Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Emulating 3G Network Characteristics on WiFi Networks

by

Alexander Alesand

LIU-IDA/LITH-EX-A–15/058–SE

2015-09-30

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Final thesis

Emulating 3G Network Characteristics on WiFi Networks

by

Alexander Alesand

LIU-IDA/LITH-EX-A–15/058–SE

2015-09-30

Final thesis

Emulating 3G Network Characteristics on WiFi Networks

by

Alexander Alesand

LIU-IDA/LITH-EX-A–15/058–SE

2015-09-30

Supervisor: Ekhiotz Jon Vergara

Examiner: Simin Nadjm-Tehrani

Abstract

Mobile applications should work regardless of which type of wireless interface is used, and should be able to conceal unstable connections from the user to improve user experience. Therefore, network testing is important when developing mobile applications, but it is a challenge to reproduce network conditions when using real cellular networks since the test engineer has no control over the quality of the cellular network. Existing software tools can restrict bandwidth and add latency to the connection, but these tools do not accurately emulate cellular networks.

This thesis proposes a system where it is possible to shape the network traffic for connected devices to mimic the network patterns of a real cellular connection when running on a WiFi connection. The design presented in this thesis is intended for testing mobile applications under diverse 3G connection parameters, such as latency, bandwidth and other characteristics.

This thesis was conducted at Spotify, a company that provides a music streaming service which is a frequent user of network data traffic. The 3G emulator was evaluated using the Spotify Android application by measuring the correlation between packet traces from a real 3G connection and the 3G emulator. This correlation was compared to the correlation between packet traces from a real 3G connection and the current network emulator at Spotify. The evaluation shows that the proposed 3G emulator outperforms the current network emulator when performing tests on the Spotify application for Android. By using this emulator, we expect the network testing to become more effective as any 3G condition can be tested with repeatable results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When considering the perceived quality of a software from the end user's perspective, not only functional factors are important. One must also consider factors such as performance and energy consumption since these factors make a big impact on the user experience, especially in devices with limited hardware and battery life.

In the case of software which is dependent on wireless networks for communication, testing becomes very complex as different types of networks, including WiFi, 3G, Long-Term Evolution (LTE) etc. behave very differently. This means that *data patterns* (patterns in which data packets are received and transmitted over the network), delays, bandwidth and signal strength changes drastically over time as opposed to wired technologies which keep a comparatively constant quality over time. The network behavior of a mobile application has significant impact on the above mentioned quality metrics. A mobile application must function properly regardless of which wireless technology is used, and should be robust enough to conceal varying or unstable wireless network conditions for the user.

Therefore, testing these adverse conditions is very important, but it is not trivial. The feasibility of testing a mobile application under diverse and realistic network settings is limited by the testing environment where different conditions are generally hard to mimic. The delays in network traffic introduced by different cellular operator settings, variations of signal strength and latencies in the network are a few examples of network conditions that the application tests should consider. Performing tests to cover these cases pose a problem without the correct tools to *emulate* the network conditions.

Emulating is the term we will be using in this thesis to explain the act of mimicking the outwardly observable behavior of a technology. An *emulator* is a device (hardware or software) which performs emulation, which in the case of this thesis means to shape the network traffic.

The vision of the project is that, if a certain 3G network in a far away

place has problems, one can emulate this network on the emulator to reproduce the problems without leaving the office. A WiFi network is much easier to control than a cellular network since the evaluator has control over the WiFi access point. A WiFi network is also more affordable than a cellular network, i.e performing extensive tests on a cellular network requires a dedicated SIM card for each network type (3G, 4G, and different configurations of them). By using a WiFi network and shaping the network data traffic to mimic the data patterns of a cellular connection instead of using a real cellular connection one can use the same network settings each test session, and therefore make it easier to reproduce issues connected to networking.

1.1 Problem Definition

This thesis aims to solve the problem of cellular network emulation over a WiFi network by developing and evaluating a tool which emulates cellular network interfaces by shaping the outgoing and incoming network traffic of a device. Any WiFi compatible device should be able to connect to the emulator, and thereby have an emulated 3G connection.

The main goals of the thesis are:

1. **Design and develop a solution to emulate 3G traffic using a WiFi connection**

   The implemented tool will be carefully designed to mimic a 3G connection as accurately as possible. The implementation will take into account the round trip time (RTT), variances in RTT, bandwidth as well as the radio allocation mechanism of 3G that impacts the data patterns.

2. **Evaluate the data pattern created by the solution against real 3G network traces**

   To ensure that the implemented 3G emulator behaves more accurately than Spotify's previous network testing solution, Snigel which only emulates RTT and bandwidth, an evaluation of the 3G emulator's and Snigel's similarity to a real 3G connection will be conducted. Since the emulator solution is developed at Spotify, the evaluation will be done using the Spotify Android application with several use-cases to generate several different sets of network data.

1.2 Methodology

In the interest of structure, the project is split into several steps. The first three steps correspond to the first goal, and the fourth step corresponds to the second goal. The explanation of each phase is described below:

1. **Study the main mechanisms that impact the traffic patterns in wireless networks.**

   We will survey the recent research literature about 3G and WiFi networks as well as network emulation and performance testing.

