Diploma Thesis

Analysis and interpretation of emulated data traffic in Android platform

under the supervision of

Dipl.-Ing. Markus Laner,

Dipl.-Ing.Dr. Philipp Svoboda

and

Univ.Prof.Dr. Markus Rupp

Institute of Telecommunications, E389

Department of Electrical Engineering

Vienna University of Technology

by

Alejandro Camacho Arias

1029244

Gran Via de les Corts Catalanes 303, 08014 Barcelona

Wien, October 2011

Abstract

Nowadays, the operating system for mobile devices Android is taking a leading position within the world of smartphones and tablets. The growth that it is experiencing is very significant and everyday more than 550,000 new mobile devices governed by Android are sold worldwide. For this reason it is very interesting to research on progresses in this field.

One of the most used features in this sense is the ability to transmit data between two devices through an internet connection. That is why in this thesis it has been implemented a traffic generator (as an Android application) that allows sending and receiving data simultaneously, being able to choose the type of data to be sent. Thus, four emulated types of data has been transmitted (videos, websites, online games and Skype) in three different environments (in the Android Device Emulator, via WiFi, and connected to the 3G network).

The final goal is to find out how Android manages sending and receiving packets. To do this, after performing all the necessary tests, the results are deeply analyzed to empirically conclude which is the most appropriate connection type for the transmission of different traffic types, being WiFi the best choice to transmit them. In the near future, applications will be able to optimize its performance by taking advantage of this knowledge.

Contents

Chapter 1

Introduction and motivation

1.1 Introduction

Since the beginning of the Second World War, mankind has been improving on how to remotely communicate with their peers. The initial objective at that time was to take advantage over the enemy during the war, while today practically anyone in the world can have access to a mobile phone. Over the years, mobile communications have been one of the technologies that have experienced more advances, and still continue in search of progress and improvements.

The first call from a mobile device was made on April 3th, 1973, when Martin Cooper, then director of R&D in Motorola, called his rival Joel Engel, head of development at Bell Laboratories, from his Dyna-Tac, in Manhattan. He was probably unaware that he had just started a global revolution in the world of mobile communications [1].

Figure 1: Motorola Dyna-Tac.
Source: motorola.index.es

Since that time, there have been uncountable advances in this field. Year after year, the mobile devices were evolving, reducing its size while improving their technical features. One of the greatest evolutions was the emergence of the smartphones, mobile devices with an internal operating system (OS) which allows the installation of customizable programs as well as offering a high capacity of connectivity [2].

On November 5th, 2007, the announcement of the appearance of the new operating system for mobile phones led by Google [3] (Android) was performed. This operating system allows users to develop their own applications.

Figure 2: Andy, the Android logo.
Source: www.android.com

In this thesis it has been developed a traffic generator (implemented as an Android application) that allows for sending and receiving data between two devices equipped with Android OS. The aim is to find out, investigating through empirical measures, how the sending and receiving of data is performed in Android and draw conclusions about its performance.

In Chapter 2 of this thesis, all the characteristics of the traffic generator implemented are shown, as well as the instructions about how to use the application.

In Chapter 3 it is explained how the traffic types have been emulated and which are the necessary preparations to get ready to run the real tests correctly.

Chapter 4 shows the transmission results and gives interpretations of them, to later extract conclusions in Chapter 5. Finally, it is deduced which connection type is the most suitable to transmit each type of traffic, and possible future tasks are proposed for the reuse of the application created, either in this or other researching areas.

1.2 What is Android?

Android is Google's operating system for mobile devices. In July 2005 Google acquired the company Android Inc., which developed software for mobile

phones, and later launched its own operating system on November 5th, 2007, along with the creation of the Open Handset Alliance [4], a consortium of 84 hardware, software and telecommunications companies dedicated to develop open standards for mobile devices. In this way, Google made agreements with various companies to incorporate Android on their devices.

The main features of the Android operating system are:

- Most of the Android code is released under the Apache License, a **free software** and **open source license**.
- Any user can program custom **applications**.
- The applications are written in the object-oriented programming language **Java** on a Dalvik virtual machine (a specialized virtual machine specifically designed for Android and optimized for battery powered mobile devices, which have limited memory and processor) with **runtime compilation**.
- It is based on the Linux kernel.
- Android includes a graphical user interface manager (surface manager), a framework OpenCore, a SQLite relational database, a graphical API OpenGL ES 2.0 3D, a rendering engine WebKit, a graphics engine SGL, and a standard Bionic C library.
- Android allows **multiple functions** such as storing data, play multimedia files, make video-calls, surf the internet or data transferring among others.
- **Multitasking** of applications is available.
- The Android operating system is composed of **12 million** lines of code.

The current Android's market share among smartphones is 32.9% and its progression over the years is shown in Fig 3., activating more than 550,000 new devices incorporating Android each day worldwide.

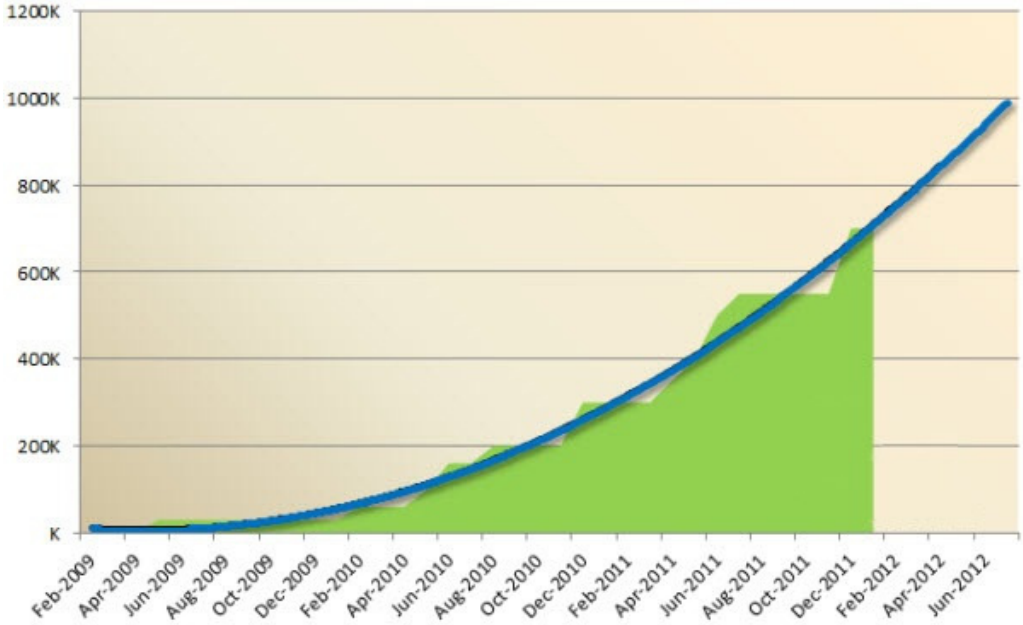

Figure 3: Evolution in sales of mobile devices with Android operating system incorporated.
Source: www.unwiredview.com

As a curiosity, say that the various versions of Android are named as dessert plates in English, and each version begins with a different letter in alphabetical order.

1.3 Motivation

This thesis is surrounded of a context in constant evolution and with a big future. Learning about these technologies is to bet on a sector that is constantly innovating and creating very interesting and useful utilities, which is very motivating and stimulating.

It is very interesting to emulate **real data traffic types** which are commonly used in data transmission. Furthermore, the tests carried out are preformed in **real Android devices**, thereby obtaining **empirical results** that reflect the real behavior and which are the practical result of transmissions that are frequently used. Thus, we can take advantage by knowing Android technology from the inside by testing how Android treats transmissions between terminals.

Moreover, in views to the future, applications programmed from now on will be able to base on the results of this thesis to develop more efficient applications and make better use of Android's features, knowing in advance how it behaves when transmitting data.

1.4 Steps to be followed

These are the steps that have to be followed to achieve the final goal:

1) Implement a **traffic generator** as an **Android application**. This application has to allow the communication between two devices, emulating the desired traffic to be tested.
2) Find reliable **statistical models** to emulate faithfully the desired data **traffic types** to test.
3) Run **real transmissions with emulated data traffic**, one for each traffic type tested in each of the three environments.
4) Extract the resulting **text files** from the application.
5) **Analyze** these text files through Matlab [5].
6) Draw **conclusions** from the results obtained.

Chapter 2

The traffic generator

2.1 Introduction to Markov chains

To emulate the various types of traffic desired Markov chains will be used. A Markov chain is a set of **states** in which each of them plays a specific role. Furthermore, the state can be **sequentially changed** according to a certain probability.

This will allow moving between various scenarios in real time, controlling these jumps according to previously established jump probabilities. In this way, the user of the application will be able to set up a group of states to emulate efficiently the desired traffic.

A more extensive explanation on Markov chains will be shown in section 2.4.

2.2 Desired characteristics

The traffic generator implemented in this thesis will be an Android application which will allow for communication between two Android devices, programmed using Eclipse [6]. During the data exchange, both devices will be able to both send and receive information. As Android applications are developed in Java programming language [7], it is possible to create various **threads**. A thread is a part of code that can run in parallel to other threads, sharing resources as memory, objects and variables defined there. The alternation between threads is managed by the Dalvik Virtual Machine (DVM), the Java Virtual Machine (JVM) optimized for mobile devices, which is responsible for assigning the execution time to each thread, based on priorities. The impression for the final user is like being sending and receiving data simultaneously.

An important characteristic of this traffic generator is flexibility and reusability. There are eight characteristics to set up before the transmission starts, so that the communication is strongly customizable.

Firstly, each of the devices has to define its role, either as sender or receiver.

- The role assumed: either sender or receiver.

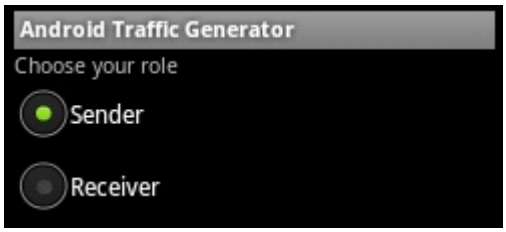

Figure 4: Appearance of the choice of role in Android Traffic Generator

In a communication between two devices, one of them will act as sender, and the other one as receiver. In the sender side, the statistical properties of both are established, so that the receiver will reply with the traffic characteristics defined by the sender.

Then, the sender can establish the following features:

- The Internet Protocol (IP) address and the port to which data is going to be sent, and the port where incoming data is going to be received.

Figure 5: Destination port and IP address, and local sender port through which receive incoming traffic.

- The protocol used: either UDP (User Datagram Protocol) or TCP (Transmission Control Protocol).

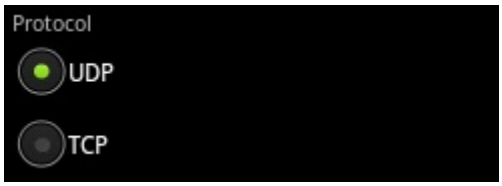

Figure 6: Protocol to be used by the sender device.

In UDP protocol, **datagrams** are transmitted, which may arrive out of order, appear duplicated in the receiver, or be missed without notice [8]. On the other hand, TCP performs a previous connection establishment before transmitting

data, also called **handshaking**; Then, communication endpoints transfer data and when finished, another handshaking (this time for terminate the connection) is done. Through this handshaking mechanism, the transmission becomes reliable, with an ordered delivery and providing congestion control [9].

- The number of packets to be sent before the communication ends.

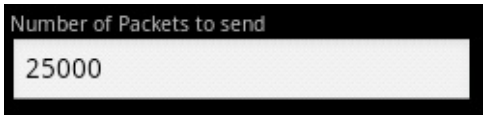

Figure 7: Number of packets to be sent.

This value indicates how many packets are going to be sent from the sender to the receiver. When all the packets have been sent, both the sender and the receiver stop sending packets.

And the receiver can determine these two features:

- The port where get data from.

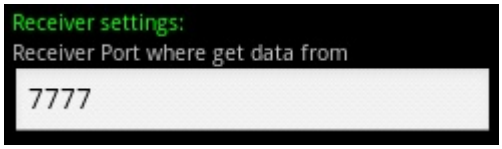

Figure 8: Receiver port through which incoming data is going to be received.

- The protocol used by the receiver, which **must coincide** with the protocol used by the sender.

Figure 9: Protocol to be used by the receiver.

Apart from these features related to the establishment of the communication, the data traffic types to be sent **for both devices** are determined **in the sender side**. These are the adjustable parameters:

- The size (in bytes) of each packet, and the time between two consecutives packets (in milliseconds). These two values may follow a statistical distribution, chosen by the user. This aspect will be discussed in further detail in section 2.2.
- Markov states, which will be determined by the Packet Size (PS) and the Time Between Packets (TBP) statistic chosen, and the transition probabilities between the Markov states. This aspect will be discussed in further detail in section 2.3.

Once all the desired parameters for the transmissions have been selected, the receiver device must press the *GET DATA* button, and subsequently, the sender device must press the *SEND* button so that the bidirectional data transfer starts.

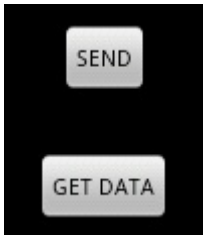

Figure 10: Buttons SEND and GET DATA for the Sender and the Receiver devices, respectively.

While the transmissions are running, the application generates two text files on each device involved in the communication, containing:

- The size of each packet sent.
- The time between two consecutive packets sent.
- The size of each packet received.
- The time between two consecutive packets received.

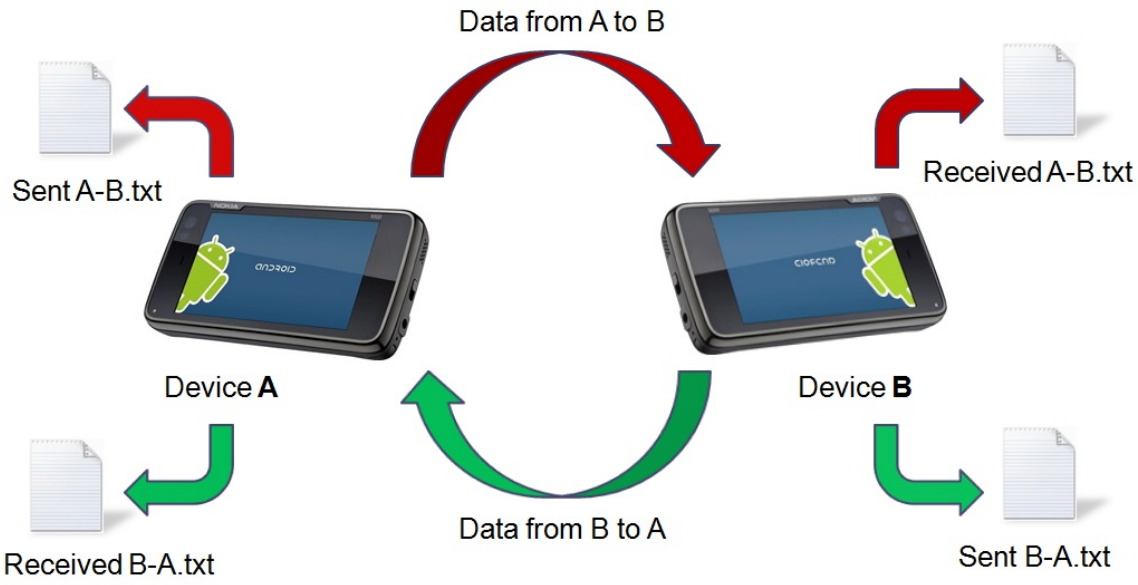

Figure 11: Text files generated during transmissions.

This four text files are really important, because they contain all the necessary information to be analyzed after the transmissions, and to extract conclusions like packet loss, delay, etc.

2.3 Packet size and time between packets

In some kinds of communication, the transmission of information is made through the sending of packets. The whole message to be sent may be fragmented in several little pieces of information, forming packets. A packet consists of two different parts:

- **Payload** bytes: those bytes that transport the useful information for the user.
- **Header** bytes: some added bytes which are used to codify the information, protect it and let the whole packet reach the final destination with the minimum number of errors.

If the communication finishes successfully, the final message should be the union of the payload bytes of all the packets received.

For simulating real traffic, a successive sending of packets has to be emulated. Depending on the traffic type emulated, these packets will have two main statistical characteristics: the **size** of the packets (in bytes), and the **time between two consecutive** packets (in milliseconds).

2.3.1 Packet size (PS)

Not necessarily all the packets in a communication are of the same length. Depending on aspects like the kind of information to be transmitted, or the protocol or the kind of network used, the length of each packet may be different from other packets in the same transmission.

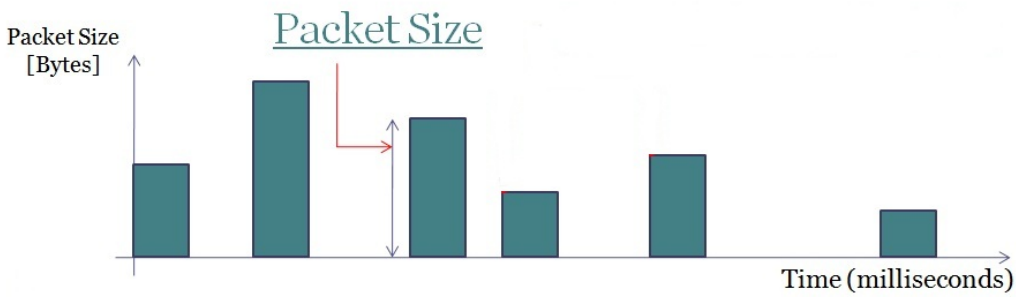

Figure 12: Representative diagram of the packets' sizes generated during a data transmission.

This characteristic (the size of each packet in bytes) usually follows a statistical distribution, that is, a **probability mass function** (PMF) because the possible values of the samples are always **integers** (in this case, bytes). In the traffic generator implemented, there is the option to choose between five different statistics for the packet size, basing on the most suitable statistics for the traffic emulated:

1) Uniform (a, b)

These are the formula and the shape of the probability mass function of a discrete uniform distribution between *a* and *b*.

$$fx(X) = \Pr(X = x) = \begin{cases} \dfrac{1}{b-a+1}, & x \in [a,b] \\ 0, & otherwise \end{cases}$$

Equation 1: Probability Mass Function of a discrete uniform distribution.

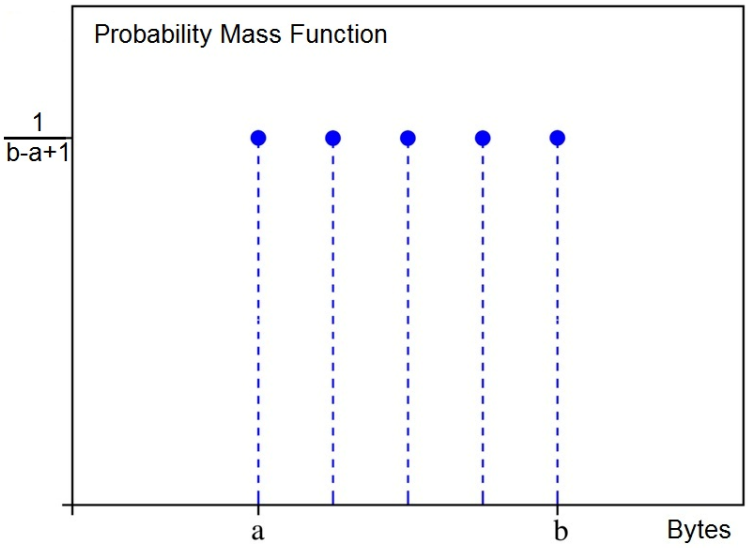

Figure 13: Shape of the PMF of a discrete uniform distribution.
Source: en.wikipedia.org /wiki/Uniform_distribution_(discrete)

Where the horizontal axis indicates the possible integer values (bytes, in steps of 1), and the vertical axis indicates the probability of each one to happen.

In a uniform distribution, all the values between *a* and *b* are equiprobable, with probability $\frac{1}{b-a+1}$, while all the other values will never occur.

The mean of a discrete uniform distribution is $\frac{a+b}{2}$, and its variance is $\frac{(b-a+1)^2-1}{12}$.

The uniform distribution is good to emulate, for example, the size of online games packets in the uplink side [10].

2) Constant (ct)

In a constant distribution, the value *ct* has probability 1 to appear, that is, in the packet size case, all the packets generated will have length *ct* bytes.

$$fx(X) = \Pr(X = x) = \begin{cases} 1, & x = ct \\ 0, & otherwise \end{cases}$$

Equation 2: Probability Mass Function of a constant distribution.

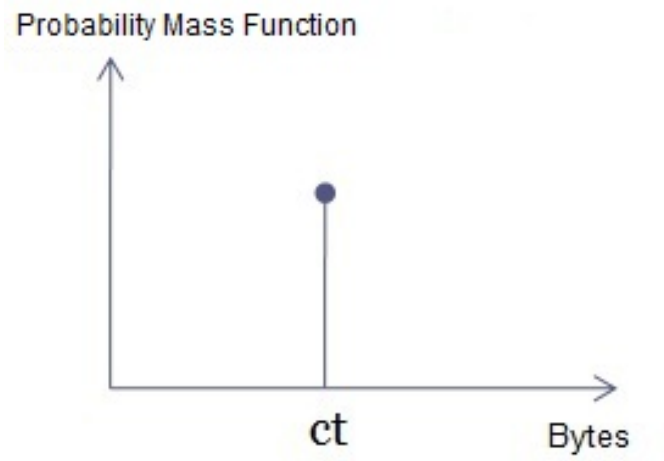

Figure 14: Shape of the PMF of a constant distribution.

Its mean is *ct* and the variance is always 0.

This distribution is appropriate to generate constant times between packets like happens in online games traffic [10].

3) Lognormal (μ, σ)

The lognormal distribution (with parameters *μ* and *σ*) is a probability distribution that comes from a Gaussian distribution:

The Gaussian distribution (also named normal distribution) is a continuous probability distribution with a shape similar to a bell. These are its formula and the appearance of its Probability Density Function (PDF):

$$fx(X) = \Pr(X = x) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Equation 3: Probability Density Function of a Gaussian distribution.

