

[2] José Luis García-Dorado, Alessandro Finamore, Marco Mellia, Michela Meo, and Maurizio Munafo. "Characterization of isp traffic: Trends, user habits, and access technology impact". In: *IEEE Transactions on Network and Service Management* 9.2 (2012), pp. 142–155.

[3] Subhabrata Sen and Jia Wang. "Analyzing peer-to-peer traffic across large networks". In: *Proceedings of the ACM SIGCOMM Workshop on Internet Measurment*. 2002, pp. 137–150.

[4] Ramón Cáceres, Peter B Danzig, Sugih Jamin, and Danny Mitzel. "Characteristics of wide-area TCP/IP conversations". In: *Proceedings of ACM SIGCOMM*. 1991, pp. 101–112.

[5] Vern Paxson. "Empirically derived analytic models of wide-area TCP connections". In: *IEEE/ACM transactions on Networking* 2.4 (1994), pp. 316–336.

[6] John D Day and Hubert Zimmermann. "The OSI reference model". In: *Proceedings of the IEEE* 71.12 (1983), pp. 1334–1340.

[7] Yingying Chen, Sourabh Jain, Vijay Kumar Adhikari, Zhi-Li Zhang, and Kuai Xu. "A first look at inter-data center traffic characteristics via yahoo! datasets". In: *Proceedings of IEEE INFOCOM*. 2011, pp. 1620–1628.

[8] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. "Understanding data center traffic characteristics". In: *Proceedings of ACM workshop on Research on enterprise networking*. 2009, pp. 65–72.

[9] Theophilus Benson, Aditya Akella, and David A Maltz. "Network traffic characteristics of data centers in the wild". In: *Proceedings of ACM SIGCOMM conference on Internet measurement*. 2010, pp. 267–280.

[10] Kuai Xu, Feng Wang, Lin Gu, Jianhua Gao, and Yaohui Jin. "Characterizing home network traffic: an inside view". In: *Personal and ubiquitous computing* 18.4 (2014), pp. 967–975.

[11] Christian Rossow, Christian J Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten Van Steen, Felix C Freiling, and Norbert Pohlmann. "Sandnet: Network traffic analysis of malicious software". In: *Proceedings of the ACM Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. 2011, pp. 78–88.

[12] Marios Iliofotou, Hyun-chul Kim, Michalis Faloutsos, Michael Mitzenmacher, Prashanth Pappu, and George Varghese. "Graption: A graph-based P2P traffic classification framework for the internet backbone". In: *Computer Networks* 55.8 (2011), pp. 1909–1920.

[13] Balakrishnan Chandrasekaran. "Survey of network traffic models". In: *IEEE Commun. Mag.* (Apr. 1994).

[14] Raoufeh Hashemian, Diwakar Krishnamurthy, Niklas Carlsson, and Martin Arlitt. "A Contention Aware Web of Things (WoT) Benchmarking Testbed". In: *ACM/SPEC International Conference on Performance Engineering*. Edmonton, Canada, 2020.

[15] Kashi Venkatesh Vishwanath and Amin Vahdat. "Swing: Realistic and responsive network traffic generation". In: *IEEE/ACM Transactions on Networking (TON)* 17.3 (2009), pp. 712–725.

[16] Antoine Varet and Nicolas Larrieu. "How to generate realistic network traffic?" In: *Proceedings of the IEEE Annual Computer Software and Applications Conference*. 2014, pp. 299–304.

[17] Niels Provos et al. "A Virtual Honeypot Framework." In: *Proceedings of USENIX Security Symposium*. 2004, pp. 1–14.

[18] Teodor Sommestad. "Experimentation on operational cyber security in CRATE". In: *NATO STO-MP-IST-133 Specialist Meeting*. 2015, pp. 7–1.

[19] Jason Alexander, Andy Cockburn, and Richard Lobb. "AppMonitor: A tool for recording user actions in unmodified Windows applications". In: *Behavior Research Methods* 40.2 (2008), pp. 413–421.

[20] Lee M Rossey, Robert K Cunningham, David J Fried, Jesse C Rabek, Richard P Lippmann, Joshua W Haines, and Marc A Zissman. "Lariat: Lincoln adaptable real-time information assurance testbed". In: *Proceedings of IEEE Aerospace Conference*. 2002.

[21] Charles V Rossey Wright, Christopher Connelly, Timothy Braje, Lee M Rabek Jesse C and, and Robert K Cunningham. "Generating client workloads and high-fidelity network traffic for controllable, repeatable experiments in computer security". In: *Proceedings of International Workshop on Recent Advances in Intrusion Detection*. Springer. 2010, pp. 218–237.

[22] Mark E Crovella and Azer Bestavros. *Explaining world wide web traffic self-similarity*. Tech. rep. Boston University, Computer Science Department, 1995.

[23] Walter Willinger, Vern Paxson, Rolf H Riedi, and Murad S Taqqu. "Long-range dependence and data network traffic". In: *Theory and applications of long-range dependence* (2002).

[24] Jan Beran. "Statistical methods for data with long-range dependence". In: *Statistical science* (1992), pp. 404–416.

[25] JC Ramirez Pacheco. "Behavior of R/S statistic implementations under time-domain operations". In: *Proceedings of IEEE*. 2006, pp. 1–4.

[26] James Clark, Steve DeRose, et al. *XML path language (XPath) version 1.0*. 1999.

[27] Jason Brand and Jeff Balvanz. "Automation is a breeze with autoit". In: *Proceedings of the ACM SIGUCCS conference on User services*. 2005, pp. 12–15.

[28] Dexter C Kozen. *Automata and computability*. Springer Science & Business Media, 2012.

[29] Noam Chomsky. "On certain formal properties of grammars". In: *Information and control* 2.2 (1959), pp. 137–167.

[30] In Jae Myung. "Tutorial on maximum likelihood estimation". In: *Journal of mathematical Psychology* 47.1 (2003), pp. 90–100.

[31] Anthony G Barnston. "Correspondence among the correlation, RMSE, and Heidke forecast verification measures; refinement of the Heidke score". In: *Weather and Forecasting* 7.4 (1992), pp. 699–709.