2. **Perform an evaluation of the available tools and methods used for network emulation in order to select the most appropriate approach for the project.**

   The evaluation will be done by studying literature concerning relevant tools for general network emulation (Netem, WANem etc.) as well as tools specifically used for 3G emulation. The tools will be compared and considered for usage in the implementation for this thesis.

3. **Design and develop a solution to mimic the dynamic environment of 3G networks.**

   This is the main implementation part of the project. A tool which emulates 3G network interfaces will be developed for the purposes of this project. We will use the knowledge from the literature studies as well as the evaluation of current tools to decide how to build the hardware- and software architecture, as well as to decide which tools to use to develop the 3G emulator. A functional test will be done to verify correct behavior of the implementation.

4. **Evaluate the 3G emulation against real 3G network.**

   The evaluation criteria includes comparisons of RTT, bandwidth and packet inter-arrival time. Network packet traces will be recorded from the mobile device during a Spotify session. The same session will be recorded on a real 3G network, the current network test environment Snigel as well as the newly implemented 3G emulator. The recorded packet traces will then be compared with each other based on packet inter-arrival times to ensure that the new 3G emulator performs more accurately than Snigel. The packet inter-arrival times will make a good metric because it captures the timing characteristics of the network connection.

1.3 Intended Audience

This thesis is intended for people interested in network emulation, network testing, and in particular 3G. The report should be understandable by anyone with basic knowledge of the Linux operating system and knows the difference between kernel level and user level applications. Basic programming knowledge is also necessary as well as basic understanding of computer networks. Knowledge of network programming in Linux and 3G technology is not necessary as everything that is needed will be explained later in this thesis.

1.4 Related Works

This section describes some related previous works and similar projects.

1.4.1 Network Emulation

Network emulation is not a new idea, it is commonly used by developers but it seldom offers any sophisticated emulation features. Some of the functions in these emulators, such as restricting bandwidth and increasing the delay in the network are also desired in this thesis. Therefore, we look at previous implementations of these functions.

Snigel Spotify uses the Snigel system, which is a simple network restrictor developed internally at Spotify. Snigel internally uses Netem which has the possibility to throttle the network by adding delay and restricting the bandwidth.

3G Emulation When it comes to emulating 3G networks, there does not exist many options. There are some solutions developed, for example, López-Benítez et al. [7] and Teyeb et al. [13]. These two papers presents two different solutions to emulate 3G connections in real-time with the aim to evaluate quality of service. Another attempt, very similar to the solution developed for this thesis is presented by Andres Lagar-Cavilla [6]. This is an open source emulator developed for an older Linux kernel 2.6.18 (released in September, 2006). We were not able to find any public evaluations of this tool.

Emulation Comparisons Nussbaum et al. [10] have presented a detailed comparison of three different network emulation tools, where they explain the technical differences and how these translate to functional differences. The main focus of the article is the accuracy of latency and bandwidth emulation. Their tests are valuable for this thesis, as it is used as inspiration for our implementation.

Velásquez and Gamess [15] gives a similar comparison of six different network emulation tools. The accuracy of bandwidth, delay and packet loss is evaluated with both IPv4 and IPv6 traffic.

Bandwidth restriction In 2013, Moe [9] implemented bandwidth restriction to Netem. The implementation is well documented, and the functionality of qdiscs is properly explained.

1.4.2 Mobile Application Testing

3G and WiFi Network Testing Wasserman [18] discusses the issues of mobile application development and mention the problem of testing for

different kind of networks (3G, 4G, WiFi). There is no solution suggested, but the issue is emphasised.

1.4.3 Mobile Energy Consumption

EnergyBox Vergara et al. [17] present the EnergyBox tool, which analyses real network traffic traces and estimates the energy consumed by the wireless interface. EnergyBox models the power consumption by estimating the state in which the 3G interface is, and how this changes over time. The implementation is quite similar to how parts of the implementation in this thesis can be implemented. The difference is that we have to model it in real-time while EnergyBox uses stored packet traces.

1.5 Contributions

The contribution of this thesis is a 3G emulator which allows any WiFi enabled device to connect and have it's network traffic restricted and shaped as if it were connected to a real 3G network. The emulator is intended to be used when testing mobile applications which are heavy users of mobile data traffic.

To achieve this, a Linux kernel module was developed to shape the outgoing network traffic to replicate a 3G connection. This module is based on the tool created by Andres Lagar-Cavilla mentioned above, but improved to properly emulate both uplink and downlink traffic. The kernel module was installed on a server at Spotify and configured to replicate the Telia 3G network in Stockholm, Sweden. An evaluation of the 3G emulator was then conducted by using realistic use-cases from the Spotify client.