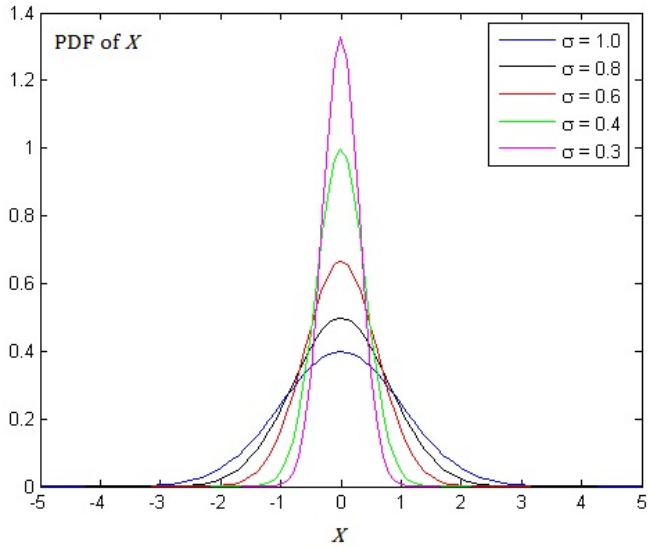

Figure 15: Shape of the PDF of Gaussian distributions with mean 0 and various values of the square root of the variance.
Source: http://www.experiment-resources.com/normal-probability-distribution.html

The parameter μ of the Gaussian distribution coincides with its mean and is where the peak is centered, while the parameter σ is the square root of its variance.

If X is a random variable which follows a Gaussian distribution, then $Y = e^X$ has a **lognormal** distribution; likewise, if Y is lognormal distributed, then $X = \log(Y)$ is normal distributed. It is important to remark that μ and σ (the parameters of the lognormal distribution) are **not** the mean and variance of the lognormal distribution ($f_Y(Y)$), but are the mean and the variance of the normal distribution ($f_X(X)$) from which it comes. So μ and σ are the *mu* and *sigma* parameters of the lognormal distribution, but do not match its mean and variance.

In Fig. 16 an example can be seen: a Gaussian random variable (X) with mean 5.5 and variance 0.33 is generated. Then, each of its samples are transformed by performing the operation $Y = e^X$. Then, all the samples of the new random variable Y are following a lognormal distribution with parameters $\mu = 5.5$ and $\sigma = 0.33$:

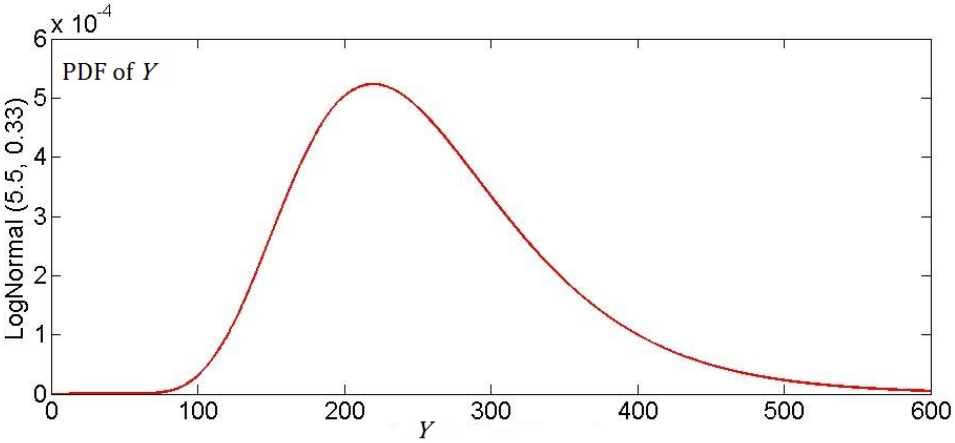

Figure 16: Shape of the PDF of a Lognormal distribution with μ = 5.5 and σ = 0.33.

In Eq. 4 it is shown the formula of the PDF of a lognormal distribution.

$$fx(x) = \Pr(X = x) = \begin{cases} \dfrac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Equation 4: Probability Density Function of a lognormal distribution.

The mean and variance of the resulting lognormal distribution can be calculated as shown:

Mean = $e^{\mu+\sigma^2/2}$ = 258.38 bytes

Variance = $(e^{\sigma^2} - 1)e^{2\mu+\sigma^2}$ = 7681.11 bytes

The lognormal distribution is suitable for emulating video packet sizes [11].

4) Exponential (λ)

The exponential distribution (depending on the value of its parameter *λ*) is a family of continuous probability distributions (always decreasing).

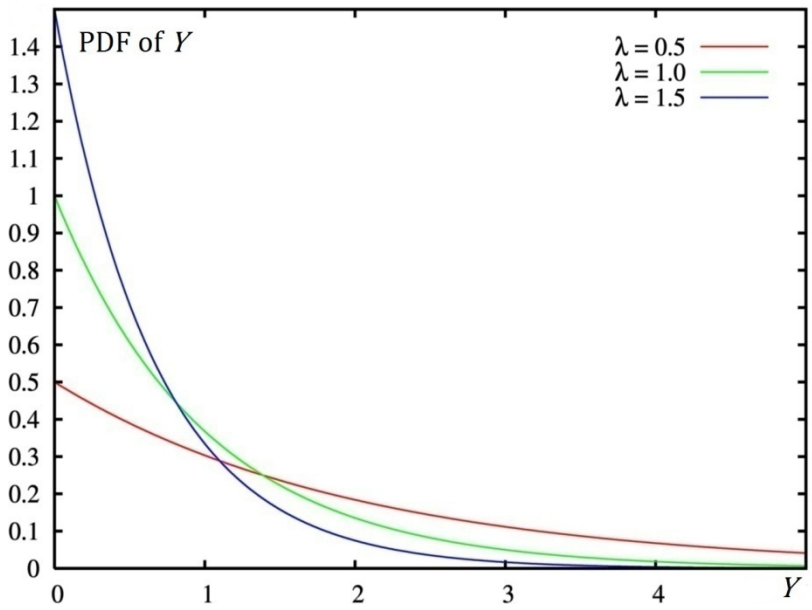

Figure 17: Shape of the PDF of an exponential distribution, for various values of λ.
Source: http://math.info/Probability/Exponential_Distribution/

Its shape depends on the value of λ, more flat when λ is small and steeper when λ increases.

In Eq. 5 it is shown the formula of the PDF of an Exponential distribution.

$$fx(X) = \Pr(X = x) = \begin{cases} \lambda e^{-\lambda x}, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Equation 5: Probability Density Function of an exponential distribution.

The exponential distribution is used to describe time between events in a Poisson process, that is, a process where its events occur independently at a given constant average rate.

The mean coincides with λ^{-1}, and its variance with λ^{-2}.

This distribution is useful when emulating the traffic of a webpage in terms of time between packets [12].

5) Geometric (p)

The geometric distribution is the discrete analogue case of the exponential distribution.

It represents the probability that the first occurrence of success require *n* number of independent trials, each with success probability *p*. If the probability of success on each trial is p, then the probability that the *n*th trial is the first success coincides with its PMF:

$$P(X = x) = (1 - p)^{n-1}p$$

Equation 6: Probability Mass Function of a geometric distribution.

With *n*=1, 2, 3… Its parameter *p* must be 0<*p* ≤1. In Fig. 18, two geometric distributions (with *p*=0.1 and *p*=0.3) are shown:

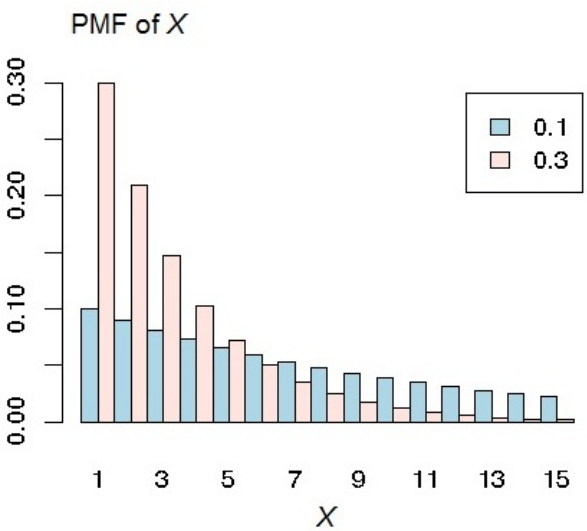

Figure 18: Shape of the PMF of two geometric distributions.
Source: http://raven.iab.alaska.edu/~ntakebay/teaching/programming/probability/node8.html

If the floor (or greatest integer function) is applied to an exponential distribution, it turns out to be a geometric distribution.

The mean of a geometric distribution is 1/p, and its variance $(1-p)/p^2$.

All these statistics can be chosen by the user in the Graphical User Interface (GUI), using a drop down menu as shown in Fig. 19.

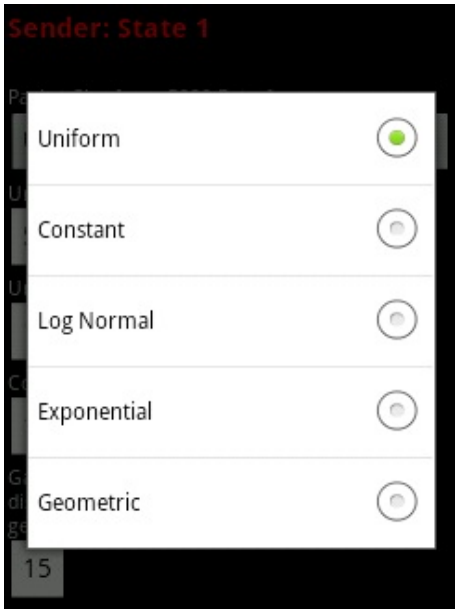

Figure 19: Eligible statistics for packet size and time between packets in the GUI.

The parameters for each statistic can be also set to your choice as shown in Fig. 20.

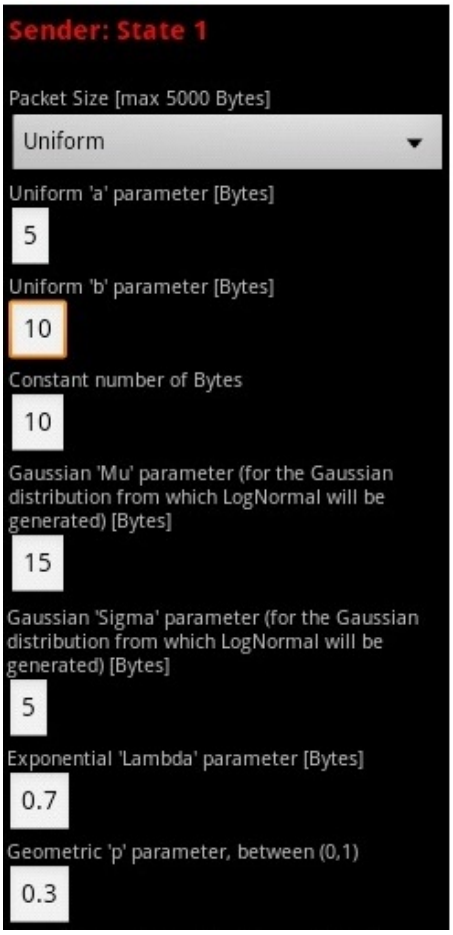

Figure 20: Eligible parameters for each statistic in the GUI.

2.3.2 Time between packets (TBP)

Between any two consecutive packets sent in a communication, there is a temporal space. Depending on the characteristics of the emulated traffic, this time between two packets may follow a pattern (for example, be short and constant in the case of online games).

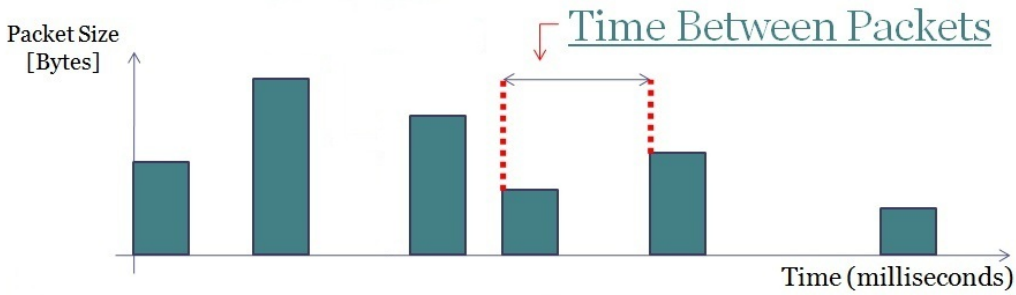

Figure 21: Representative diagram of the times between packets generated during a data transmission.

It is important the time between two packets to be calculated from the same point of two consecutive packets, that is, between the **beginning** of two consecutive packets (like in the figure) or between the **end** of both, and do the same with all the remaining packets.

This aspect (TBP) can also be determined in the traffic generator designed. As well as in the case of packet size, the same five statistical distributions can be chosen (uniform, constant, lognormal, exponential, and geometric) because the possible values of TBP are also integers.

It is important to mention that implementing the time between packets in Java means demanding to the thread that has to be "sleeping" a certain number of milliseconds. To do this, the method *sleep* must be invoked, and for the own nature of threads (and as it is explained in Java's class Thread), *"the precision is not guaranteed when using the method sleep".* This means that the time demanded to the thread to be sleeping will not necessarily be strictly accomplished.

2.4 Markov chain

The traffic generator also uses Markov chains to switch between different statistics at real time. A Markov chain is defined as the probabilistic transition between a countable number of states (S_1, S_2, S_3, …, S_n). The process starts in one of these states and moves successively from one state to another. If the chain is currently in state S_i, there is a certain probability (initially determined) to jump to state S_j. This probability is called the *transition probability*, and is denoted as p_{ij}. There is also the possibility of resting in the same state S_i and do not jump to any other. This happens with probability p_{ii}.

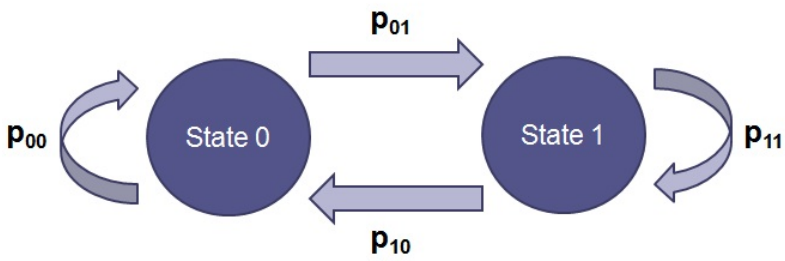

Figure 22: Markov chain with two states and their transition probabilities.

In Fig. 22, p_{01} and p_{10} are the transition probabilities, while p_{00} and p_{11} are the probabilities of resting in the same state.

These probabilities can be placed in a matrix, forming the *matrix of transition probabilities*, or *transition matrix* ($\underline{T}$). In this matrix, p_{ij} is placed in the row *i* and in the column *j*, indicating that the transition is from the state S_i towards the state S_j [13].

$$\begin{bmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{bmatrix}$$

Figure 23: Transition matrix of a Markov chain with two states.

In the next transition, the following state reached will only depend on the current state, so if S_i is the current state, the next one will only depend on the transition probabilities in the row *i* of the transition matrix.

In the traffic generator, a state is defined as the statistic chosen for packet size, and the statistic chosen for time between packets.

Figure 24: Consisting statistics of one single Markov State.

For example, the state 1 can be defined as:

- PS: Lognormal (3.74, 1.82) [bytes]
- TBP: Uniform (100, 150) [milliseconds]

In the traffic generator there can be up to 5 different states, and the default initial state is the state 1.

Figure 25: An example of five-states Markov chain created in the traffic generator.

2.5 Multiple jumps

Additionally to the Markov chain with 5 states, the traffic generator allows the user to choose to which state it is possible to jump, and with which probability to do anyone of these jumps. For instance, if the sequence is in the state S_1:

- It is able to jump to anyone of the other states (S_2, S_3, S_4, S_5) or rest in the same state (S_1).
- The transition probabilities to other states (p_{12}, p_{13}, p_{14}, p_{15}) can be chosen.
- Automatically, p_{11} gets determined as $p_{11} = 1-(p_{12}+p_{13}+p_{14}+p_{15})$.
- The following condition must be satisfied:

$$\sum_{i=1}^{5} p_{1i} = 1$$

Equation 7: Condition regarding to the transition probabilities which must be accomplished in a Markov chain.

Thereby, it is possible to jump from state 1 to any other, choosing with which probability do it.

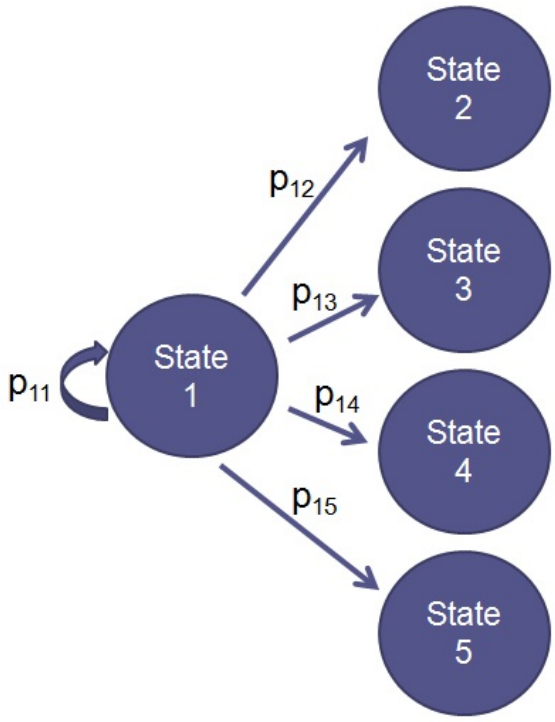

Figure 26: Possible transitions from state 1 in a five-states Markov chain.

Expanding this property to all the states allows the interconnection between all the states, being able to jump from any state to any other.

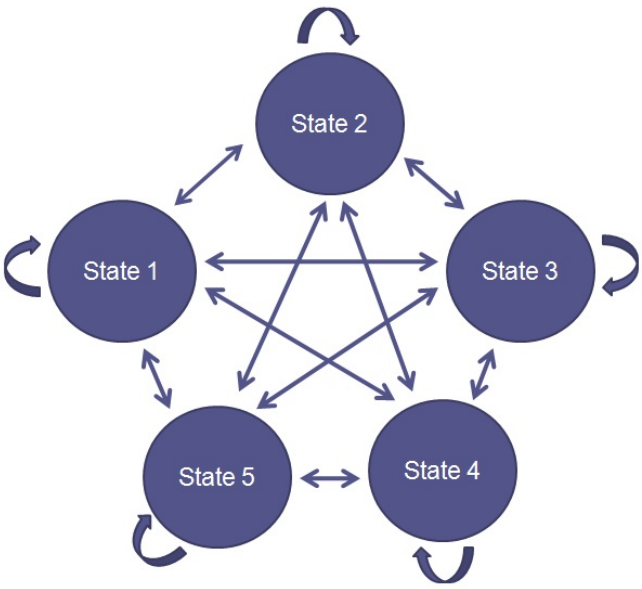

Figure 27: All the possible transitions in a five-states Markov chain.

This characteristic gives much more power to the application and allows the user to perform really precise transmissions. The traffic types emulated in this thesis are usually formed by more than one statistic, so a Markov chain with multiple states was necessary.

In the application it can be defined all the possible jumps performed in the Markov chain, choosing which will be the next state and with how much transition probability reach it. The selection can be made through the options shown in Fig. 28.

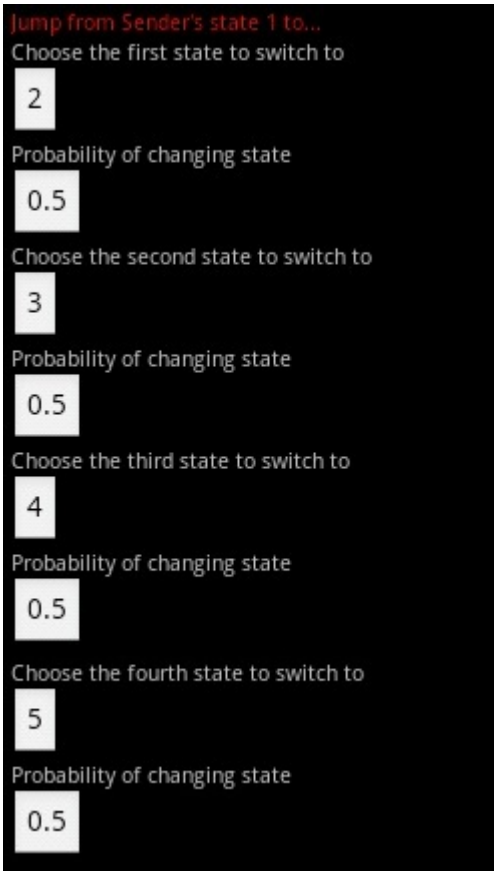

Figure 28: User interface to set the following states in the Markov chain with its corresponding transition probabilities.

2.6 Advantages of this mixed implementation

Usually, traffic generators are based on **one** of these features:

- One fixed statistic for packet size and another also fixed for time between packets.
- A Markov chain, where each state occurs with a certain probability and represents a fragment of the global desired probability mass function.

Putting all the states together, the global shape approaches to the desired PDF.

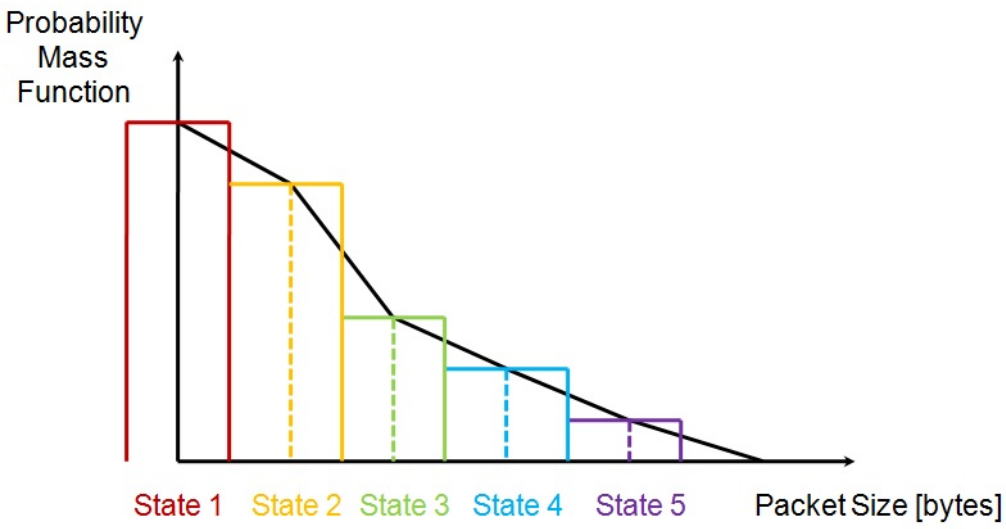

Figure 29: Resulting PMF formed by a five-states Markov chain.

The traffic generator designed during this thesis combines both features, yielding a powerful tool, able to emulate complex traffic types. Without the option of creating Markov chains, the emulator would only be able to generate a finite number of PDFs, without correlation between samples; whereas adding Markov states, the complexity of emulations that can be done increases significantly. The outcome is a Markov-Modulated random process.

This aspect makes the application flexible and completely reusable, because in the future, any user will be able to emulate the type of traffic desired and test how it behaves in Android.

2.7 Generation and transmission of packets

When the buttons *Get Data* (on the receiving device) and *Send* (on the transmitter) are pressed, a **bidirectional** transmission of packets starts. In this section it is going to be briefly explained how this transmission of packets takes place.

To start the transmission of packets, the receiving device must press the *Get Data* button and then the transmitter must press the *Send* button. Then, two

threads are started on each device, one that will be dedicated to receive packets and the other to send them.

Firstly, the transmitter device sends to the receiver **the Markov chain** governing the receiver's behavior referred to sending packets, along with the number of packets to be sent. Once this transmission has been done, the two devices are ready to start sending packets, following each one its Markov chain.

The generation and the transmission of a single packet are done within a loop as follows:

- One packet is generated with its size in concordance with the **packet size statistic** established.
- That packet is sent and the thread is paused a certain time, whose value follows the **time between packets statistic** of the current Markov state.
- After this delay, it is decided **whether to change state** in the Markov chain, depending on transition probabilities.
- Another packet is generated, starting the cycle again **until all the packets are sent**.

The transmission of packets is bidirectional, meaning that both devices send and receive packets at the same time, since one thread receives the incoming packets and the other one sends the outgoing packets to the destination device.

If the transmission fails, a message appears in the screen indicating how many packets have been sent and received successfully so far. In appendix D some problems occurred during the implementation of this application are described and solved.

Chapter 3

Measurement setup

3.1 General settings

To perform measurements, it is necessary to know the statistical characteristics of each of the four types of traffic that are going to be emulated (video, web, online games and Skype), that is, create Markov chains in whose states the **size of the packet**s and the **time between packets** have to be determined. These two parameters are the most relevant ones when generating and analyzing **data traffic**. It is also important determine the transition probabilities of each state for a proper overall functioning of the Markov chain. Moreover, all the traffic types will be transmitted in each of the three environments available (emulator, WiFi [14] and 3G [15]), being able to compare the results between them and see how traffic types empirically behave using each of the connection types.

The packet transmissions will be carried out between two ZTE Blade mobile phones (in WiFi and 3G environments) and between two instances of Android Virtual Device (AVD) on a single computer for the case of the emulator (through the Android Software Development Kit (SDK)). It is important to highlight that the traffic between these devices will be bidirectional and that each device will transfer different traffic: generally the first device will request for a service type, and the other will respond by offering the traffic with statistical properties corresponding to the requested service. It is important to note that the contents of the packets are randomly generated bytes, but they satisfy the required statistics in terms of packet size and time between packets.

Figure 30: One device requests a certain traffic type and the other one responds with the requested traffic.
Source of the device image: http://www.hardreset.eu/zte_blade_hard_reset_soft_reset_en.html

3.2 Traffic setup

In Appendix C it can be found a table with all the statistics for all types of traffic, to get an easier way to consult how they are emulated.

3.2.1 Video

In a video stream, the jitter is the most important feature to have a good quality of signal. Jitter is the fluctuation in the delay of the packets received, that is, the loss of periodicity. It is desired the jitter to be small to have a constant delay (under a certain threshold), otherwise the video playback would not be continuous.

For emulating video traffic, the method followed reproduces how video is encoded/ decoded: the video is partitioned into pieces called *group of pictures* (GoP), always following the same structure repeatedly. Every GoP is made up of three kinds of frames:

- I-frames (intra coded pictures): this is the only kind of frame that does not need another frame to be decoded. It is the reference picture, and most

of the other frames base on its information to be decoded. Every GoP starts with an I-frame.

- P-frames (predictive coded picture): the P-frames bases on the **previous** frame to be decoded. The number of bytes of a P-frame uses to be lesser than in an I-frame.
- B-frames (bidirectionally predictive coded picture): this is the kind of frame with less bytes, because it takes profit on **both the previous and next frames** to get decoded, basing on their differences, so the necessary bytes to do this is lower.

These I, P and B-frames gets combined forming a GoP structure, which will be repeated periodically. There are many options to combine I, P and B frames to form GoP of different lengths, but the choice to be implemented in this thesis has been IBPB, see Fig. 31.

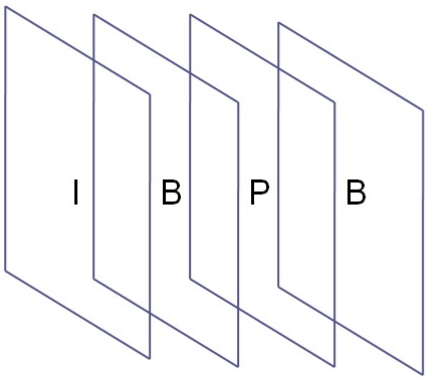

Figure 31: GoP structure implemented for video traffic.

Thereby, the traffic sent will correspond to this GoP repeatedly: IBPB-IBPB-IBPB… This is implemented in the traffic generator creating four Markov states (I-B-P-B) and forcing its sequential execution through the transition probabilities set to 1:

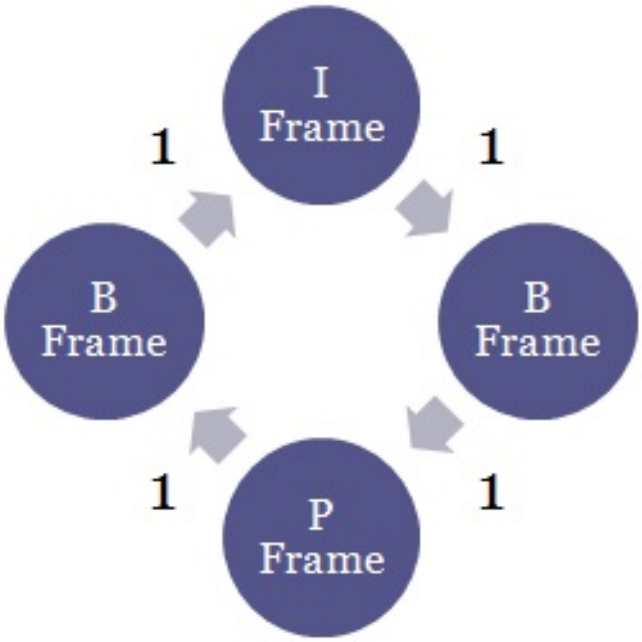

Figure 32: Markov chain to emulate response video traffic.

Each kind of frame (I, P and B) has got its own statistical characteristics (see Tab. 1) [11], and the time between them is such that the final rate is 24 frames/sec, that is, one frame each 42 milliseconds. The kind of protocol used in this transmission is UDP.

	Packet Size [bytes] [11]	Time Between Packets [milliseconds]
I-frames	Lognormal (9.167, 0.2014)	Constant (42)
P-frames	Lognormal (6.839, 0.2674)	Constant (42)
B-frames	Lognormal (7.708, 0.5968)	Constant (42)

Table 1: Statistical characteristics of response video traffic.

On the other hand, the request side of the communication has been emulated as a simple request (350 bytes) [12]. It has been assumed that after each GoP a request is made, that is, six requests per second, also using UDP protocol:

Figure 33: Markov chain to emulate request video traffic.

	Packet Size [bytes] [12]	Time Between Packets [milliseconds]
Request	Constant (350)	Constant (167)

Table 2: Statistical characteristics of request video traffic.

3.2.2 Web

When a web page is requested from the user, the desired response is a quick load of the whole page, or at least, a quick load of the **main** information of the web page. This means that a big block of information must be transmitted quickly to get a short response time. Then, the user takes a while to read and digest the web page, and uses to consult another website, starting again a cycle that uses to be repeated.

The information loaded in a web page can be classified in two categories:

- The **main object**, representing the useful information of the website as one big-sized packet.
- Some **embedded objects**, that is, some other parts of the website that have to be loaded, like advertisements, animations, music…

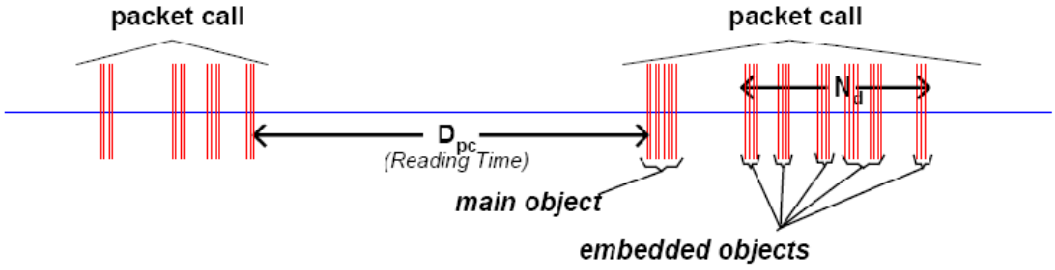

Figure 34: Packet Trace of a Typical Web Browsing Session.
Source: [12]

The number of embedded objects is not fixed, and depends on the website, but the mean is 5.64 embedded objects per web page [12]. The group formed by the main object and all the embedded objects loaded represents the transmission made from the web server to the user.

The last part of the cycle is the **reading time** (that time taken for the user to read and digest the web page), modeled with a Markov state where the packet contains the minimum amount of data allowed to be sent in Android, that is, 1 byte. This permits to identify all the packets of size 1 byte as packets generated by the *Reading* state.

The resulting Markov chain created to emulate the whole cycle of web replies is the following:

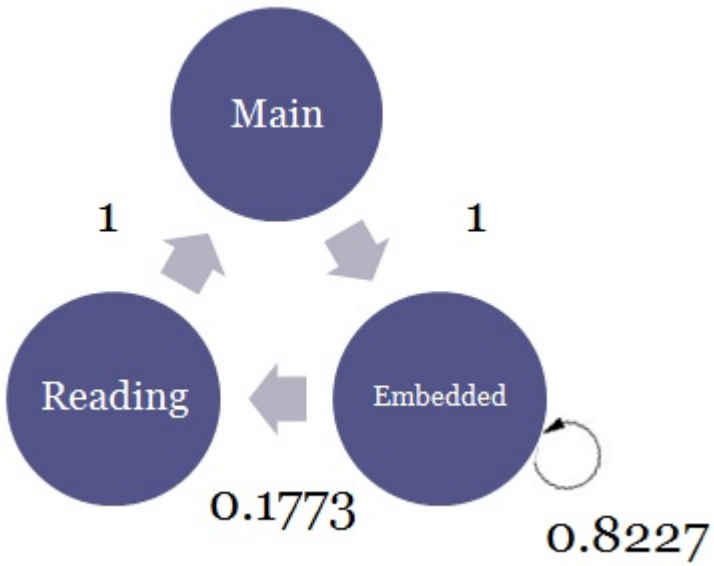

Figure 35: Markov chain to emulate response web traffic.

And the corresponding statistical parameters for each state are:

	Packet Size [bytes] [12]	Time Between Packets [milliseconds] [12]
Main Object	Lognormal (8.37, 1.37)	Exponential (0.0077)
Embedded Objects	Lognormal (6.17, 2.36)	Exponential (0.0077)
Reading Time	Constant (1)	Exponential (0.000033)

Table 3: Statistical characteristics of response web traffic.

The sequence starts in the *Main object* state, sending the Main object of the web page requested. After this, one *Embedded* object is sent, and then, the probability of changing to the *Reading* state is 0.1773, that is, 1/5.64. Consequently, the probability of sending another *Embedded* object is 1-1/5.64, leading to a behavior where, in mean and at long term, 5.64 embedded objects will be sent per cycle. This is accomplished because there will be always one *Embedded* object (the first one) that will be sent **before** checking the transition probability for the first time. Let's clarify this with a simple example:

If we have two Markov states (A and B) where the transition probability from A to B is 1/3:

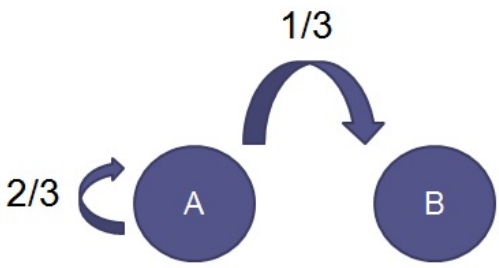

Figure 36: Example of transition probabilities from state A to state B in a two-states Markov chain.

The first thing that A does is to send a packet (like embedded object state does), and then the transition probability either to jump to B or to rest in A is

checked. From now on, in the long term, one of each three times will be a jump from A to B, and two of each three times the sequence will remain in state A, that is, two of each three times another packet will be sent from A. If we add the initial packet to these two sent, we get, in average, 3 packets sent from A. Once shown this behavior with a transition probability of 1/3, the same behavior is expected to happen with a transition probability of 1/5.64.

When the *Reading* state is reached, one byte is sent and the mean time waiting is 30 seconds (approximately what it takes the user to read the page) following an exponential distribution. Then the cycle starts again, simulating another website reply.

All these packets are sent using the TCP protocol, and it is important to highlight that all the times between packets established with TCP protocol have been increased 100 milliseconds; the reason is that packets sent with TCP protocol and with less than 100 milliseconds between them, appear together in the receiver side.

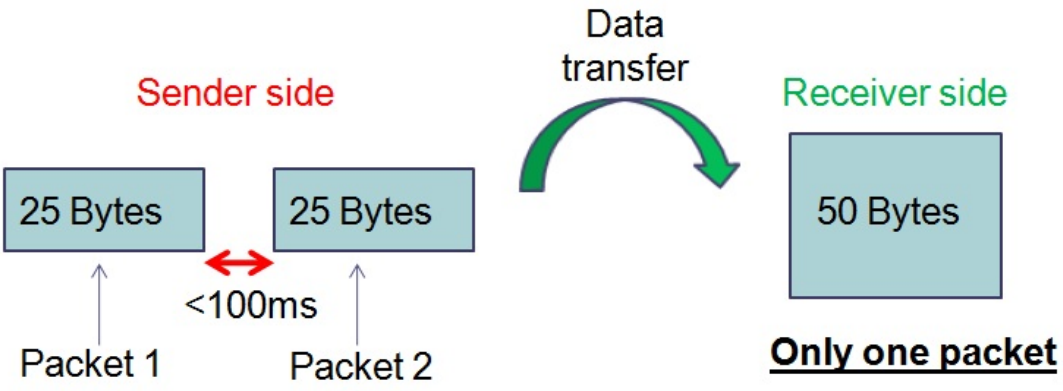

Figure 37: Problem occurred using TCP protocol with times between packets lesser than 100 milliseconds.

This behavior was verified empirically and only happened when using TCP protocol, probably because both packets (temporally separated less than 100 ms) entered in the TCP receiver's buffer before the data were passed to the application. This caused an undesired behavior, because two packets were treated as one, and adding 100 milliseconds of delay between each packet, this effect was considerably reduced.

The request side of the web communication was implemented as a simple request message every 31 seconds, approximately the duration time of one cycle of the above mentioned.

Figure 38: Markov chain to emulate request web traffic.

	Packet Size [bytes] [12]	Time Between Packets [milliseconds]
Request	Constant (350)	Constant (31000)

Table 4: Statistical characteristics of request web traffic.

3.2.3 Online games

When playing an online game, the user desires to feel connected all the time to the game and have a constant and almost imperceptible delay. Thus, during all the game session, the delay between packets will have to be constant and very small. This will lead into an interactive experience where the user will feel as taking part on the game at any moment, having a responsiveness game experience and providing a sense of consistency and robustness.

To get this behavior, the traffic from the game *Team Fortress 2* has been emulated [10] using the UDP protocol. This is the scheme implemented for the traffic in the **uplink** side (the communication from the client to the game server):

Figure 39: Markov chain to emulate uplink online games traffic.

	Packet Size [bytes] [10]	Time Between Packets [milliseconds] [10]
Uplink	Uniform (60, 102)	Constant (34)

Table 5: Statistical characteristics of uplink online games traffic.

The size of the packets is distributed uniformly between 60 and 102 bytes and all of them are temporally separated 34 milliseconds.

On the other side, the **downlink** traffic (those packets from the game server to the user) is emulated as follows:

Figure 40: Markov chain to emulate downlink online games traffic.

	Packet Size [bytes] [10]	Time Between Packets [milliseconds] [10]
Downlink	Lognormal (5.5, 0.33)	Constant (50)

Table 6: Statistical characteristics of downlink online games traffic.

In this case all the packets are equally separated 50 milliseconds and their sizes follow a Lognormal distribution.

3.2.4 Skype

When using Skype, it is desired to have a good video and voice quality (clarity, fidelity and noiselessness) and do not have conversational delays nor echo, so that the conversation can be fluid and clear.

To emulate Skype traffic, it has to be taken into account that video and audio is transmitted at the same time. Furthermore, the audio signal generated by a user changes dynamically in terms of time between speaking and remaining silent. Therefore, it is required a Markov chain where the audio and the video go alternating. This is the implementation proposed using UDP protocol:

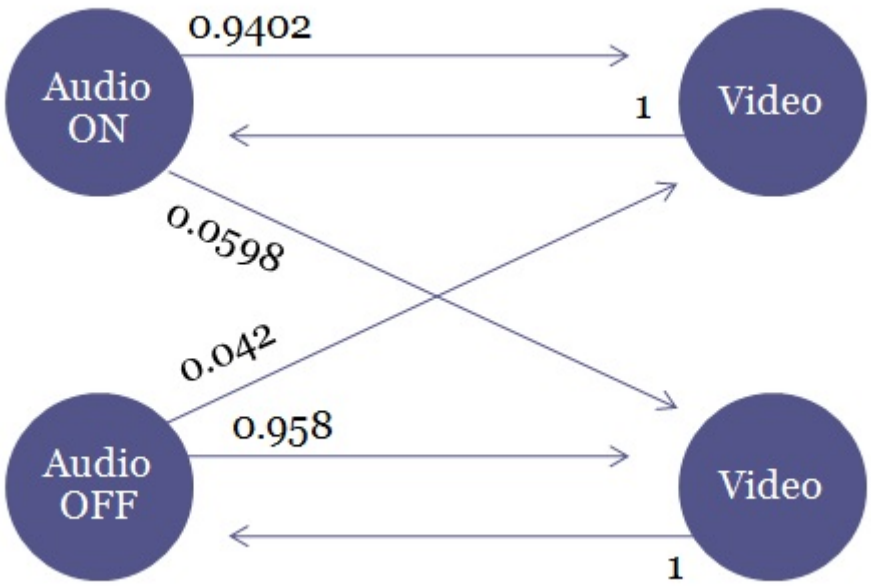

Figure 41: Markov chain to emulate Skype traffic.

Where the statistics for each kind of traffic are the following:

	Packet Size [bytes]	Time Between Packets [milliseconds] [17]
Audio ON	Constant (40) [16]	Constant (30)
Audio OFF	Constant (1)	Constant (30)
Video	Uniform (453, 488) [18]	Constant (30)

Table 7: Statistical characteristics of Skype traffic.

First of all, it is important to note that the two states named *Video* are exactly equal; this is because whether a user is speaking or not, video is transmitted anyway, in parallel to the audio.

Imagine that the sequence starts in the *Audio ON* state: there will be a very high probability of jumping to the upper-right *Video* state. Once there, the sequence will go again to the *Audio ON* state with probability 1. This cycle is very probable to be repeated several times (94.02% of the jumps from *Audio ON* will be in this way) and represents the part of the Skype conversation where the user is **speaking**.

When the jump is made from the *Audio ON* state to the bottom-right *Video* state, another very probable cycle will happen: consecutive jumps between *Audio OFF* and the bottom-right *Video* states, which implements the part of the Skype conversation where the user remains **in silence**.

The transition probabilities to switch between speaking and silence (and vice versa) are 5.98% and 4.2% respectively, allowing each of the cycles to last in time like happens in real Skype conversations. These transition probabilities have been extracted from a paper analyzing VoIP traffic based on human conversation patterns [19].

This scheme lets the alternation of audio and video packets, don’t prioritizing any of them. Furthermore, to test Skype traffic, the same scheme is applied to both devices involved in the communication, so the request traffic scheme is the same than the response traffic.

3.3 Environment setup

3.3.1 Emulator

The first of the three environments where the application is tested is an Android Virtual Device Emulator, see Fig. 42.

Figure 42: Appearance of the Android virtual device.
Source: Android SDK

To carry out the transmissions, two emulators have to be running on a single computer and the traffic generator application must be loaded on both emulators. The two emulator instances are isolated from Internet, and run behind a virtual router, but they can get connected redirecting their ports.

To perform this redirection, the next steps have to be followed:

- Start two Telnet sessions (using a Telnet client, for example PuTTY), connecting to the IP address 127.0.0.1 (localhost, that is, the loopback interface) and the corresponding port for each emulator. In this way, we can give orders to the emulator remotely. The port assigned to the first emulator running is 5554, and for successive emulators this number has to be increased in pairs (5556, 5558 and so on).
- Subsequently, the ports will have to be redirected. To do this, the following command must be introduced in each telnet session started:

  redir add <protocol>:<host_port>:<guest_port>

Thus, we will be redirecting the packets sent to localhost:<host_port> to the corresponding emulator in the port <guest_port>, using the protocol indicated. As packets do not go out of the computer, they are sent to localhost (the emulator's own loopback interface) and to the port <host_port> indicated. This instruction redirects them to 10.0.2.15 (the emulator's own network/Ethernet interface) and to the port <guest_port>, by which the application should receive packets.