1.6 Report Structure

In chapter 1, the problem definition and methodology is explained. Also, related works are described here. Chapter 2 starts by explaining how Spotify uses network testing and the future requirements for their testing environment. Further, we explain the basics about WiFi and 3G networks and data pattern characteristics which is necessary to fully understand this thesis. In chapter 3, all the tools considered to implement in the solution are explained and discussed. Chapter 4 contains the design choices as well as explanation of the hardware and software used in this thesis. In this chapter, the developed 3G emulator software is explained, and configuration details are presented. To emulate a 3G network, one have to measure certain parameters of the 3G network first. These measurements are presented in chapter 5. The evaluation of the 3G emulator is addressed in chapter 6. The results are summarised in chapter 7 as well as a discussion about the feasibility of the proposed solution.

Chapter 2

Background

This chapter explains the necessary background information regarding WiFi, 3G and the current network testing at Spotify. This background information is important to fully understand the implementation and evaluation of the solution proposed in this thesis.

2.1 Network Testing at Spotify

Network testing at Spotify is currently done using the internally developed system Snigel, see Figure 2.1. Snigel consists of a WiFi hotspot connected to the Internet via a Linux computer (shaper) which individually restricts the network for the connected devices. The resulting 3G emulator system developed in this thesis uses the same physical architecture, only the software in the shaper is changed. The network restrictions that can be emulated by Snigel are increased RTT, restricted bandwidth, packet loss, packet duplication, packet corruption and packet reordering.

One of the main upsides with Snigel is the simplicity to connect to it. Any device with a WiFi interface can connect to it just as any other wireless network. The connection is then configured with connection parameters (e.g. bandwidth or latency) individually for each IP address using a Representational State Transfer (REST) application programming interface (API) in the Snigel computer.

The Snigel computer runs a Linux distribution using kernel 3.13, and uses the Linux Traffic Control (TC) framework, an application level program which lets the user control network queuing disciplines (qdisc) in kernel space. TC and qdisc are further explained in section 3.1.1. The Snigel computer contains two wired network interfaces, where the one named eth0 is directly connected by wire to the WiFi bridge, and the interface named eth1 is connected to the Internet. The mobile devices are wirelessly connected to the WiFi bridge.

Due to restrictions in TC and qdiscs, the Snigel computer can only shape

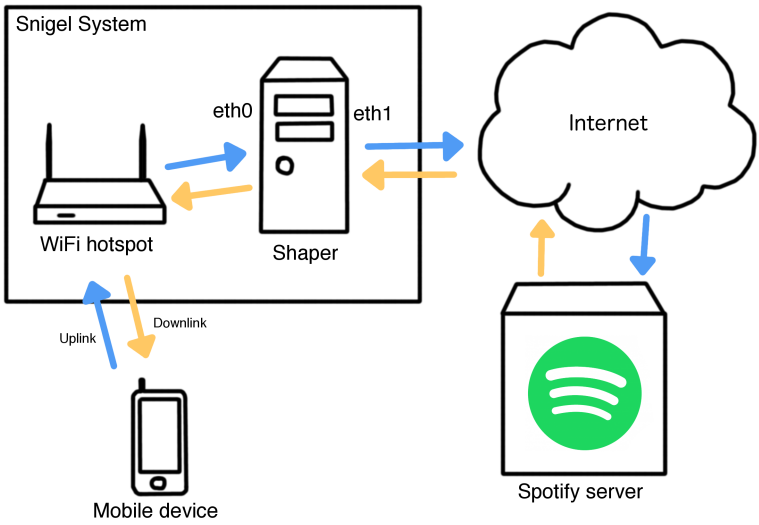

Figure 2.1: Snigel, the current network test setup at Spotify.

the outgoing network traffic. When a data packet enters the Snigel computer on eth0 it is directly bridged to eth1, and vice versa. By shaping the Snigel computers' outgoing data traffic on eth0 according to the downlink rules, and shaping outgoing traffic on eth1 according to the uplink rules, all traffic to and from the device under test (DUT) will be shaped according to the set up rules. The shaping is completely transparent to the devices connected to Snigel and therefore allows any type of device to connect. Any data which source or destination is an IP address which does not have any shaping rule is not shaped.

The shaping in the Snigel system is simply the means of buffering data packets some time before they are sent. Packet loss is emulated by simply not sending the packets. By doing this in certain ways, all of the above mentioned restrictions can be made (increased RTT, restricted bandwidth, packet loss, packet duplication, packet corruption and packet reordering). The time the packets are stored in the Snigel computer before they are sent depends on the configuration of the bandwidth restrictor (Hierarchical Token Bucket) and the latency increaser (Netem). These will be explained in more detail in chapter 3.

The functionalities of Snigel are enough for much of the testing conducted at Spotify, but it does not provide enough functionality to realistically emulate a 3G network. The current approach is able to emulate a bad WiFi connection (increased RTT, reduced bandwidth etc.). However, this does not represent a 3G connection since the network interfaces work differently, and thereby treats the sending and reveicing of data differently. These dif-

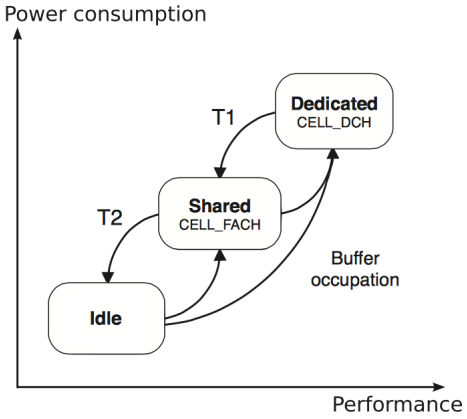

Figure 2.2: The RRC state machine inspired by Ericsson [19].

ferences are covered in section 2.2 and 2.3.