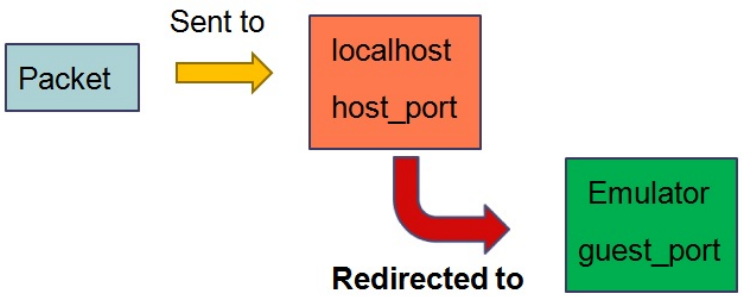

Figure 43: Port redirection performed in the emulator.

For example, if we connect via telnet to the first emulator and execute the instruction *redir add udp: 8080:7777*, all UDP packets which destination is localhost:8080 will be redirected to port 7777 of the first emulator (the application should expect to receive incoming packets on port 7777).

Once the listener ports have been redirected, the two emulators will be virtually connected, being able to send and receive packets between them; these packets must be sent to the address **10.0.2.2**, which corresponds to the virtual emulator's loopback interface, mapped as 10.0.2.2 in the virtual router.

In addition to this redirection needed for two emulators to communicate, there are certain lines of code that vary according to whether the application is running in an emulator or in a real device, for example, the destination IP address in the emulator case is 10.0.2.2, while when the application is tested in real devices the destination address will be the IP address of the network the phone is connected to; this implies having to define sockets in a different way (in programming sense, this is equivalent to use different socket constructors), associating them to different IP addresses depending on the connection type. Therefore, there are differences in the code of the application depending on the environment where the application is loaded, whether emulator or real device.

To view information about the real-time operation of the application (via Eclipse) while the communication is running, it is possible to use the debugging tool that Android provides called Dalvik Debug Monitor Server (DDMS). In section LogCat of this tool we can see messages about the current state of the application. Additionally, some custom-made code can be inserted to see messages in LogCat which allow the programmer to debug the program functioning and take care of its correct behavior.

3.3.2 WiFi

To perform the transmissions in a WiFi environment, two mobile phones with Android operating system running and an intermediate router are needed. The two terminals are connected to the network that the router provides, and the communication between them will take place through the router.

In the practical implementation during this thesis, two ZTE Blade devices and a TP-Link TL-WR841N router have been used. This router complies with the standard IEEE 802.11n.

Figure 44: Two Android devices connected via WiFi.
Source of the devices: http://www.informatica-hoy.com.ar/smartphones-android/ZTE-BLADE-smartphone-android.php
Source of the router: http://www.catthanh.com/z_pro646

Once the devices are connected to the WiFi network, the data sending is carried out by fixing as destination IP address the address that the router has assigned to the target mobile phone where data is going to be sent; the packets go to the router, who will redirect them to the destination mobile phone because it belongs to the same subnet.

3.3.3 3G

To perform tests in the 3G environment, a hybrid scheme of 3G and WiFi has been implemented, as shown in the figure below:

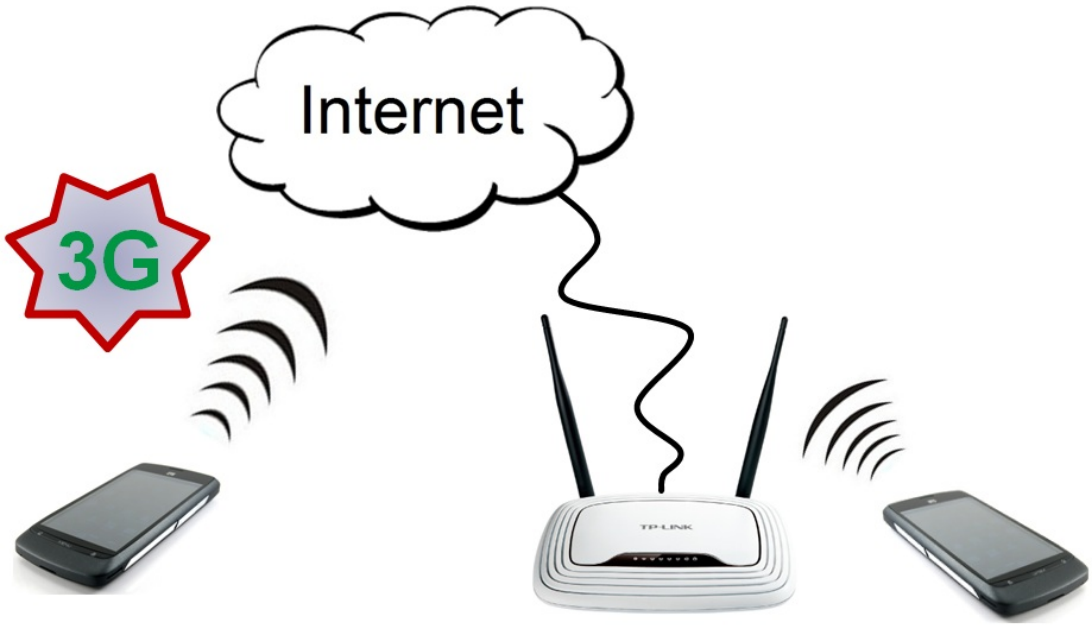

Figure 45: Two Android devices connected to 3G and WiFi respectively.
Source of the devices: http://www.informatica-hoy.com.ar/smartphones-android/ZTE-BLADE-smartphone-android.php
Source of the router: http://www.catthanh.com/z_pro646

One of the two phones is connected to the 3G network while the other is connected via WiFi to a router which is also connected to the internet. The device which is connected via WiFi has an external IP address visible from the internet, and that is the destination address where the phone connected to 3G must send the packets. Thus, packets are routed through the 3G network to the destination.

However, the transmissions performed in this environment had a problem: the communication was cut a few minutes after it started. Looking at the LogCat (where problems are reported on the current status of the application) it was found that there was a Dynamic Host Configuration Protocol (DHCP) that caused the end of the connection. Under this procedure, the IP address provided by the 3G network operator (i.e., which characterizes the terminal connected to 3G) changes, so the application cannot continue transmitting data to (or from) an IP address which no longer belongs to him.

This problem influenced in that, in the transmissions using 3G, less packets were sent than in the transmissions in other environments (emulator and WiFi). For this reason, the results obtained using 3G are more variant and, in some cases (analysis of video traffic in 3G), there could not be representative conclusions extracted because of the low number of packets transmitted successfully.

Chapter 4

Results and interpretation

4.1 Results obtained

After performing the measurements for each type of traffic in each of the three environments, the text files that the application itself generated have been extracted and analyzed with the mathematical program Matlab [5]. Based on these data, graphs have been plotted, which compare the traffic behavior in the three environments and with the ideal profile that they should present. Additionally, communication features such as packet loss, average bytes transmitted and received, average elapsed milliseconds between two successive packets or the energy distance between two graphs have also been calculated. The results have been interpreted and interesting conclusions have been extracted from them.

The results obtained will be analyzed and classified by the data traffic type tested (video, web, online games and Skype), highlighting the most important aspects of each analysis. Afterwards, conclusions can be drawn on how each of the connection types (emulator, WiFi and 3G) treats the sending and receiving of data using the Android operating system. In Appendix A, there is additional information available on each type of traffic for each connection type, treated individually.

4.2 Video

The first traffic to be analyzed is video. Specifically we will focus on the transmission of the video itself, rather than in the request of it. More detailed information on how video traffic was generated can be found in section 3.2.1.

It should be noted that to emulate this type of traffic in the 3G environment there were problems, because the connection got cut after a while the transmission was started, due to a DHCP; in addition, some of the emulated video packets had a big size (thousands of bytes), which meant that when the connection was cut, very few packets had been sent to the destination (about 10 packets), so the video transmissions in 3G have not been taken into account, since there was not a significant number of samples to draw representative conclusions.

So in the case of video, it is going to be studied the **size of packets** sent and received only for the **emulator** and **WiFi** environments, comparing them with the ideal curve that they should follow.

The graphs presented below show the size of video packets sent and received in the emulator (in blue), in WiFi (in red) and the ideal theoretical curve that they should follow (in black). The horizontal axis shows the packet size in bytes, while in the vertical axis there is represented the probability with which each one of them appeared. In the graphs shown, the values of the horizontal axis are discrete, with steps of one byte, so they are representing PMFs. However, when drawing them in Matlab, the samples have been united with lines, so in practice they become PDFs.

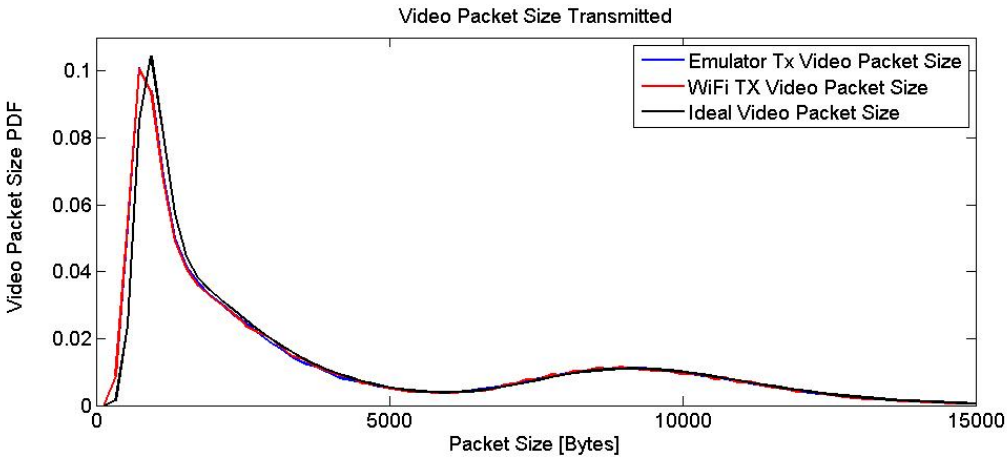

Figure 46: PDF of the Video packet size Transmitted.

Regarding to the size of the packets **sent** it can be seen that both in emulator and in WiFi, the curves are overlapped and close to the ideal, considering this a correct behavior (later in this section, the calculation of the energy distance between two curves will corroborate this statement).

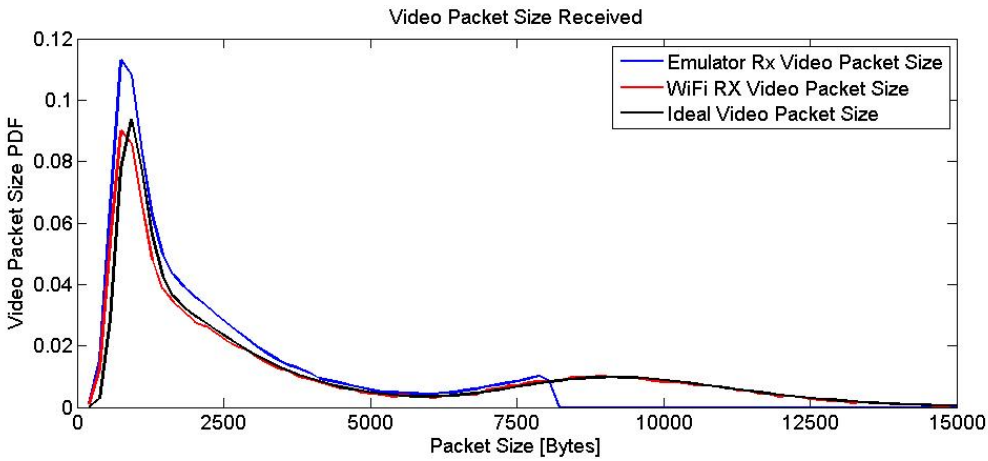

Figure 47: PDF of the Video packet size Received.

Looking at the figure on the size of the packets **received**, it can be observed that in the case of WiFi, the ideal curve is faithfully followed (with a packet loss of only 0.12%); while in **emulator** there is a two-stage behavior to be highlighted:

- The **peak** in the received packets is even **higher** than in the case of packets sent.
- Packets **bigger than** 8192 bytes do not reach the destination, as the graph drops drastically at that point.

The interpretation that can be deduced from this peculiar behavior is that the transmissions performed in emulator where packet size are **larger** than 8 kilobytes (Kbytes) (that is, 8192 bytes) are silently discarded, probably because there is an internal buffer that prevents packets larger than this size to be sent.

This aspect is also reflected in the average size (in bytes) of packets sent and received:

	Average bytes transmitted	Average bytes received
In emulator	4011.5	2377.1
In WiFi	4011.6	4008.4

Table 8: Average Video bytes transmitted and received in emulator and WiFi environments.

In addition, **packet loss** in emulator between packets sent and received is 20.26%, which proves that packets larger than 8 Kbytes are **lost** because they never reach their final destination.

It is going to be calculated the **energy distance** between each of the curves and the ideal curve. The energy distance between two probability distribution curves is a quantitative way to measure how similar two curves are.

$$\int_{-\infty}^{+\infty} (F(x) - G(x))^2 \, dx$$

Equation 8: Formula to calculate the energy distance between two probability distributions.

Let *f(X)* and *g(X)* two probability **density** functions, *F(X)* and *G(X)*, must be calculated, which are **cumulative** distribution functions (CDF). Thereafter, subtract the two CDFs sample to sample, square it and integrate it from minus infinity to plus infinity. The unit of the result is the same unit that the probability distribution function is referred to (in this case, bytes).

Thus, the more bytes of energy distance between two curves, the greater its **statistical distance** will be. Energy distances between the transmitted and the ideal curve and between the transmitted and received curves are shown in Tab. 9.

	Emulator	WiFi
Energy Distance Ideal-Tx [Bytes]	0.0101	0.0096
Energy Distance Tx - Rx [Bytes]	1.2675	$3.4771\ 10^{-6}$

Table 9: Energy distances between the probability distribution curve of transmitted packets and the ideal curve, and between the transmitted and received curves for video traffic.

When comparing the transmitted curves with the ideal one, the resulting energy distance is very small, which means that both transmitted curves are close to the ideal. However, when comparing the transmitted and received curves, the energy distance in the emulator increases and gets to be six orders of magnitude bigger than in the case of WiFi.

This behavior allows us to guess that the emulator environment will not be suitable for sending large sized packets.

In terms of time between packets, the WiFi environment offers better performance than the emulator. In WiFi, the required delays are achieved fairly accurately, while the emulator takes TBPs always higher than those required. A further analysis on the time between packets is performed in section 4.4.

4.3 Web

About web traffic, it is going to be analyzed more deeply the **size** of the packets in the traffic that emulates the transmission of web pages. More detailed information on how web traffic was generated can be found in section 3.2.2.

According to the emulated web traffic, the way of working consists on sending a main object and various embedded objects, emulating after, the time that it takes the user to digest the web page. The protocol used is TCP.

The following images show the size of packets sent and received for Web traffic, while the horizontal axis represents the size of packets, and the vertical axis the probability of occurrence; the blue line indicates the behavior of the emulator, the red corresponds to WiFi, the green line to 3G, and the black curve represents the ideal shape to which the rest should fit.

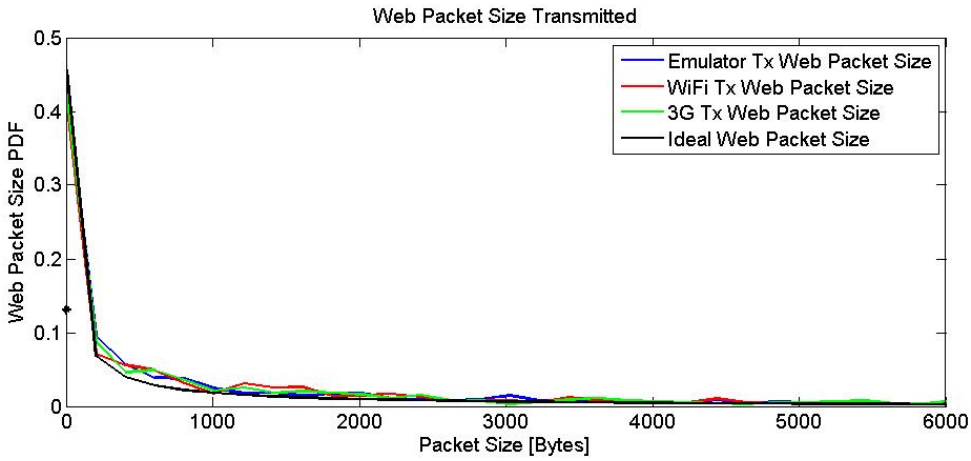

Figure 48: PDF of the Web packet size Transmitted.

It can be observed as in the three environments, the theoretical curve is followed in a fairly faithful way. The dot that stands on the vertical axis represents the behavior of the state *Reading*, characterized by sending a symbolic byte; thus, it matches the theoretical point (they are overlapped, one upon each other) so we know that the transmission was successful, having reached the state *Reading* as many times as expected.

The graph with the statistics about the received packet sizes is shown below:

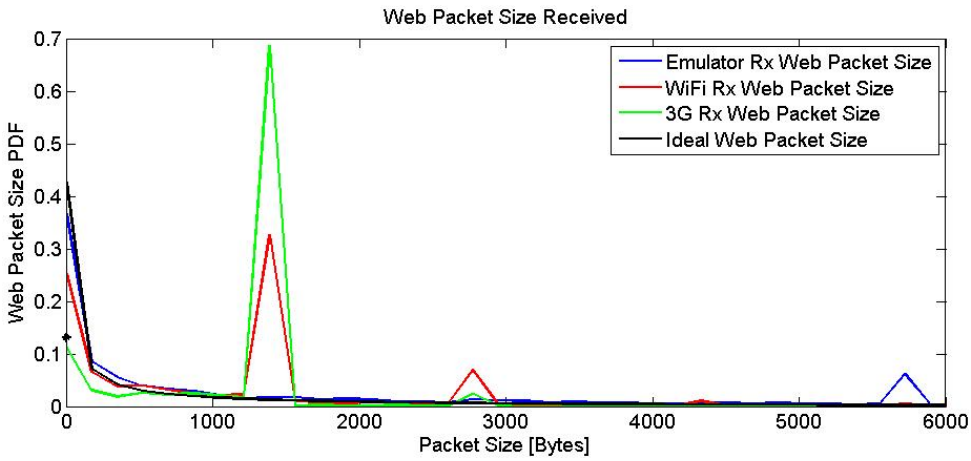

Figure 49: PDF of the Web packet size Received.

When comparing the received packets' size, four very pronounced peaks can be observed:

- In 1412 bytes there are the two major peaks: the highest referred to 3G, and the other to WiFi.
- In 2824 bytes, there are also two peaks of 3G and WiFi, but this time the one related to WiFi is higher.
- In 4236 bytes there is a small peak of WiFi.
- In 5648 bytes there is a peak belonging to the emulator.

In addition to that, it surprises that the packet loss in the three environments are **negative**:

	Packet loss [%]
In emulator	(-10.19)
In WiFi	(-63.97)
In 3G	(-63.36)

Table 10: Packet loss while transferring web traffic.

A negative packet loss means that there have been more packets received than sent, which seems, a priori, impossible. What is happening is that there has been a TCP fragmentation of 1412 payload bytes, this is, with a Maximum Transfer Unit (MTU) of 1412 payload bytes. When the sender sends packets bigger than 1412 payload bytes, they are fragmented by TCP as follows:

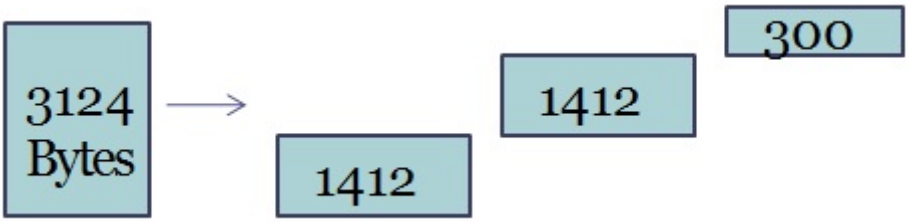

Figure 50: TCP fragmentation occurred while transferring web traffic.

If for example a packet with 3124 payload bytes is sent, those bytes are fragmented into various fragments of 1412 bytes (in this case 2), and a packet with the remaining bytes (300). That is the reason why as many packets of 1412 bytes have reached the destination, due to this fragmentation.

Additionally, the occurrences of peaks at **multiples of 1412 bytes** (2824, 4236 and 5648) are because, when fragmenting, the various packets of 1412 bytes in which the main packet is decomposed arrive together to the destination. Most probably they are separated less than 100 milliseconds, causing them to get congregated in the receiver and get considered as one single packet, thereby increasing the number of packets received with lengths **multiples** of 1412 bytes.

Note that packet loss in emulator (-10.19%) is much lower (in absolute value) than in WiFi (-63.97%) and 3G (-63.36%). This may be because TCP fragmentation in the emulator is applied with a MTU greater than 1412 bytes, making a lower percentage of the overall packets to be fragmented.

These fragmentations get reflected in the average size of packets sent and received. The table shown below demonstrates it:

	Average bytes transmitted	Average bytes received
Emulator	1998.2	1811.9
WiFi	2041.9	1245.3
3G	2038	1222.9

Table 11: Average size of web packets transmitted and received.

There are very pronounced and remarkable **decreases** on the average bytes received in WiFi and 3G (40%) compared to 10% in emulator. These decreases have appeared because the lengths of the received packets have been redistributed, according to the TCP fragmentation.

The fragmentation has led to an alteration of the time between packets: the **average delays** have **decreased** about 40% (in reception respect to transmission) in WiFi and 3G (where the fragmentation was greater), possibly because the resulting fragments are sent closer in terms of time, with a consequent reduction of the average value.

4.4 Online games

For the analysis about traffic that emulates the behavior of online games, the time between packets has been emphasized. In this kind of traffic, this is a **fundamental** feature for the proper functioning of the game, as there is a continuous contact with the server and it must be fluid.

The part of the communication analyzed is the downlink, this is, packets going from the server of the game towards the user who is playing the game on an Android device. More detailed information on how online games traffic was generated can be found in section 3.2.3.

In this implementation, all the packets are temporarily equally spaced 50 milliseconds.

The following graph shows the statistics of the time between two consecutive packets **sent**, in the three environments (emulator in blue, WiFi in red and 3G in green), and the ideal constant value (in black).

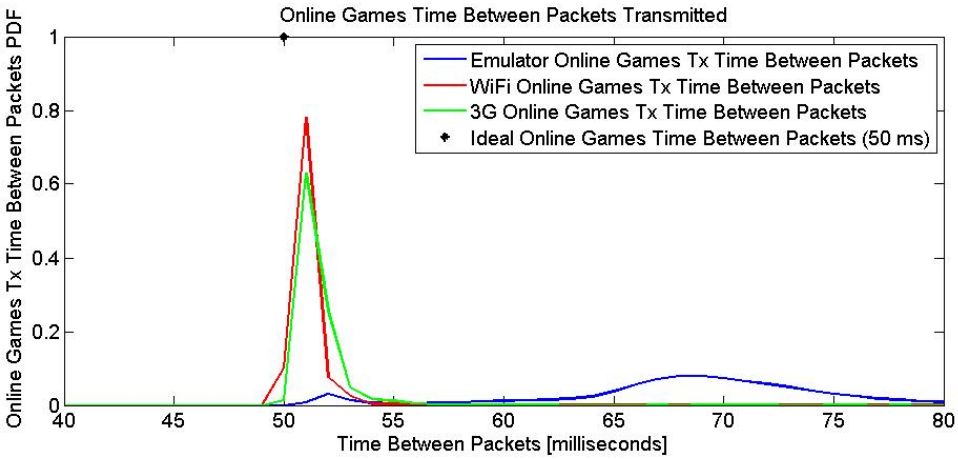

Figure 51: PDF of the transmitted time between packets in Online Games.

The ideal value that all packets should follow is 50 milliseconds, represented by a black dot in Fig. 51. On this basis, it can be verified as the red line (corresponding to WiFi) is the closest to the ideal shape, followed by the curve that represents 3G; in both, most of the times between packets are between 50 and 55 milliseconds, which can be considered within acceptable limits. However, the blue line (representing the emulator's behavior) is far from the value of 50 milliseconds, and the TBPs generated in this environment are far greater. Here is a table showing the average values generated for each connection type:

	Average time between packets transmitted [milliseconds]
In emulator	75.9357
In WiFi	51.7878
In 3G	52.3900

Table 12: Average TBPs transmitted in Online Games.

In emulator, TBPs are much bigger than required (50% higher, on average). This behavior has been repeated in the rest of the tests carried out: the emulator does not respect the established times between packets and tends to enlarge the values. Is for this reason why the transmissions performed in emulator took longer time than within other connection types.

Three **reasons** why the emulator enlarges the delay values may be these:

- The Java threads are managed by the DVM, assigning execution times to each thread depending on its priority. If two threads have the same priority (as happens in the present case), while one thread is executing an instruction, the other is waiting, which may cause the time between two instructions to be randomly bigger.
- The *sleep* function (method belonging to the Java threads) is the one that allows a thread to stop its execution for a certain number of milliseconds. In the explanation of its operating mode, it is exposed that "precision is not guaranteed", so an exact behavior in this sense is not ensured.
- The virtual router that connects the emulators may have influenced adding some delay during transmissions.

Although the generation of TBPs in the emulator is not the desired, it has been studied how these values vary when packets are transmitted. Let's look at the statistics for the time between packets on the **receiving** side.

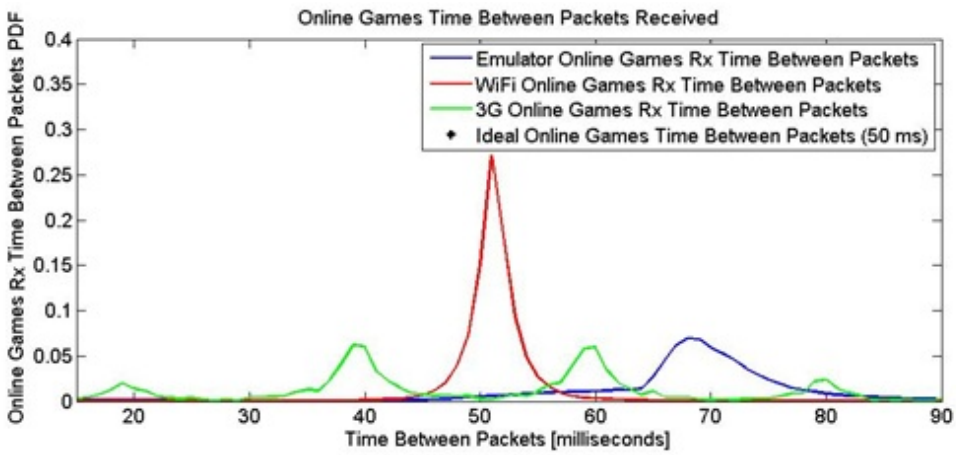

Figure 52: PDF of the received time between packets in Online Games.

Note that the ideal value is not visible because it is at the height of the value 1 in the Y axis. The image was zoomed so that the colored lines could be clearly visible.

Furthermore, the table with the average TBP values calculated at the end of the transmissions is shown below:

	Average time between packets received [milliseconds]
In emulator	75.9356
In WiFi	51.8364
In 3G	52.2559

Table 13: Average TBPs received in Online Games.

Let's analyze now the time between packets **received**, classified according to the connection type:

- Emulator: TBPs are still much bigger than 50 milliseconds, which is an undesired behavior. However, it is important to emphasize that the average TBP is almost the same, both in packets sent and received. This means that the average TBP is maintained, despite being displaced almost 26 milliseconds. So, the emulator shifts the average TBP almost 26 milliseconds, but this value remains the same in emission and reception.
- WiFi: On most of the transmissions (around 90%) packets arrive at the destination separated between 45 and 55 milliseconds (considering this interval as the margin of confidence). In addition, the average time is still very close to the desired 50 milliseconds. This is the connection type that presents a better behavior.
- 3G: The values in this environment are drastically different in emission and reception: while when transmitting packets there was a proper functioning, when receiving them it is observable as they have changed very much and are much more dispersed. However, the **average value** remains around 52 milliseconds. Looking at the graph, it can be seen as there are two peaks of similar height at 40 and 60 milliseconds, and two more at 20 and 80, which causes the average value to be placed close to 50 milliseconds. However, more than 95% of the values are different to 50ms and the 80% are out of the margin of confidence. So, even if it

seems that in terms of average TBP value 3G behaves correctly, actually it is not true. For the case of online games, where it is desired to have a constant delay **in all the packets**, this is not a proper way of working.

Summing up: The delay in emulator is too large; in WiFi the behavior is pretty correct; and in 3G the ideal value is achieved **in average**, but basing on taking bigger or smaller delays (out of the margin of confidence), something **undesirable** for online games traffic.

Regarding the size of the packets, a good behavior is followed in the three environments, having resulting curves very similar to the ideal, both in transmission and in reception.

4.5 Skype

The most interesting aspect to analyze and where major conclusions can be extracted from Skype traffic is the **packet size**. More detailed information on how Skype traffic was generated can be found in section 3.2.4.

The resulting PDF will be divided into 3 sections, corresponding to the possible lengths transmitted: or 1 byte, or 40 bytes, or a value uniformly distributed between 453 and 488 bytes.

The graph about the sizes of the **transmitted** packets is shown below.

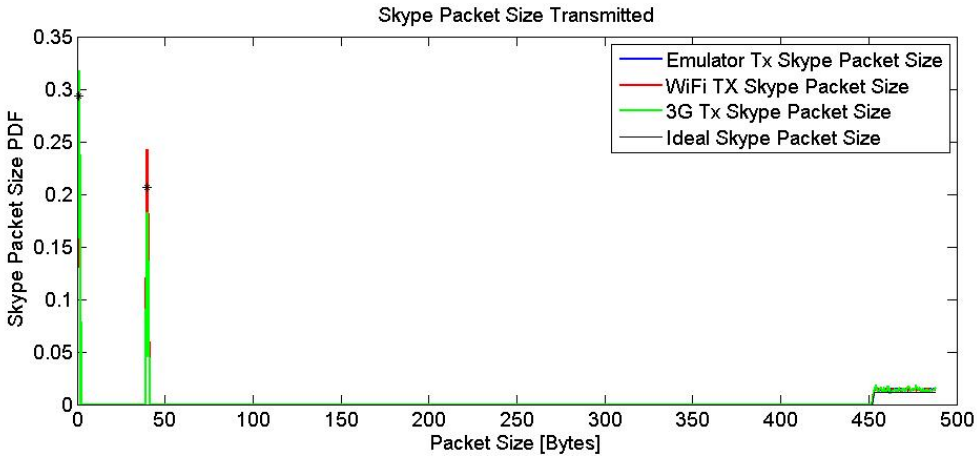

Figure 53: PDF of the transmitted packet size in Skype.

Given the low resolution observed in this image, a detailed analysis is going to be made, separating the case where two constant values appear (corresponding to Audio) from the part with the uniform values (corresponding

to Video). To do this, the PDFs **transmitted** and **received** are going to be compared.

Let's begin with the PDF corresponding to the **Audio** values **transmitted**.

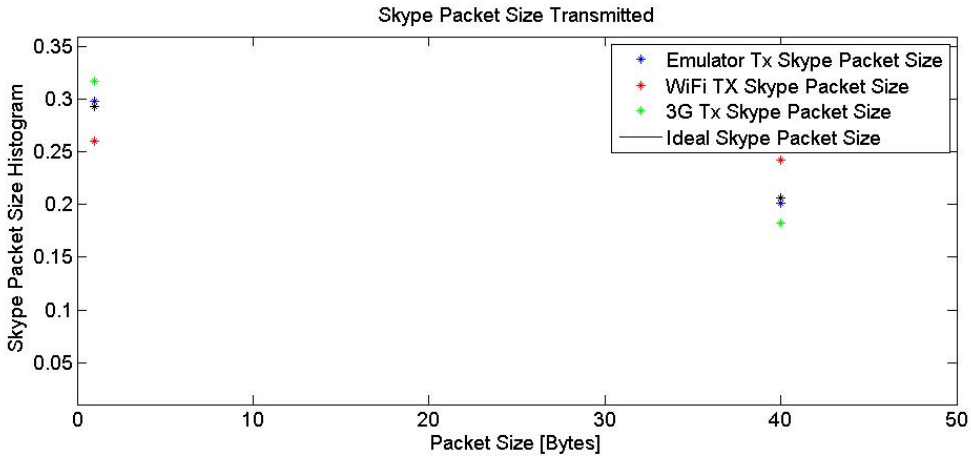

Figure 54: PDF of the transmitted Audio packet size in Skype.

And get compared with the **Audio** values **received**.

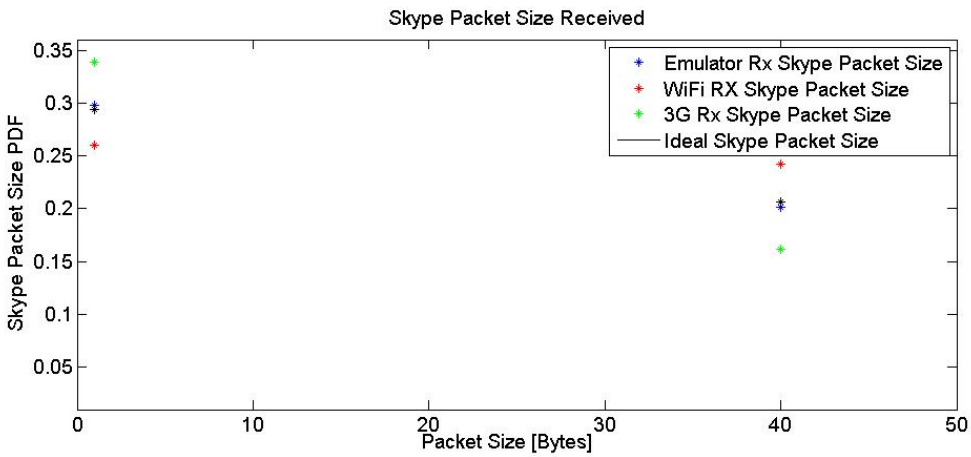

Figure 55: PDF of the received Audio packet size in Skype.

It can be observed that both graphs are almost identical, except that 3G samples **move away** from the ideal value: the sample at 1 byte becomes higher (this is, occurs with higher probability) and the sample in 40 bytes descends (decreases its probability of occurrence), getting compensated. The reason for this shift may be the lower number of transmissions in 3G (implying shorter tests) than in other environments (100.000 packets sent in the emulator in front

of 4.000 in 3G), due to the problems encountered and discussed in section 3.3.3. If the transmissions were longer, the samples would tend to match the ideal value.

On the other hand, let's consider the PDFs about the lengths of Video packets, distributed uniformly. The following figure stands to the size of **Video** packets **transmitted**.

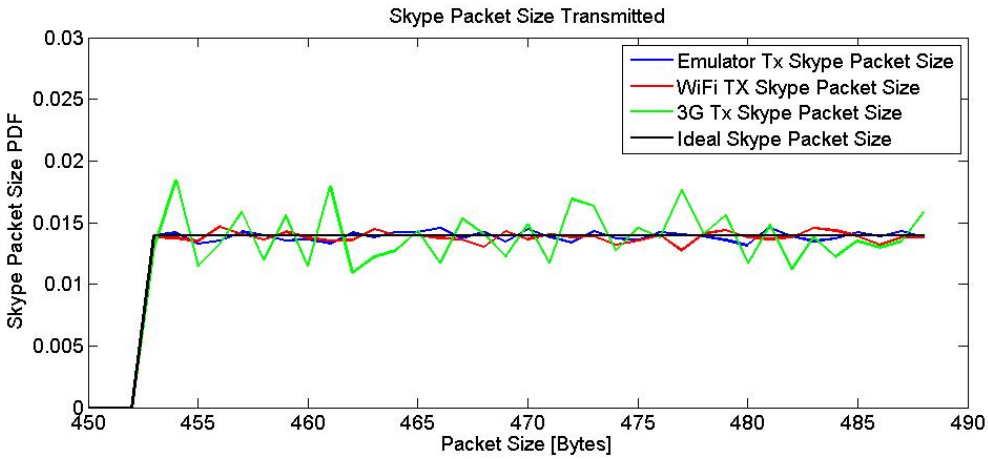

Figure 56: PDF of the transmitted Video packet size in Skype.

While the next one represents the size of **Video** packets **received**.

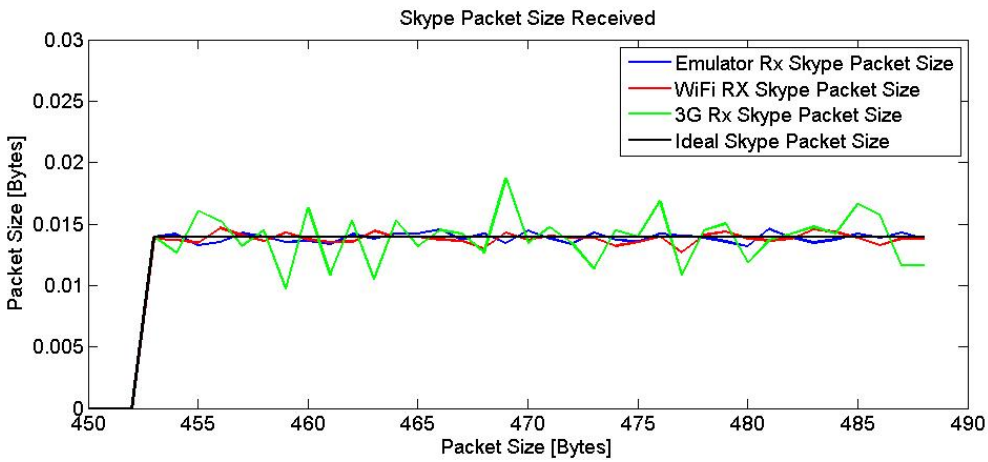

Figure 57: PDF of the received Video packet size in Skype.

The graph corresponding to the emulator is identical in transmission and in reception; in WiFi both graphs have a very similar profile (although not

identical), while in 3G the samples are more dispersed and the curve is more sharpened.

This variability of samples observed in 3G is due to the low number of transmissions achieved successfully using this connection type, because the more extensive the transmissions were, the more likely was the curve to resemble a horizontal line (as well as blue and red curves also tend to be a straight line). This behavior can be demonstrated by calculating the **energy distance** for the three graphs with respect to the ideal curve.

Let's see how much energy distance is there between the statistical functions **received** in each environment respect to the ideal uniform line.

	Energy distance [bytes]
Emulator - Ideal	$4.1017\ 10^{-5}$
WiFi - Ideal	$5.4702\ 10^{-5}$
3G - Ideal	$7.1851\ 10^{-4}$

Table 14: Energy distances between the probability distribution curve of each environment and the ideal curve, for the received packets in Skype traffic.

The values of the energy distances in the emulator and WiFi are really low, while the energy distance in 3G is almost one order of magnitude higher. It is because of the lower number of iterations, causing the curve to be more peaked in 3G and thus more separated from the ideal. However, the three values can be considered very low, since the three curves tend to flatten out increasing the number of iterations, and their shape tend to fit faithfully the ideal flat line.

Referring to the time between packets, the aforementioned behavior is repeated again: in the emulator, the delays are too big to be acceptable; in WiFi values faithful to the ideal behavior are produced; and in 3G the ideal value is reached in average, but based on taking delays out of the margin of confidence.

Chapter 5

Conclusions and future work

5.1 General conclusions

From the results obtained in the real tests, conclusions are going to be extracted about how Android treats the transmission and reception of packets. These conclusions will be targeted in each connection type, analyzing separately the emulator, WiFi and 3G. Thus, we will be able to obtain a general idea on how each of the three environments treats packets, and will also be able to draw a final conclusion surrounding the best and worst aspects of each connection type. Finally, it will be concluded which of the three environments is more appropriate for the transmission of each of the traffic types tested (video, web, online games and Skype). As the last section and in views to the future, future tasks that could be performed using the traffic generator implemented are proposed, which would be useful for a deeper analysis on the Android scenario or possible new areas of research.

5.2 Conclusions about emulator

The emulator, a priori, should be the most reliable connection type of the three tested, as transmissions that occur therein are not sent away from the computer. This environment serves to test the applications on the computer before doing it on real devices. The analysis will be segmented into three parts: how the emulator behaves towards the size of the packets, how treats the time between packets and finally, general conclusions on its overall functioning.

5.2.1 Packet size in emulator

Firstly, let's consider the results derived from the sizes of the packets in the emulator. The most important conclusion that has been drawn empirically is the existence of an internal buffer in the emulator that prevents sending packets bigger than 8 Kbytes (8192 bytes). These packets become to be lost packets, because the emulator apparently discards them silently. That is why the emulator is not a good connection type for sending big sized packets.

However, the behavior of the emulator with **small** packets is **trustworthy** and **reliable**, because graphical comparisons between the sizes of packets sent and received result very similar (for small packets, the energy distance in the worst case is lower than 0.27 bytes).

It is also remarkable the fact that **TCP fragmentation** observed when analyzing web traffic does not affect the emulator as much as it does in the other two cases. There are **less** fragmented packets and a **smaller decrease** in the average size of the packets.

5.2.2 Time between packets in emulator

Regarding to the time between packets, there are two important aspects to discuss, one of them positive and the other negative:

- The positive side is that the TBP of transmitted and received packets are very similar **in average value** and keeps almost equal. However, in practice, the received values are **more dispersed** than initially transmitted values.
- The negative side is that the TBPs that the user sets in the application **are not** those transmitted by the emulator; these transmitted values tend to be **bigger** than those required (in the case of online games, up to 50% higher), thus obtaining undesirable results. Moreover, this causes the transmissions to last more time than expected.

Thus, since the TBP set by the user is not generated by the emulator and, additionally, these values are expanded, we can conclude that the emulator **is not** a good environment for testing traffic types which require an **accurate** temporal behavior.

5.2.3 General conclusions about emulator

The conclusions on the emulator can be summarized in these main points:

- The emulator has an **internal buffer** of 8 Kbytes which prevents sending bigger packets.
- It exhibits a good transmission-reception performance with **small** packets.
- It is **not** a suitable connection type for testing **real-time** applications or applications with **precise timing requirements**.

5.3 Conclusions about WiFi

The WiFi environment, in general, offered **best performance** throughout the transmissions. The packet loss has been very small and by the results obtained we can conclude that it is a trustworthy connection type.

5.3.1 Packet size in WiFi

The treatment of packet size on the part of WiFi has been very correct. When using the **TCP** protocol, a **fragmentation** is performed; in this case, packets larger than 1412 bytes are fragmented, preventing the sending of bigger packets in one single transmission. As a result, in the reception side it appeared a big amount of packets with sizes **multiples** of 1412 bytes and the average size of the packets were reduced by 40%.

When sending packets, WiFi has always respected the statistics that the user selected in the application, sending packets which fitted the corresponding statistics.

When receiving packets, the losses have been very tiny and the curves obtained have always been very close to the ideal ones, despite having a real channel which could have influenced in changing its profile.

5.3.2 Time between packets in WiFi

In a matter of time between packets, WiFi has proved to be **the best** environment. As can be deducted from the analysis of online games traffic, the values of TBP achieved when transmitting packets are those which show the **best profile** compared to the rest of connection types, as well as happen with the received packets, with the 90% of its values placed in a margin of confidence, that is, 10% greater or lesser than the desired value.

In terms of TBP, WiFi is the connection type which offers a better performance.

5.3.3 General conclusions about WiFi

Therefore, the final conclusions deductible from WiFi are the following:

- WiFi is a good environment to run **exacting packet-size** applications.
- In TCP, there is a **fragmentation** of 1412 payload bytes.
- WiFi is recommended for required **time-precision** apps.
- The ideal shape is followed really **precisely**.

5.4 Conclusions about 3G

During the real tests carried out in 3G, it has to be reminded that a lower number of transmissions were performed, due to a DHCP that prevented the continuity of communication. This aspect gave rise to observe that the resulting

graphs were more irregular, even not allowing to extract reliable conclusions when video packets were transmitted. Maybe that is the reason why this one is the connection type where a biggest packet loss was obtained in all transmissions.