2.2 3G Characteristics

When considering performance, 3G differs from WiFi in a number of ways. One of the most important aspect is of course the bandwidth and RTT, but we also have to consider more 3G specific aspects, such as the radio allocation performed at the operator which introduces additional delays and performance differences. In this chapter, we explain and study these aspects to be able to imitate them when considering the implementation of the 3G emulator.

2.2.1 Radio Resource Control (RRC) Protocol

One of the most important aspects of 3G, which differs a lot from WiFi, is the state machine implemented by the mobile device and controlled by the cellular operator. The state machine is shown in Figure 2.2. The states are described below.

RRC States: The three states which the 3G interface can be in are idle, Forward Access Channel (FACH) and Dedicated Channel (DCH) from the least energy consuming to the most energy consuming. In the DCH state, the user equipment (UE) is allocated a dedicated channel for communication both uplink and downlink. This provides the highest possible bandwidth and lowest RTT, but also results in the highest energy consumption at the UE. In the FACH state, the UE monitors the downlink channel for incoming packets, and is provided a shared uplink channel which can be used to

State radio power	
Idle	200 mW
FACH	500 mW
DCH	1300 mW
Inactivity timer	
DCH to FACH (T_1)	4.1 sec
FACH to idle (T_2)	5.6 sec
Transition duration	
Idle to FACH	0.43 sec
Idle to DCH	1.7 sec
FACH to DCH	0.65 sec
DCH to FACH	0.7 sec
FACH to idle	0.3 sec
RLC Buffer threshold	
Idle to DCH (Uplink)	1000 bytes
Idle to DCH (Downlink)	515 bytes
FACH to DCH (Uplink)	294 bytes
FACH to DCH (Downlink)	515 bytes

Table 2.1: Example of RRC states observed on an Ericsson F337 broadband module and TeliaSonera 3G network at one location in Sweden, as reported by Vergara et al. [17].

send small packets. The idle state does not provide any means of transmission, but continuously monitors the downlink channel to see if there are any incoming packets.

The states provide different performance which varies between network operators, UE and signal strength. For example, Perälä et al. [11] observes a network where DCH provides 14.4 Mbit/s downlink bandwidth and 173 $\pm$ 11 ms RTT and FACH provides 4 kbit/s downlink bandwidth and an RTT of around 578 $\pm$ 3.8 ms.

State transitions: Since the idle state does not allow any data to be sent or received, when the device in idle wants to send or receive data, the device will be told to switch to the FACH state. To be allowed to switch from the FACH state to DCH, the device will need to fill the send or receive buffer, which varies in size depending on the network operator's settings. The chance that a state promotion happens increases as the buffer allocation gets closer to the threshold. A downswitch on the other hand, happens when the device has been in low or no activity during a certain amount of time. We call these timers the inactivity timers. The values of the timers are decided by the network operator.

When the RRC decides to transition the 3G interface to a higher or a lower state, a delay (transition duration) occurs before the device enters the

new state since signaling occurs between the mobile device and the operator to set up the new state. During this time, no data at all can be sent or received. We expect this to have a negative impact on network performance, and it will therefore be considered when implementing the 3G emulator.

Since the 3G interface uses much more power in DCH than in FACH or idle, a long DCH to FACH tail will force the device to stay in the DCH state even when the data usage is very small, or none at all. This will impact the energy consumption a lot. If the network operator has the parameter configured in the other extreme, a very short DCH to FACH tail, the device would constantly switch between DCH and FACH and therefore cause unnecessary state switches which negatively impact the network performance. It is the network operator's responsibility to configure these settings for optimal balance between performance and energy consumption [12].

The buffer thresholds, inactivity timers and transition durations differ between network operators, for example, the parameters of a network measured by Vergara et al. [17] are presented in Table 2.1, where the inactivity timers are in the range of 4.1 to 5.6 seconds, transition durations in the range of 0.3 to 1.7 seconds, and buffer thresholds in the range of 294 to 1000 bytes. Also, Qian et al. [12] measure a network with inactivity timers in the range of 5 to 12 seconds, transition durations in the range of 1.5 to 2 seconds and buffer thresholds of 543 ± 25 to 475 ± 23 byte.

2.3 WiFi Characteristics

The WiFi network is much simpler than the 3G network, it does not have any forced energy saving features, and we should therefore be able to emulate the RRC states on a WiFi network. However, emulating other networks over WiFi has its limitations since the emulation will inherit the data pattern characteristics of the WiFi interface. To know the operational range we are given by the WiFi network we have studied the WiFi technology and presented the findings in this section. Previous works [20] have shown that power saving mode can impact the latency of the WiFi network, and thus we disable all power saving features in the WiFi switch.

An interesting aspect of WiFi is that it uses CSMA (Carrier Sense Multiple Access). Carrier Sense means that the transmitter that wants to send data first tries to detect the presence of a current sender, and if a sender is detected the transmitter waits for a random amount of time before trying again. This means that only one transmitter can send at a time. The effect of this is that the bandwidth and latency quickly deteriorates as more users connect to the network [8]. This also applies on nearby networks which operate on the same channel. Because of this, if the best possible emulation is desired one has to isolate the testing environment from all other WiFi networks, and make sure to only test one device at a time. This as done in this project by selecting the least busy 5 GHz band to conduct all tests.

Chapter 3

Tool Selection

To implement the 3G emulator, we first need to find the adequate tools to use. In this chapter, we examine the most interesting tools in this field, and decide on which to use for the implementation of the 3G emulator.

3.1 Network Shaping Tools

There exists a number of tools that are interesting for the purposes of this thesis. This section will cover the most notable tools and explain the functionality and mention use-cases for all of them.

3.1.1 Traffic Control

Traffic Control (TC) [4], is the user level application which controls the Linux kernel's network scheduling. TC allows the user to apply different types of queuing disciplines (qdisc) and filters to a network interface.

Queuing Disciplines

In the context of networking, a qdisc decides how network traffic should be sent on the network. The default setting on any network interface in a Linux environment is simply to use a FIFO queue, which sends the first available packet out to the network.

A Linux system can use the standard FIFO queue, or use more complex qdisc modules which could for example delay packets, or randomly drop packets before they are sent. Qdiscs can be combined and applied in sequence to obtain effects like a traffic shaper.

The qdiscs are applied between the data link layer and the network layer (level 2 and 3) in the OSI model [14]. As such, all network traffic sent is captured by the qdisc. Each qdisc has two important virtual functions that must be implemented: enqueuing packets to be sent and dequeuing packets

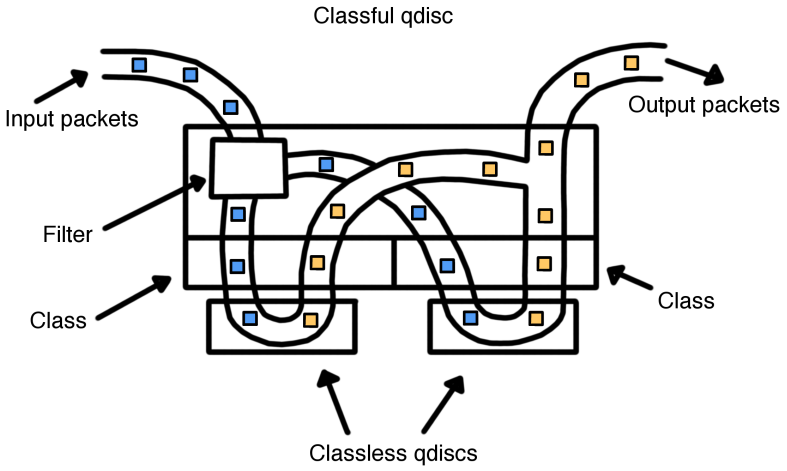

Figure 3.1: Visual explanation of the qdisc and filter system of TC.

for sending away to the network interface. There are a wide selection of qdiscs available in Linux[1,2], the ones used in this thesis are explained later in this chapter. Qdiscs are categorised in classless and classful qdiscs. Figure 3.1 shows a visual representation of classful and classless qdiscs.

Classless Qdisc: Classless qdiscs are those that can only delay or drop packets. They can not divide data into different categories, and can therefore only shape the entire network interface. A classless qdisc (FIFO, Token Bucket Filter etc.) is the simplest type of qdisc and can be used as the primary qdisc on a network interface, or as a child to a classful qdisc. A classless qdisc cannot have an additional qdisc attached to it (child qdisc). Therefore a classless qdisc will always be the leaf in a qdisc tree.

Classful Qdisc: Classful qdiscs on the other hand may divide the data into different categories, and are therefore much more versatile. A classful qdisc (Netem, Hierarchical Token Bucket etc.) can have child qdiscs, (classful or classless) attached to it to make combinations of qdiscs. Some classful qdiscs can even have several children, and divide the data traffic between the children qdiscs. To decide which data traffic should go to which child, filters are attached to the classful qdisc.

[1] http://tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html
[2] http://tldp.org/HOWTO/Traffic-Control-HOWTO/classful-qdiscs.html

Filters

Filters are applied to classful qdiscs to classify the data traffic. For example, filters can let packets from a certain IP address go through one set of qdiscs and packets from all other IP addresses go through another set of qdiscs. One of the most common filters is the u32 filter. This filter allows to filter traffic based on source and destination, port number etc. The filters let us shape different kinds of traffic with different qdiscs. Figure 3.1 contains a visual representation of qdiscs and filters.

3.1.2 Netem

Netem is one of the most well known qdiscs used for emulation. Since Linux kernel 2.6 Netem is delivered as part of the TC system [3]. The most important feature of Netem is the possibility to delay packets before they are sent, which is used as a way to emulate network connections with high round trip time. Apart from delaying traffic, Netem also provides functionality for emulating packet loss, packet duplication, packet corruption, packet reordering as well as throughput restriction.