5.3.1 Packet size in 3G

The behavior of 3G is **good** for **small** packets, as the profile of the curve tends to fit the ideal curve. However, 3G does not respond too well when sending big packets through either of the two protocols tested:

- TCP, where the fragmentation appears with an MTU of 1412 bytes, with a **greatest incidence in 3G** than in other environments.
- UDP, where the transmissions when emulating video traffic could not be completed due to the abovementioned DHCP.

5.3.2 Time between packets in 3G

Referring to the time between packets, the 3G environment offers the second best behavior when transmitting packets, but a bad behavior when receiving them:

- The TBP established by the user in the application is accurately generated when transmitting packets.
- However, at the time of receiving packets, these TBPs are modified, taking upper and lower values (that is, spread values) in such a way that the **average TBP received** coincides with the average TBP of the transmitted packets. This suggests that a **strong** but **symmetric spread** of TBP is taking place in 3G. This behavior is not suitable for those applications where a **constant and fixed TBP** (without fluctuations in its value) is required.

5.3.3 General conclusions about 3G

To sum up, the global conclusions deducted on the 3G environment are the following:

- **Higher packet loss** than in the rest of connection types.
- **Small** packets **successfully** transmitted, but does **not** offer a good behavior with **big** packets.

- A **fragmentation** is carried out when using **TCP** protocol, with an MTU of 1412 bytes.
- The presence of DHCPs avoided performing successfully long transmissions.
- 3G is **not** a suitable connection type for applications with **temporally equally spaced** packets (needing **constant timing requirements**).

5.5 Most suitable environment for different traffic types

Basing on the conclusions about the behavior of each of the three connection types, we are going to conclude which environment is the one that performs better for each of the emulated traffic types.

In Tab.15 it is shown schematically which connection types present a good behavior (represented as a green tick) or misconduct (symbolized with a red cross) for each of the emulated traffic types.

It is considered misconduct when a connection type meets one or more of the following conditions:

- The packet loss is bigger than 10%.
- The energy distance between the transmitted and received curves is greater than 1 byte.
- The average PS/TBP received increases or decreases more than 10% respect to the transmitted average.
- The average PS/TBP transmitted is 10% larger or smaller than the value selected by the user.

	Emulator	WiFi	3G
Video	✗	✓	-
Web	✓	✗	✗
Online Games	✗	✓	✗
Skype	✓	✓	✓

Table 15: Comparison on the good or bad behavior in each environment for each emulated traffic type.
Source of the green tick:
http://vtiperformance.com/BusinessCommunicationSkills/3stepskillerpresentation.html
Source of the red cross: http://dotd.wikia.com/wiki/File:600px-Red_x.svg.png

- The **WiFi** environment is really suitable to transmit **video** traffic.
- The **emulator** is the connection type where **web** traffic was better transmitted, although this is an environment for testing applications only.
- To play **online games,** the best is to do it through a **WiFi** connection.
- To make video-calls using **Skype**, **any of the three connection types** would be a good option.

5.6 Future work to be done

In order to get the most profit from the traffic generator implemented, possible future tasks are suggested. Given the great versatility of the application, it can be reused for other purposes and take advantage of its many possibilities. Here are some possible additional utilities for which the traffic generator could be useful:

- Emulate other types of traffic, basing on the eligible statistics already implemented in the traffic generator.
- Reprogram the application to allow the user to choose from a wider range of statistics and more states in the Markov chain. In this way, more complex traffic types could be emulated.
- Send, at the same time, traffic to one single device from more than one mobile phone to check if a multiple reception affects the proper functioning. There can be one phone hosting multiple traffic types, but also multiple phones hosting one traffic type each.

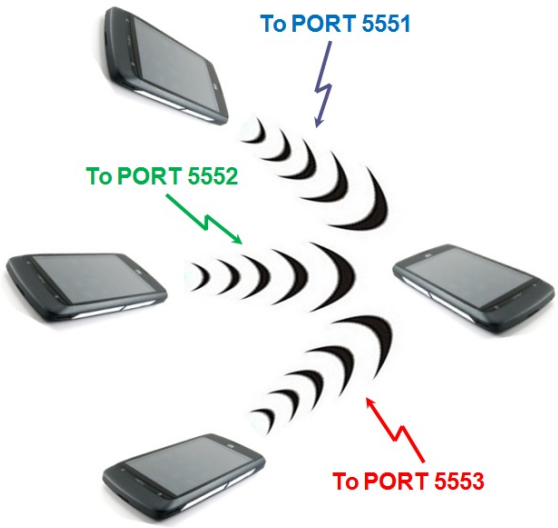

Figure 58: Various sender devices transmitting to different ports of one receiver device. Source of the devices: http://www.informatica-hoy.com.ar/smartphones-android/ZTE-BLADE-smartphone-android.php

- Develop Android applications taking into account the conclusions drawn in this thesis. The application would be optimized for a proper performance, taking advantage of having a prior empirical and experimental knowledge on how Android behaves sending and receiving packets.

Acknowledgements

En primer lugar, me gustaría darle las gracias a mis supervisores Markus Rupp, Markus Laner y Philipp Svoboda por darme la oportunidad de realizar esta tesis, enfocada a un tema que me interesa tremendamente. He de decir que ha sido un auténtico e inmenso placer realizar mi tesis en la Universidad Técnica de Viena (TU Wien) y que estoy muy orgulloso de ello. Merece mención especial la ayuda que obtuve de Josep Colom desde Viena, ya que él contactó con la universidad y fue quien consiguió que pudiera realizar un proyecto de estas características.

Quiero dar las gracias también a cuatro personas con las que tuve la suerte de compartir estancia en la universidad de Viena y día tras día estuvieron a mi lado, tanto para ayudarme en los momentos más difíciles como para compartir las experiencias más enriquecedoras. Ellos son David Vargas, Alexandra Martínez, Amalia Lorca y Javier Rodas.

De una manera muy especial, quiero agradecerle de todo corazón a Lena Tillman el apoyo incondicional que me dio desde Barcelona. Sin los ánimos que ella me dio constantemente, esta experiencia no hubiera sido ni de lejos tan productiva y enriquecedora. Ella siempre estuvo ahí para echarme una mano cuando lo necesitara, y eso para mí tiene un valor incalculable.

He de agradecer a Jordi Vallcorba, Mariona Roca y Henar de la Rosa que la experiencia Erasmus fuese más positiva y placentera de lo que nunca imaginé. Gracias por compartirla conmigo, sinceramente fue un enorme placer.

Agradecer también la ayuda prestada a la hora de solventar dificultades con mi tesis que me dieron Joan Roca, Sergi Gimeno y Juan Luís Gorricho. Siempre hay alguien que sabe más que tú y te puede ayudar, y en este caso fueron ellos.

Y por último, me reservo los agradecimientos más grandes que una persona puede dar para mis padres Alfonso y Carmen, quienes hicieron que una de las experiencias más increíbles de mi vida se hiciera realidad, y también porque siempre creyeron en mí y confiaron plenamente en mis capacidades para que pudiera sacar adelante cualquier dificultad que se cruzara en mi camino. Sois los mejores padres del universo y os quiero infinitamente.

Bibliography

[1] D. Parra Valcarce, *"Ciberperiodismo móvil: el peso específico de la cuarta pantalla en el panorama informativo internacional"*, Dialnet OAI Articles, July 29th. 2011.

[2] J. Schneidawind, *"Big blue unveiling"*, USA Today, November, 1992.

[3] Google Inc. is a multinational corporation dedicated to develop software and internet-based services and products. URL: www.google.com.

[4] Open Handset alliance is a consortium of 84 firms to develop open standards for mobile devices. URL: http://www.openhandsetalliance.com.

[5] Matlab is a numerical computer environment, developed by MathWorks. URL: http://www.mathworks.de.

[6] Eclipse is an integrated development environment for Java developers. URL: http://www.eclipse.org.

[7] Java is a general-purpose, class-based, object-oriented programming language. URL: http://www.oracle.com/technetwork/java/index.html.

[8] J.F. Kurose, K. W. Ross, *"Computer Networking: A Top-Down Approach"*, 5th edition. Boston, Pearson Education, 2010.

[9] RFC 793, September 1981.

[10] AT4 wireless S.A., *"Histograms and CDFs for packet size and inter-arrival time in PC-based online games"*, E-mail discussion [75#34], 2011.

[11] Marwan Krund, Herman Hughes, *"A traffic model for MPEG-Coded VBR Streams"*, Michigan State University, 1995.

[12] Shahar Hauzner, *"Data Traffic Model for Wireless Coexistence in US 3.65GHz band Simulation"*, IEEE P802.19 for wireless coexistence, Tel-Aviv, Israel, January 1st, 2008.

[13] C. Grinstead, J. Snell, *"Introduction to Probability",* American Mathematical Society, 2006. URL: http://www.math.dartmouth.edu/~prob/prob/prob.pdf.

[14] WiFi (or Wi-Fi) is a trademark of the Wi-Fi Alliance and the brand name for products using the IEEE 802.11 family of standards.

[15] 3G (or third Generation mobile telecommunications) is a generation of standards for mobile phones and mobile telecommunication services fulfilling the International Mobile Telecommunications-2000 (IMT-2000) specifications by

the International Telecommunication Union. It is used for transmitting voice and data via UMTS (Universal Mobile Telecommunication System).

[16] Orange, China Mobile, KPN, NTT DoCoMo, Sprint, T-Mobile, Vodafone, Telecom Italia, *"LTE physical layer framework for performance verification"*, St. Louis, MI, USA, February 2007.

[17] Xinyuan Wang, Shiping Chen, Sushil Jajodia, *"Tracking anonymous peer-to-peer VoIP calls on the internet"*, Alexandra, Virginia, USA, November 2005.

[18] Dario Bonfiglio, Marco Mellia, Michela Meo, Nicolò Ritacca, *"Tracking Down Skype Traffic"*, Torino, Italia, 2008.

[19] Chen-Chi Wu, Kuan-Ta Chen, Yu-Chun Chang, Chin-Laung Lei, *"Detecting VoIP Traffic Based on Human Conversation Patterns"*, National Taiwan University, 2008.

List of abbreviations and symbols

3G: Third Generation mobile telecommunications.

AVD: Android Virtual Device.

CDF: Cumulative Distribution Function

DHCP: Dynamic Host Configuration Protocol.

DVM: Dalvik Virtual Machine

GoP: Group Of Pictures.

GUI: Graphical User Interface

IP: Internet Protocol.

JVM: Java Virtual Machine.

Kbytes: Kilobytes.

MTU: Maximum Transfer Unit.

PDF: Probability Density Function.

PMF: Probability Mass Function.

PS: Packet Size.

SDK: Software Development Kit.

TBP: Time Between Packets.

List of figures

Figure 91: Histogram of the response Online Games packet size transmitted, received and ideal in 3G.

Figure 92: Histogram of the response Online Games time between packets transmitted, received and ideal in emulator.

Figure 93: Histogram of the response Online Games time between packets transmitted, received and ideal in WiFi.

Figure 94: Histogram of the response Online Games time between packets transmitted, received and ideal in 3G.

Figure 95: PDF of the uniform part of Skype packet size transmitted, received and ideal in emulator.

Figure 96: PDF of the uniform part of Skype packet size transmitted, received and ideal in WiFi.

Figure 97: PDF of the uniform part of Skype packet size transmitted, received and ideal in 3G.

Figure 98: Histogram of the Skype time between packets transmitted, received and ideal in emulator.

Figure 99: Histogram of the Skype time between packets transmitted, received and ideal in WiFi.

Figure 100: Histogram of the Skype time between packets transmitted, received and ideal in 3G.

List of tables

List of equations

Appendixes

Appendix A

The graphs below show statistical functions of each traffic **treating each connection type separately**, drawing on them three curves: the one in blue corresponding to the transmitted packets, the red one corresponding to the received packets and the ideal curve in black.

Requested Video traffic (packet size):

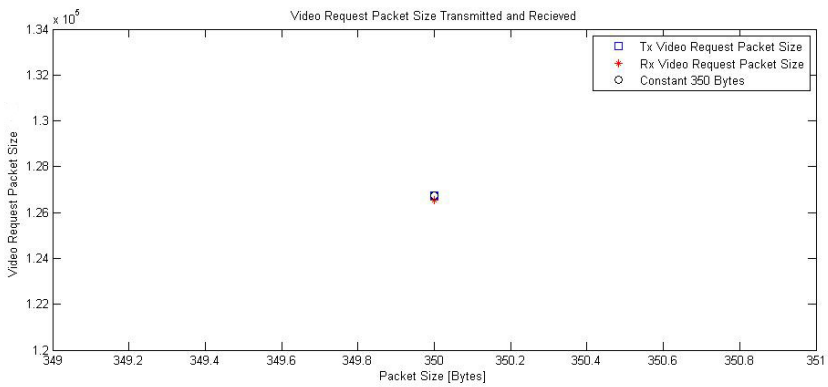

Figure 59: Histogram of the requested Video packet size transmitted, received and ideal in emulator.

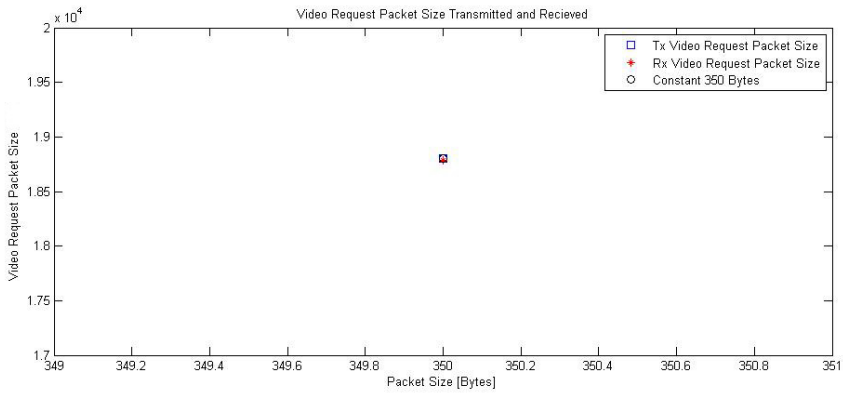

Figure 60: Histogram of the requested Video packet size transmitted, received and ideal in WiFi.

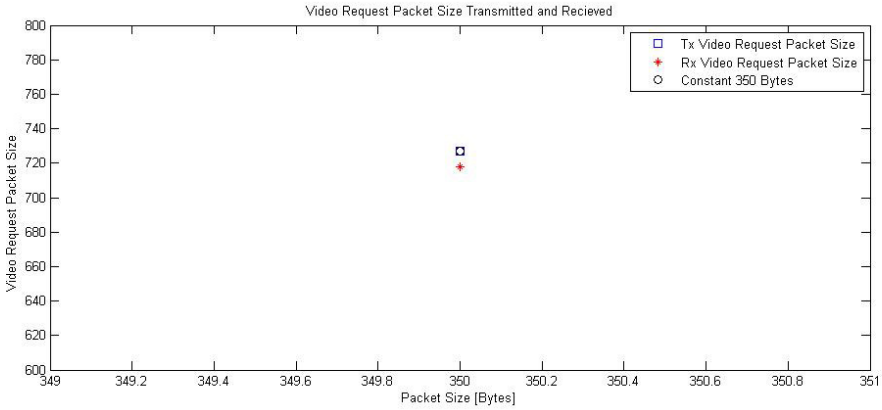

Figure 61: Histogram of the requested Video packet size transmitted, received and ideal in 3G.

Requested Video traffic (time between packets):

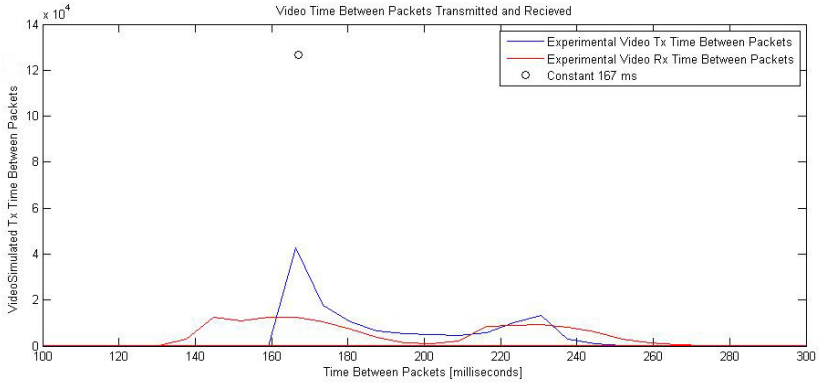

Figure 62: Histogram of the requested Video time between packets transmitted, received and ideal in emulator.

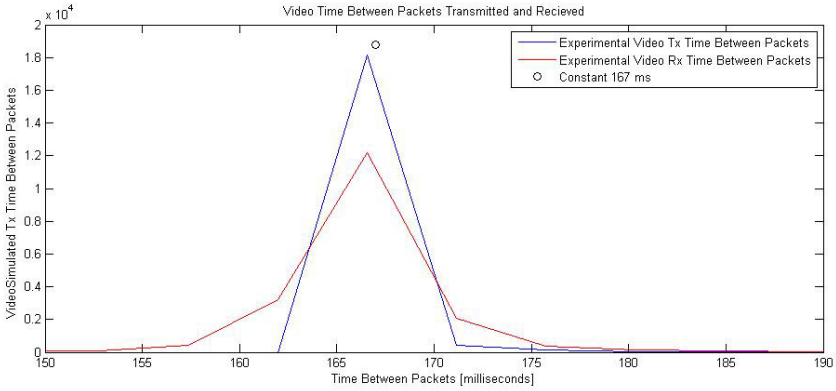

Figure 63: Histogram of the requested Video time between packets transmitted, received and ideal in WiFi.

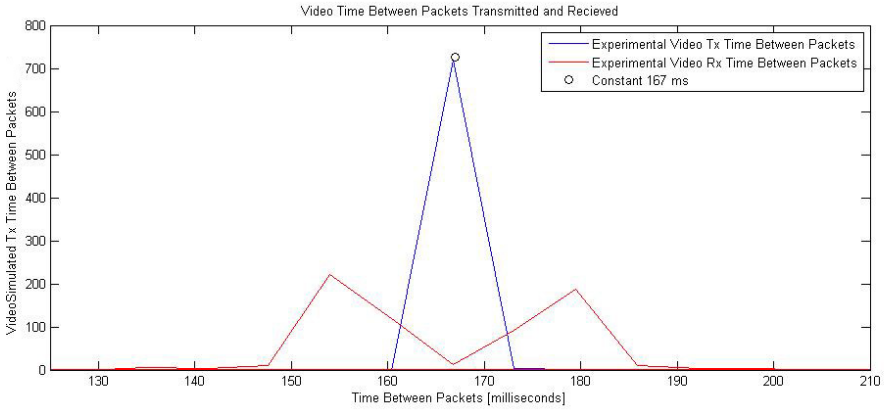

Figure 64: Histogram of the requested Video time between packets transmitted, received and ideal in 3G.

Response Video traffic (packet size):

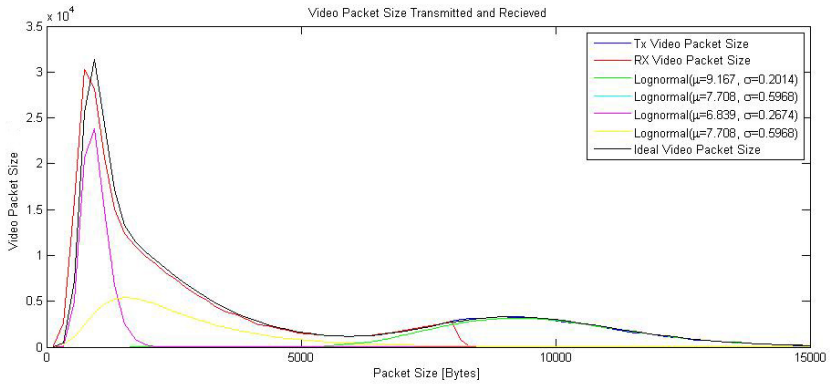

Figure 65: Histogram of the response Video packet size transmitted, received and ideal in emulator.

The colored curves are added to form the final ideal curve, painted black. In this case, the transmitted curves (blue) and received (red) are overlapped, being visible only the red curve.

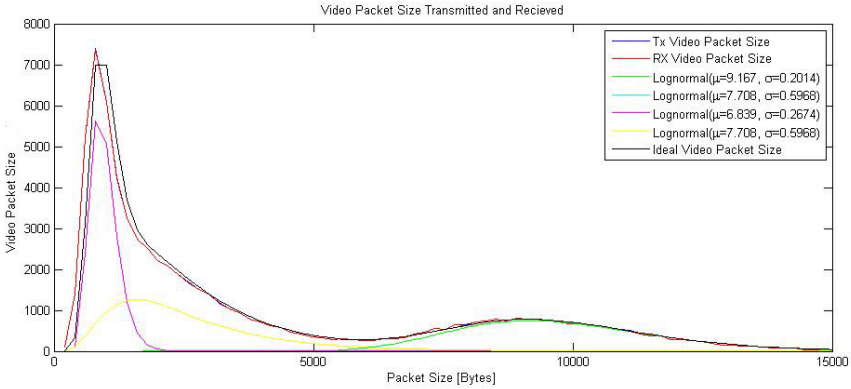

Figure 66: Histogram of the response Video packet size transmitted, received and ideal in WiFi.

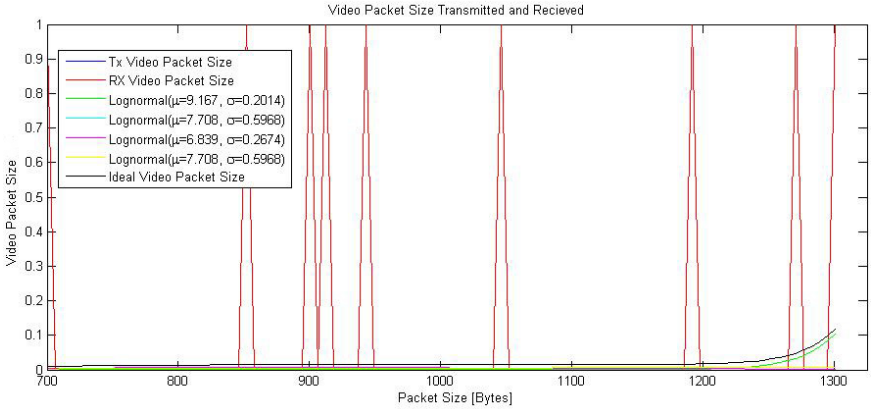

Figure 67: Histogram of the response Video packet size transmitted, received and ideal in 3G.