Recently [9], bandwidth restriction was implemented into Netem by calculating the time-to-send for each packet considering both the configured latency and bandwidth when a packet enters Netem. This means that it is no longer dependent on other qdiscs to restrict the bandwidth. Netem is currently used for RTT emulation in the Snigel system at Spotify.

3.1.3 Hierarchical Token Bucket

The Hierarchical Token Bucket (HTB) qdisc is used to restrict the bandwidth of several traffic flows at the same time. With the use of filters, one can direct different data traffic to different qdiscs attached to the HTB. One could, for example, restrict the bandwidth to a certain port or while allowing full bandwidth to another port. The HTB is used to restrict the bandwidth on the Snigel testing network at Spotify.

The HTB restricts bandwidth by allowing packets to be sent only if there are enough tokens available. One token might, for example, represent one byte. If we want to send a packet of 1200 bytes, the HTB will wait until 1200 tokens are available, and then send the 1200 bytes large packet. When sending, the tokens that were needed to send are consumed, which means that we in this case end up with 0 tokens left. The tokens are replenished at a rate which is decided by the configured bandwidth [14].

As a result of the HTB algorithm, the maximum amount of tokens which can be held will decide the amount of burstiness that is allowed. If the maximum amount of tokens is 10000, then 10000 bytes can be sent in a quick succession even if that means that the configured bandwidth is temporarily exceeded. However, the average bandwidth over time will not exceed the configured bandwidth.

3.1.4 UMTS RRC Emulation Module

There exists a qdisc developed by Andres Lagar-Cavilla for AT&T released under the GPL v2 [6], which emulates the delays caused by transitions in the RRC state machine in 3G hardware as explained in chapter 2. The tool is implemented as a qdisc for Linux kernel 2.6.18 and a patch to the TC command line tool. The qdisc allows the user to specify many RRC parameters, such as RRC state transition times, buffer limits for state transitions, rate limits, inactivity timers etc.

Apart from the RRC state machine, the UMTS RRC Emulation Module (RRC-EM) works much like Netem, as it allows for configuration of RTT, packet loss and bandwidth limitation. One limitation of this RRC-EM is that the uplink and downlink emulations are completely separated. This means that if a RRC state transition happens for the downlink emulation, the same transition might not happen in the uplink emulation. This is not a good enough emulation for our purposes. We want to always have the same state for uplink and downlink emulation to realistically emulate a 3G network.

The reason for this flaw is not very hard to understand. Since qdiscs only apply to packets we are sending, we need to have two instances of the emulator running to emulate both uplink and downlink. In the case of this RRC-EM, two instances have no means of communicating with each other, and therefore act as separated emulations.

The bandwidth restriction in the RRC-EM works by calculating the time-to-send in the same way as in Netem. Upon inspection of the UMTS RRC-EM and the Linux kernel revision history, it is clear that much has changed in the Linux kernel between version 2.6 and the kernel the current Snigel is using, 3.13. Thus, a complete rewrite would be necessary if we are to use this kernel module for the purposes of this thesis.

3.1.5 WANem

TC can be complicated to configure with all command line options and filters. Engineers at the Performance Engineering Research Centre at TATA Consultancy Services[3] tried to solve this by developing WANem. Everything needed to emulate networks with a Linux distribution and providing an easy-to-use web interface for configuration is included in WANem. WANem makes use of TC as well as Netem and HTB to achieve the network emulation [5].

WANem is an open source project built on a re-mastered Knoppix distribution [5], and could therefore be further modified for the purposes of this thesis. This would provide a nice graphical user interface which can be used by Spotify, but it would also require more work to integrate the missing parts in a quite large system.

[3] http://wanem.sourceforge.net

3.1.6 Dummynet

Dummynet contains mostly the same functionality as TC/Netem, but has some differences in the implementation. Firstly, Dummynet was originally developed for FreeBSD, but has since been ported to Linux, OS X and Windows [2]. Dummynet is an open source project as well, so it can be modified for this thesis. One of the most notable differences compared to TC/Netem is that Dummynet allows the user to configure both uplink and downlink shaping on a single network interface. Dummynet contains a similar implementation of bandwidth limitation as Netem. If used with high resolution timers, this implementation prevents the bursty traffic that the token bucket could cause [10]. Instead, this method results in a very even packet inter-arrival-time without any bursts.

3.2 Tool Selection

The tools described above are all capable of performing some parts of the requirements of this thesis. Table 3.1 summarises all tools and their capabilities. It is clear, however, that none of the tools provide everything that we need to achieve the desired 3G emulation. It is therefore clear that we have to develop a new tool, based on one of the emulation tools we have studied.

There are functional differences in the tools as well. According to Nussbaum and Richard [10], there are differences in the RTT and bandwidth emulation in these tools. There is a problem when using too low resolution timers, that the emulated RTT is not accurate. TC can achieve considerably better results by using High Resolution Timers.

Since we want to keep our new 3G emulator structurally as close as possible to the current Snigel system, Dummynet is not an option since it does not use the TC system, and will therefore not be compatible with the current Snigel system.

WANem is a large project containing fancy configuration tools and graphical user interface. For the size of this thesis, a rewrite of WANem to emulate 3G networks would require too much work.

HTB contains quite little functionality and does not provide anything which is not already provided by Netem. Therefore the choice is between implementing RRC state emulation in Netem, or converting RRC-EM to Linux kernel 3.13 as well as implementing the uplink-downlink state communication. Since packet duplication, packet corruption and packet reordering is not used at Spotify, these functionalities do not impact the decision.

The most interesting aspect to evaluate in the 3G emulation tool is the RRC emulation. Therefore, this is the most important feature in the tool selection. The advantage of the RRC-EM is that it is the only option providing RRC state emulation. However, as previously mentioned, the RRC-EM has flaws in the implementation that would lead to inaccurate state switching.

Minimal features	Netem 1.3	HTB	RRC-EM	WANem	Dummynet
Open source	Yes	Yes	Yes	Yes	Yes
Bandwidth	No	Yes	Yes	Yes	Yes
Round trip time	Yes	No	Yes	Yes	Yes
Packet loss	Yes	No	Yes	Yes	Yes
RRC emulation	No	No	Yes*	No	No
For kernel 3.13	Yes	Yes	No	Yes	Yes
Extra features	Netem 1.3	HTB	RRC-EM	WANem	Dummynet
Packet duplication	Yes	No	No	Yes	No
Packet corruption	Yes	No	No	Yes	No
Packet reordering	Yes	No	No	Yes	No

Table 3.1: Comparison between three tools for network emulation.
* With limitations, RRC-EM separates uplink and downlink RRC emulation. RRC states can be different for uplink and downlink network traffic.

This is fixed by implementing shared data between the uplink and downlink qdiscs.

Chapter 4

Emulator Design and Implementation

In this chapter, we will present the solution we propose for the 3G emulator. The design choices are explained, both in terms of hardware and in terms of software. Emulator installation and configuration are also explained in this chapter.

4.1 Placement of 3G Emulator

The emulator placement in the network chain is an important design choice. It decides which tools will be available, and which parts of the 3G network that we can emulate.

For this thesis, two options for the emulator placement were considered. Either we place the emulator on a separate server which all network traffic will be bridged through, or we place the emulator within the Device Under Test (DUT) itself. Table 4.1 summarises the advantages and disadvantages of both approaches which are described in the next paragraphs.

The applications running in the Android operating system are aware of

	In phone	In server
Platform independent	No	Yes
Easy setup in a new phone	No	Yes
Can be used with any device	No	Yes
Emulation on any WiFi network	Yes	No
Phone behaves as when connected to 3G	Yes	No

Table 4.1: Comparison between performing the 3G emulation in the phone or performing it in an external server.

which type of connection that is used (3G or WiFi). By placing the emulator in an external server, we will not be able to trick the Android applications to believe that they are connected to a 3G network. By placing the emulator in the device itself instead we gain some additional control as we can tell the applications that the phone is connected to a 3G network instead of the WiFi network that the DUT actually is connected to. Placing the emulator in the device itself would also make it possible to test the device on any network, and we would not be tied to a certain test location.

However, by having the emulator in the device, the emulator would have to be developed for every type of device we want to test (Android, iOS etc.). At Spotify, both Android and iOS apps are developed, therefore it is preferable if both platforms can use the 3G Emulator. This approach will also make it necessary to tamper with the device itself, which is not desirable as it might affect test results and make it harder to test new devices.

If the emulation takes place in an external server, we will be tied to a certain WiFi network, but we do not need to make any modifications to the devices we want to perform tests on, and we can test all WiFi compatible devices. By having the emulator behave in this plug-and-play manner, we make it easier to perform tests on the 3G emulator. However, we lose the knowledge of which network the device is connected to, which can mislead intelligent applications that use the knowledge of which connection they are using. Even with this downside, we decide to perform the emulation on a separate server because the DUT remains untouched and it is easier to test new devices.

4.2 Emulator Architecture

This section will describe the general architecture of the 3G emulator. In the last section, we decided to place the 3G emulator on a separate server, connected in one end to the WiFi bridge and in the other end connected to the Internet. All traffic will then be bridged through the 3G emulator. The tools we are using are TC for the front end controller for the kernel-level 3G emulator we have developed as a qdisc.

The 3G emulator qdisc is inspired by the work of Andres Lagar-Cavilla on the RRC Emulation Module for AT&T [6]. The new 3G emulator, however is developed for a newer Linux kernel version (3.13), and provides proper RRC state emulation, as the RRC state will always be the same for uplink and downlink traffic.

One of the most important parts of the 3G emulator is the software implementation as a qdisc controlled by the TC command line tool. It is important to keep in mind that the qdisc may only shape outgoing network traffic. Incoming traffic to the network interface the qdisc is attached to remains untouched and unaffected.