Due to the low number of successful transmissions in the 3G environment, the response video traffic has so few samples.

Response Video traffic (time between packets):

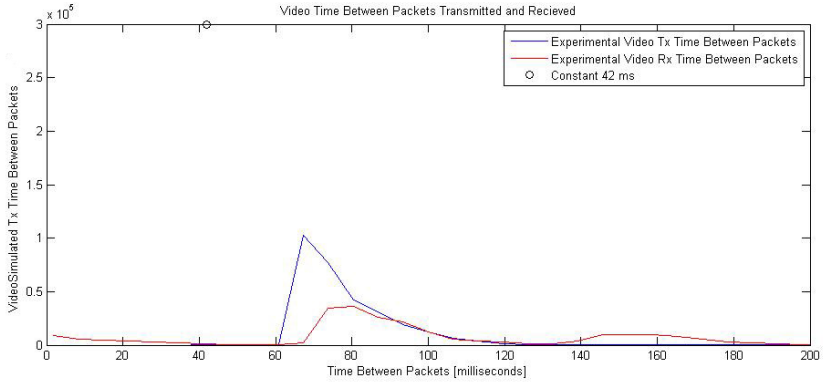

Figure 68: Histogram of the response Video time between packets transmitted, received and ideal in emulator.

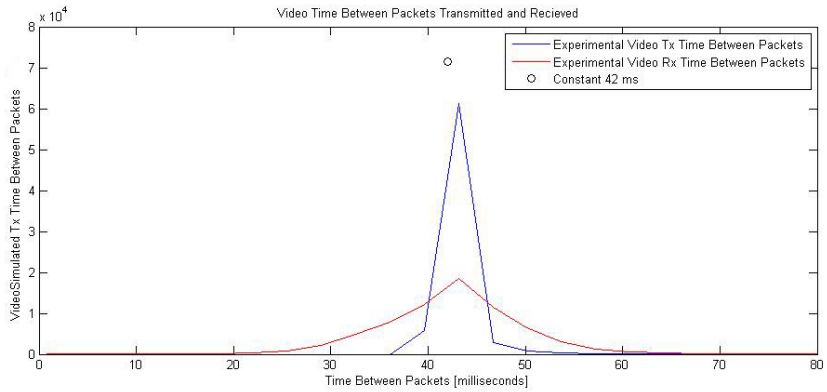

Figure 69: Histogram of the response Video time between packets transmitted, received and ideal in WiFi.

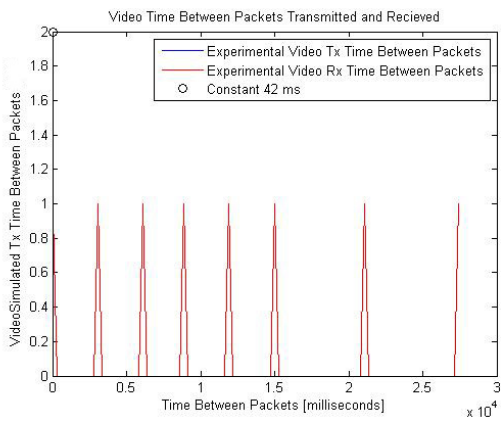

Figure 70: Histogram of the response Video time between packets transmitted, received and ideal in 3G.

Request Web traffic (packet size):

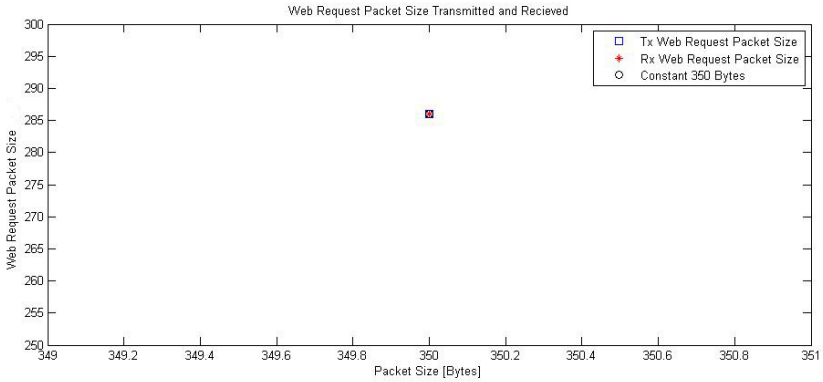

Figure 71: Histogram of the requested Web packet size transmitted, received and ideal in emulator.

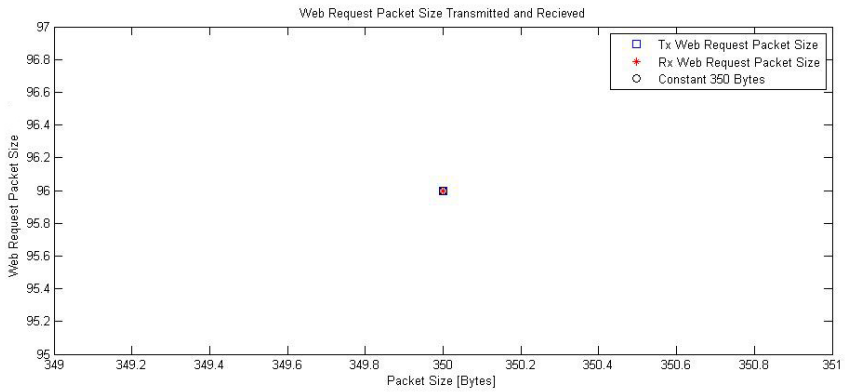

Figure 72: Histogram of the requested Web packet size transmitted, received and ideal in WiFi.

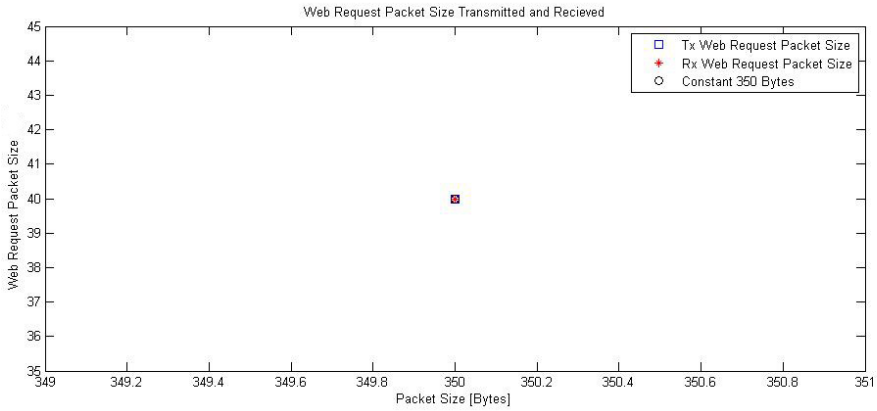

Figure 73: Histogram of the requested Web packet size transmitted, received and ideal in 3G.

Request Web traffic (time between packets):

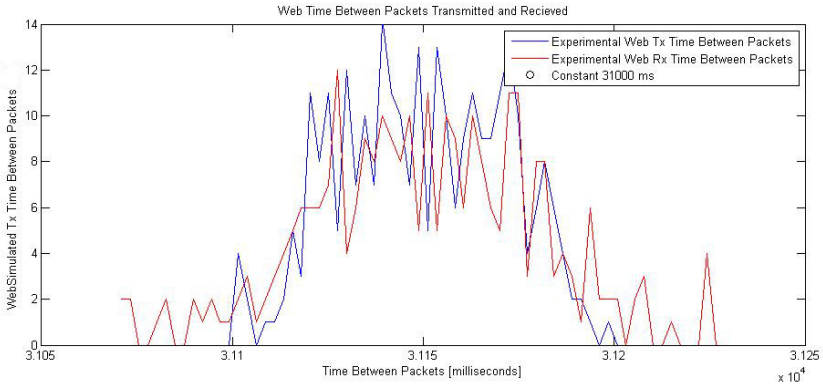

Figure 74: Histogram of the requested Web time between packets transmitted, received and ideal in emulator.

The ideal value is out of the previous figure, concretely in 31.000 milliseconds, but to better appreciate the behavior offered, the zoom has been adjusted such that the curve is as visible as possible.

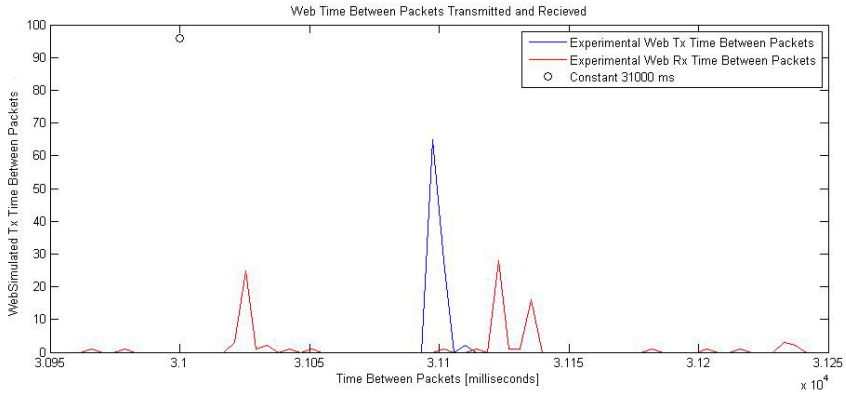

Figure 75: Histogram of the requested Web time between packets transmitted, received and ideal in WiFi.

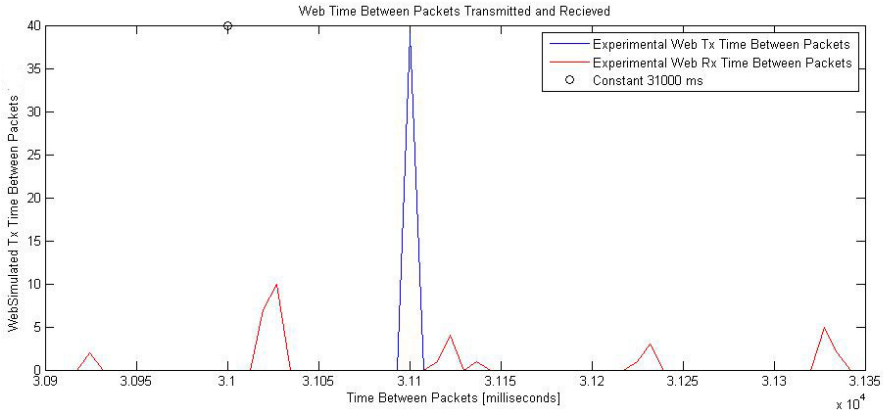

Figure 76: Histogram of the requested Web time between packets transmitted, received and ideal in 3G.

Response Web traffic (packet size):

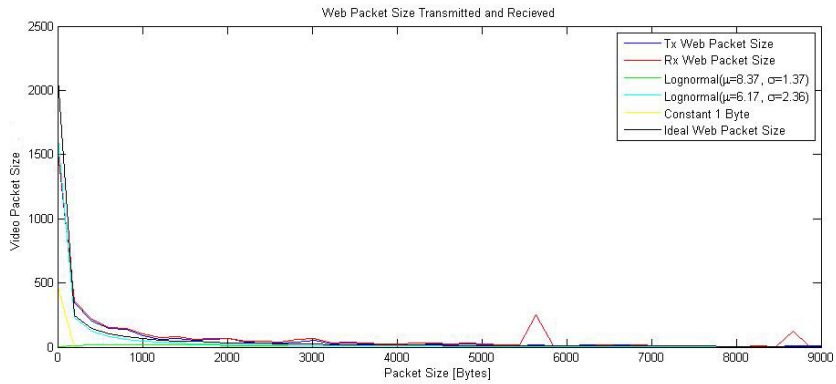

Figure 77: Histogram of the response Web packet size transmitted, received and ideal in emulator.

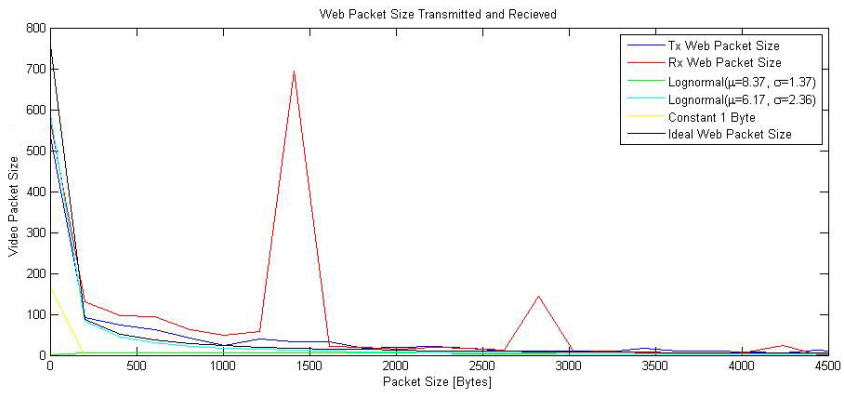

Figure 78: Histogram of the response Web packet size transmitted, received and ideal in WiFi.

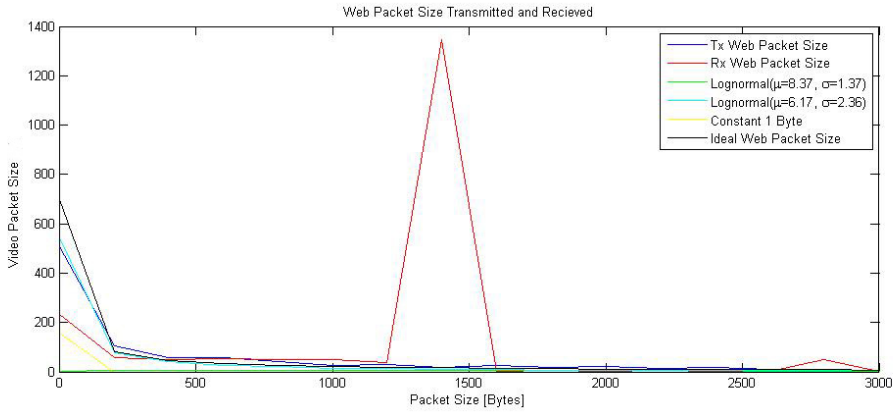

Figure 79: Histogram of the response Web packet size transmitted, received and ideal in 3G.

Response Web traffic (time between packets):

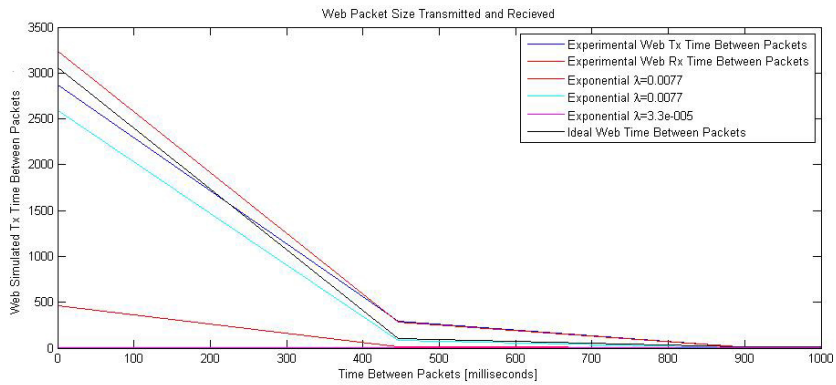

Figure 80: Histogram of the response Web time between packets transmitted, received and ideal in emulator.

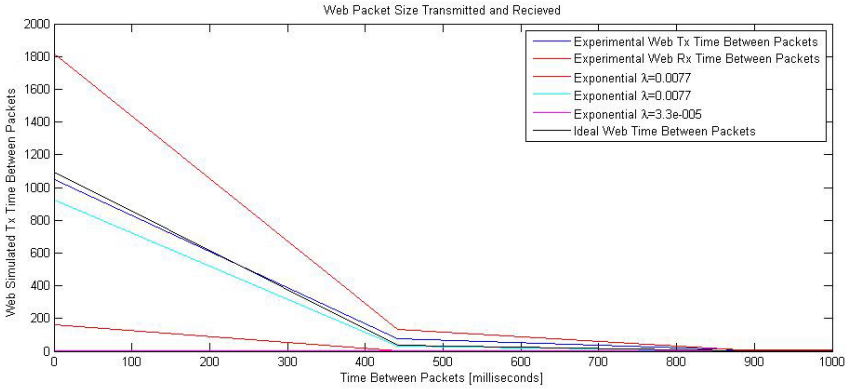

Figure 81: Histogram of the response Web time between packets transmitted, received and ideal in WiFi.

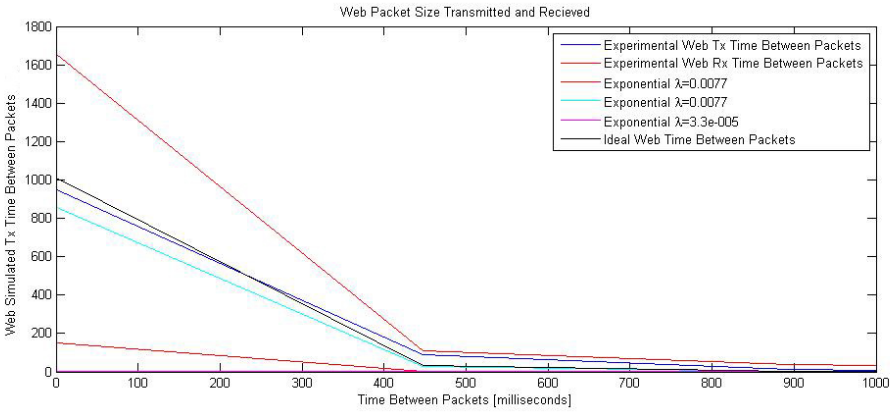

Figure 82: Histogram of the response Web time between packets transmitted, received and ideal in 3G.

Request Online Games traffic (packet size):

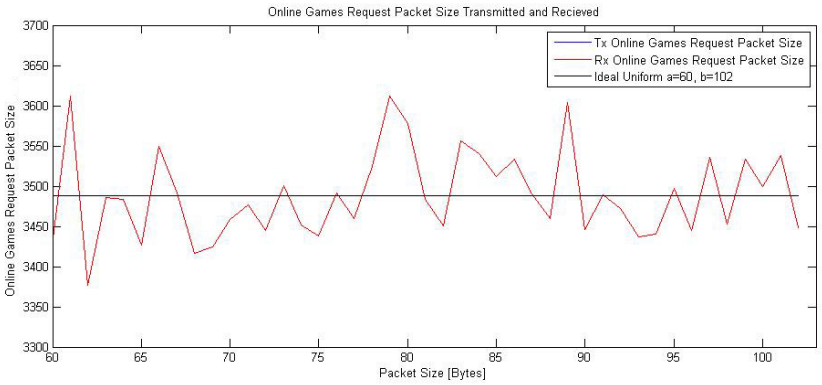

Figure 83: Histogram of the requested Online Games packet size transmitted, received and ideal in emulator.

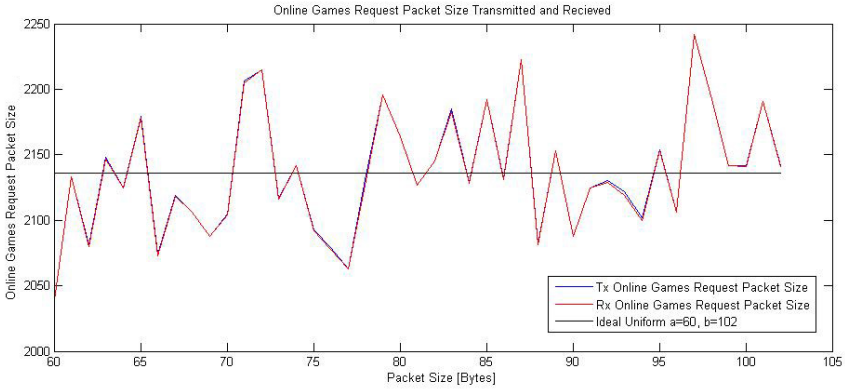

Figure 84: Histogram of the requested Online Games packet size transmitted, received and ideal in WiFi.

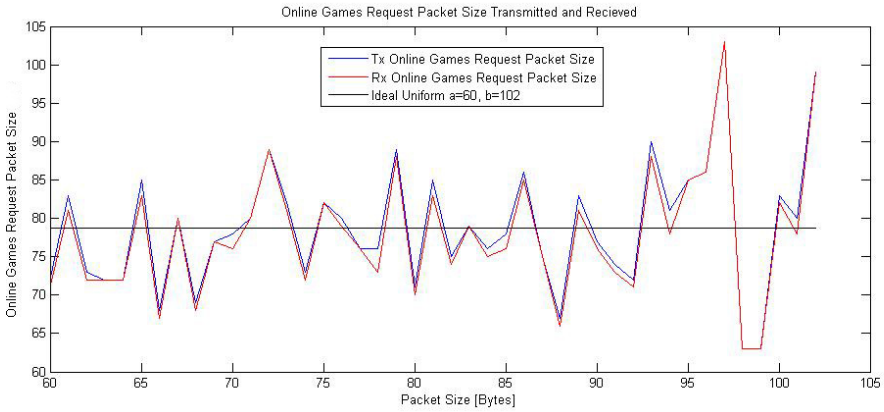

Figure 85: Histogram of the requested Online Games packet size transmitted, received and ideal in 3G.

Request Online Games traffic (time between packets):

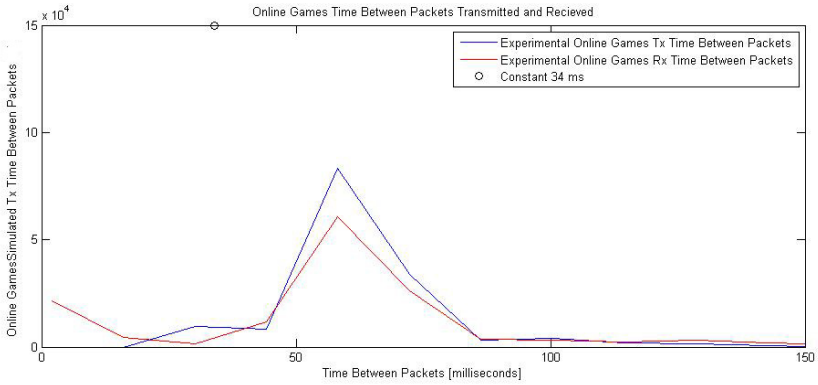

Figure 86: Histogram of the requested Online Games time between packets transmitted, received and ideal in emulator.

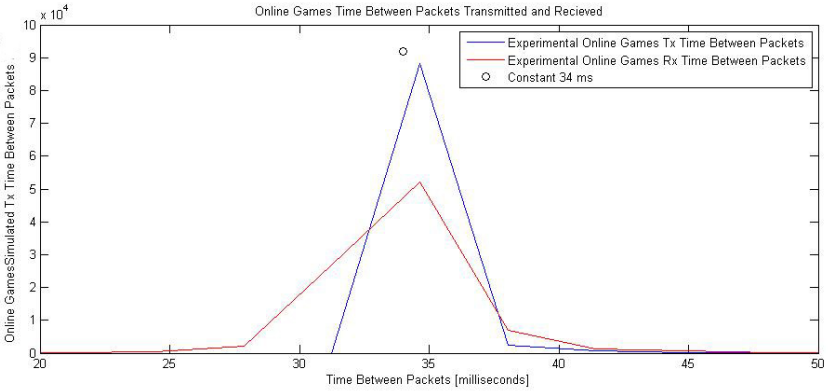

Figure 87: Histogram of the requested Online Games time between packets transmitted, received and ideal in WiFi.

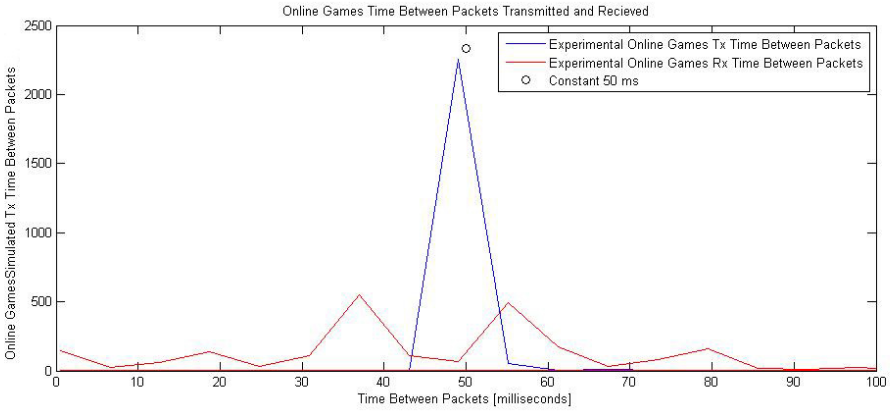

Figure 88: Histogram of the requested Online Games time between packets transmitted, received and ideal in 3G.

Response Online Games traffic (packet size):

In this case it is satisfied that the more extensive the transmissions are (in terms of number of packets sent), the curve obtained fits better with the ideal one.

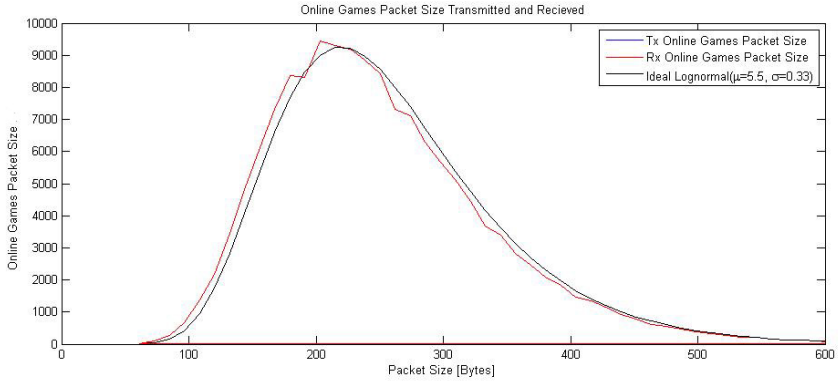

Figure 89: Histogram of the response Online Games packet size transmitted, received and ideal in emulator.

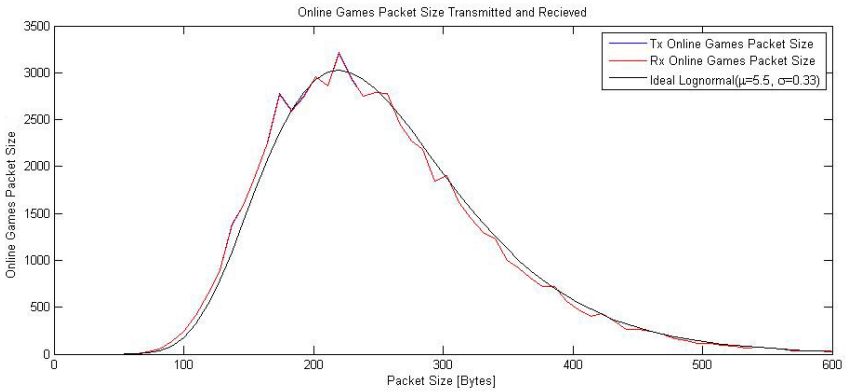

Figure 90: Histogram of the response Online Games packet size transmitted, received and ideal in WiFi.

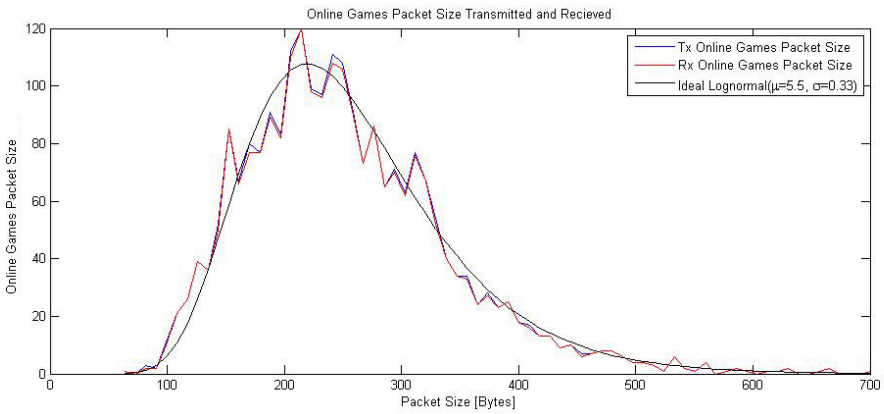

Figure 91: Histogram of the response Online Games packet size transmitted, received and ideal in 3G.

Response Online Games traffic (time between packets):

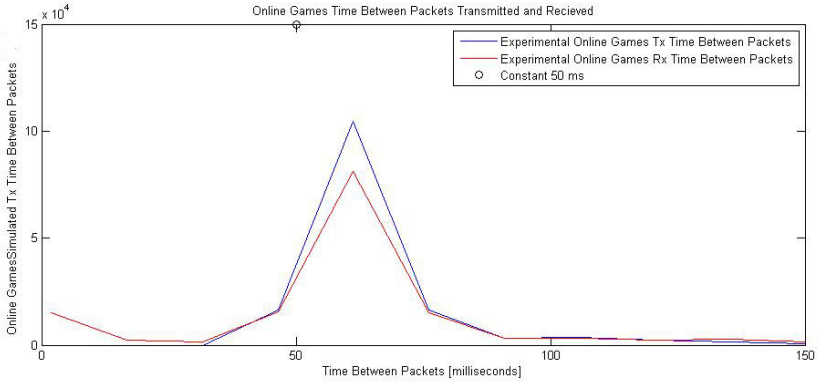

Figure 92: Histogram of the response Online Games time between packets transmitted, received and ideal in emulator.

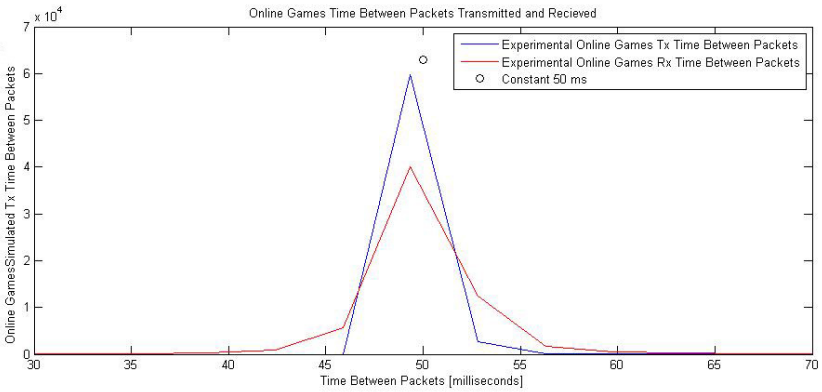

Figure 93: Histogram of the response Online Games time between packets transmitted, received and ideal in WiFi.

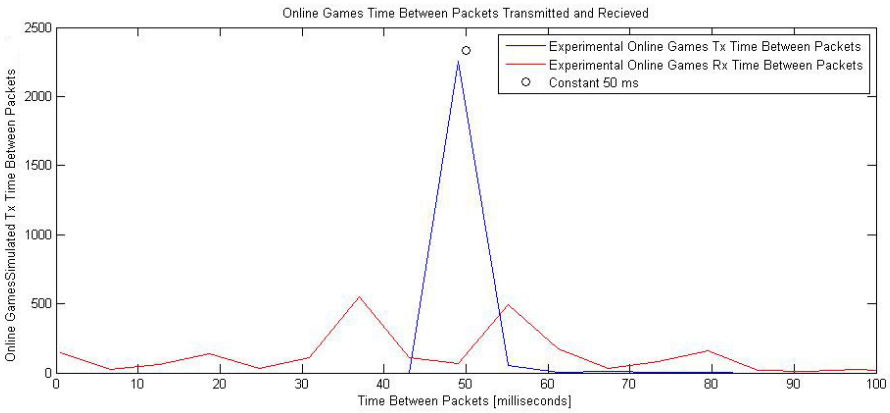

Figure 94: Histogram of the response Online Games time between packets transmitted, received and ideal in 3G.

Skype traffic (packet size):

In the case of Skype's packets sizes, it is shown its probability density function instead of its histogram.

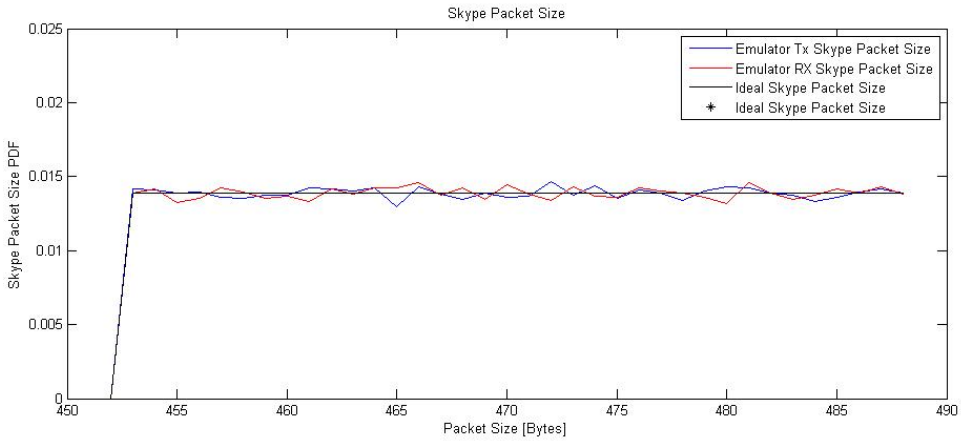

Figure 95: PDF of the uniform part of Skype packet size transmitted, received and ideal in emulator.

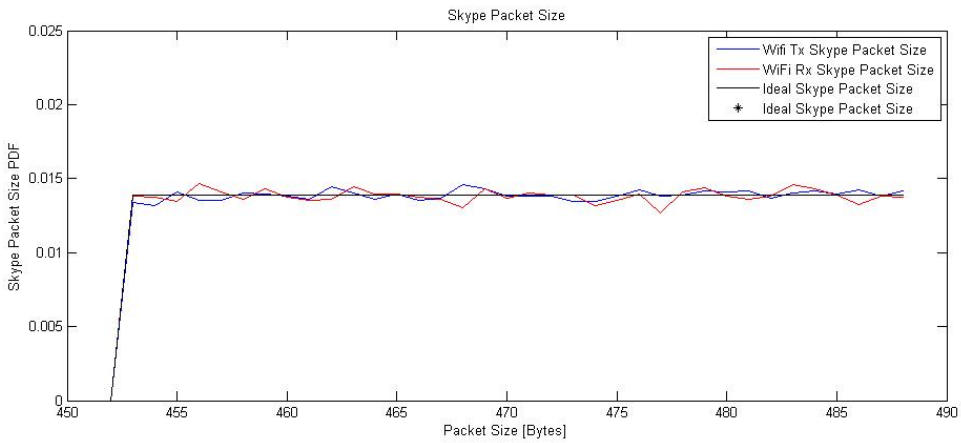

Figure 96: PDF of the uniform part of Skype packet size transmitted, received and ideal in WiFi.

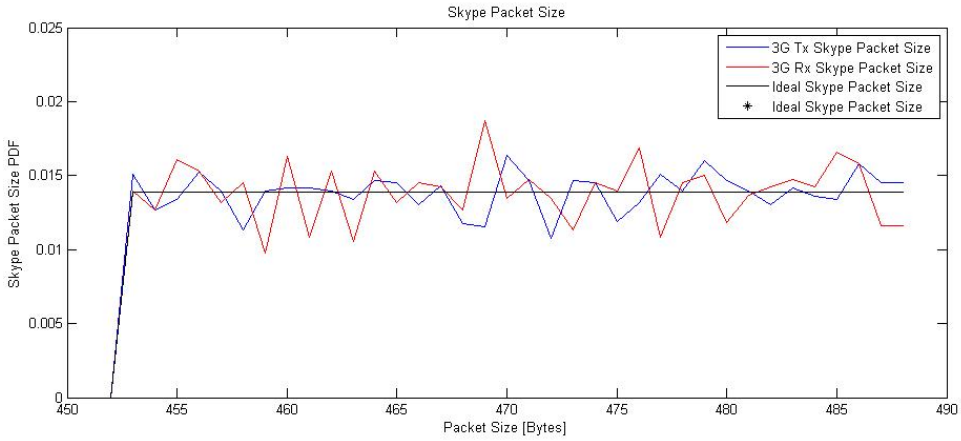

Figure 97: PDF of the uniform part of Skype packet size transmitted, received and ideal in 3G.

Skype traffic (time between packets):

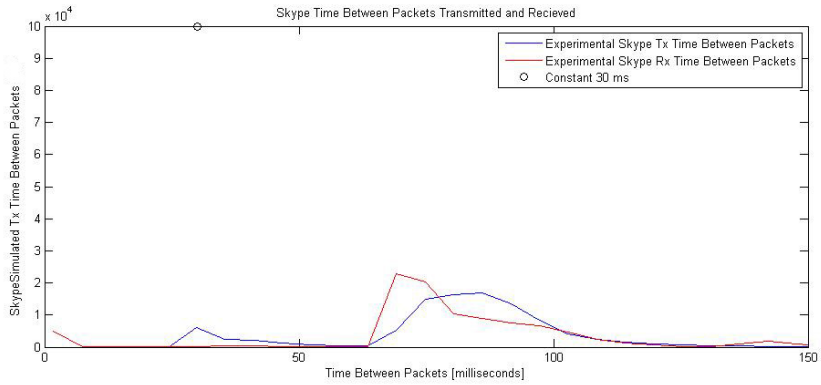

Figure 98: Histogram of the Skype time between packets transmitted, received and ideal in emulator.

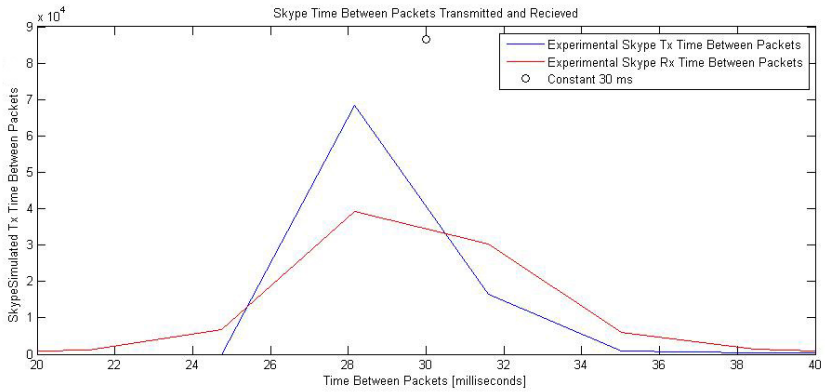

Figure 99: Histogram of the Skype time between packets transmitted, received and ideal in WiFi.

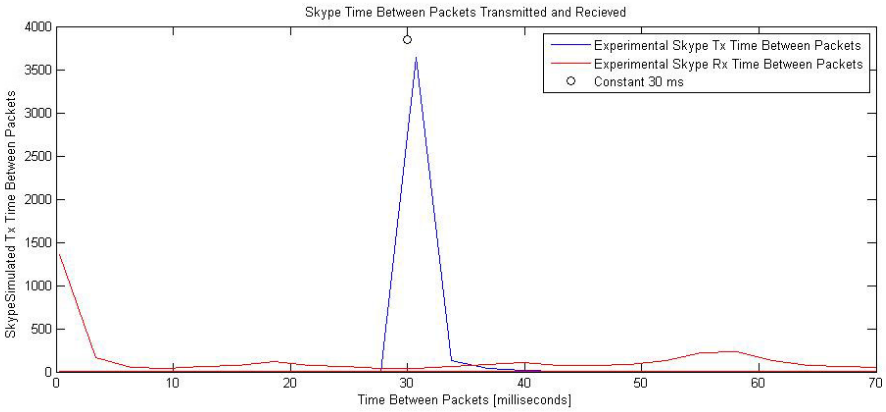

Figure 100: Histogram of the Skype time between packets transmitted, received and ideal in 3G.

Appendix B

Below there is a table with the statistical functions used in the application, as well as their means and variances.

	PDF / PMF	Mean	Variance
Uniform	$fx(X) = \Pr(X = x) = \begin{cases} \frac{1}{b-a+1}, & x \in [a,b] \\ 0, & otherwise \end{cases}$	$\frac{1}{2}(a+b)$	$\frac{1}{12}(b-a)^2$
Constant	$fx(X) = \Pr(X = x) = \begin{cases} 1, & x = ct \\ 0, & otherwise \end{cases}$	ct	0
Lognormal	$fx(x) = \Pr(X = x) = \begin{cases} \frac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}, & x > 0 \\ 0, & x \leq 0 \end{cases}$	$e^{\mu+\sigma^2/2}$	$e^{(\sigma^2-1)(2\mu+\sigma^2)}$
Exponential	$fx(X) = \Pr(X = x) = \begin{cases} \lambda e^{-\lambda x}, & x > 0 \\ 0, & x \leq 0 \end{cases}$	λ^{-1}	λ^{-2}
Geometric	$P(X = x) = (1-p)^{n-1}p$	$\frac{1}{p}$	$\frac{1-p}{p^2}$

Table 16: Statistical functions to be chosen in the traffic generator, including its PDF or PMF and its mean and variance.

Appendix C

The next table shows the statistics of all the types of traffic, put all together for an easier consulting.

	Packet Size	Time Between Packets
Video Request	Constant (350) [12]	Constant (167)
I-Frames (Video Response)	Lognormal (9.167, 0.2014) [11]	Constant (42)
P-Frames (Video Response)	Lognormal (6.839, 0.2674) [11]	Constant (42)
B-Frames (Video Response)	Lognormal (7.708, 0.5968) [11]	Constant (42)
Web Request	Constant (350) [12]	Constant (31000)
Main Object (Web Response)	Lognormal (8.37, 1.37) [12]	Exponential (0.0077) [12]
Embedded Objects (Web Response)	Lognormal (6.17, 2.36) [12]	Exponential (0.0077) [12]
Reading Time (Web Response)	Constant (1) [12]	Exponential (0.000033) [12]
Uplink traffic for Online Games	Uniform (60, 102) [10]	Constant (34) [10]
Downlink traffic for Online Games	Lognormal (5.5, 0.33) [10]	Constant (50) [10]
Skype Audio ON	Constant (40) [16]	Constant (30) [17]
Skype Audio OFF	Constant (1)	Constant (30) [17]
Skype Video	Uniform (453, 488) [18]	Constant (30) [17]

Table 17: Statistics used for the four data traffic types emulated.

Appendix D

Problems encountered while programming the application

During the programming of the application there were various problems that made its implementation more difficult. In order to help a new programmer to solve these problems more easily, here are presented some of the difficulties appeared with their corresponding solutions:

- When starting TCP transmissions, those packets that were closely temporarily separated appeared as one single packet in the receiving side. To fix this, it was added a **minimum temporal separation** between packets of **100 milliseconds**, as it was empirically found out that most of the packets with this interval between them managed to reach the destination separately.
- When the application was successfully tested in the emulator and then loaded onto a real device, it was found out that the application was not working properly, despite working well in the emulator. The problem was that some commands failed and had to be **replaced** by others (more specifically, some constructors had to be changed). For this purpose, *if-else* structures were scheduled in which, depending on the value of a variable, it was **enabled** either the constructors for the emulator or the constructors that worked well for the mobile device.
- When programming an application for Android, it will run on hardware with **limited resources** (mobile devices). This causes that we have certain limitations, as the **maximum number of instructions** that a Java method itself may have. To avoid reaching this limit where Eclipse complains that virtual memory is insufficient, the solution is to collect a set of instructions and include them within a new method, and then call it from the main method; so, the operations to be carried out by the main method will be reduced, because they have been transferred to the new method.
- The Android SDK is updated from time to time, and one of those occasions, after the update, the **directory** where the SDK was located **was invalid**. The folder where the SDK is installed by default is /.android, and the new updated version did not support directories beginning by the character *point*; the solution was to rename the directory to a different name.
- After performing the transmissions, the application generates **text files** which will be later analyzed. When performing transmissions on mobile devices and wanted to **move** the text files **to computer**, the device was connected to the computer but the text files were **not visible** on the

computer, even though they were accessible from the mobile device itself. The solution was to reset the device so that later, when connected to computer, the text files could be normally visible.