However, we want to shape both incoming traffic and outgoing traffic. The way we solve this is to set up the 3G emulator as a network bridge,

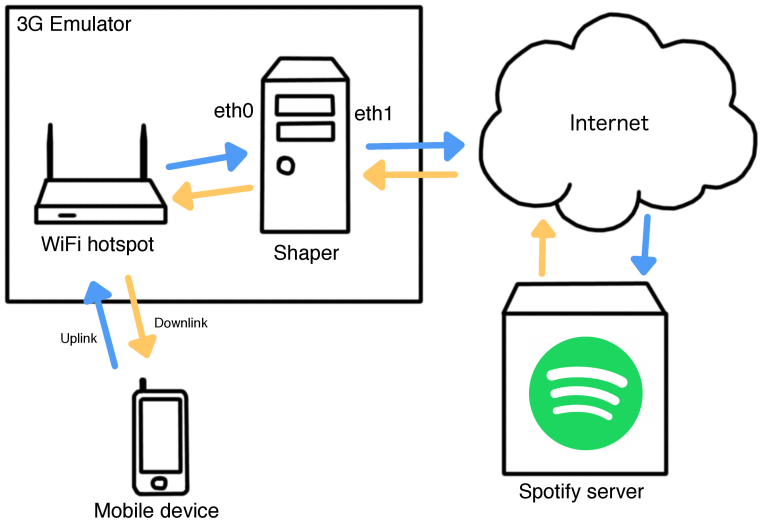

Figure 4.1: The proposed 3G emulator.

according to Figure 4.1. We apply two instances of the 3G emulator qdisc, one applied to the network interface eth0 to shape downlink traffic, and one applied to eth1 to shape uplink traffic. The two instances are configured independently to reflect the different network properties on uplink and downlink data traffic. The RRC states, however, are shared between the two qdiscs since we want uplink and downlink emulation to always be in the same state.

4.2.1 Hardware Environment

The WiFi bridge used in this thesis is a ZyXEL NWA1123-NI, operating in the 5GHz frequency band. Every power saving feature is disabled on the WiFi bridge, to get maximum performance. The WiFi bridge is connected by Ethernet cable to the eth0 port on the computer running the emulator.

The 3G emulator runs on a Mac Mini (Late 2012) with Intel Core i5-3210M running at 2.5 GHz and using 4 GB RAM. The Mac mini has Ubuntu 14.04 server edition installed. The Mac mini has two Ethernet ports, one called eth0 and one called eth1. The WiFi bridge is connected to the Ethernet port eth0, and the eth1 port is connected to the Internet via a router.

For the 3G emulator to handle both upstream and downstream networking, the computer hosting the 3G emulator requires at least two physical network interfaces. One interface (eth1) will be used for uplink shaping and the other network interface (eth0) will be used for shaping downlink network traffic. The Mac Mini is configured as a bridge which simply forwards

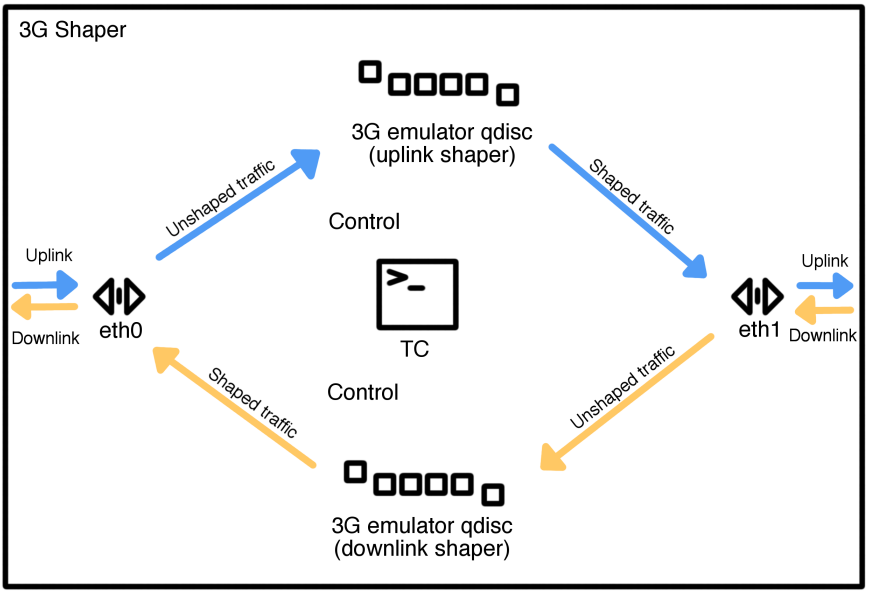

Figure 4.2: Software architecture overview.

all traffic from the interface which shapes downstream (eth0) traffic to the interface which shapes upstream (eth1) and vice versa. Since the qdiscs only shape outgoing traffic, only the outgoing traffic of the 3G emulator will be shaped.

4.3 Kernel 3G Emulator

The 3G emulator software consists of two parts, TC and the 3G emulator qdisc. TC is the front end that serves with communication to and configuration of the emulation qdisc. TC is part of the Iproute2 tool suite for Linux, which was modified for this project to provide the communication with the new 3G emulation qdisc.

Figure 4.2 shows the software content of the 3G shaper from Figure 4.1, and depicts the data flow within the 3G shaper. The uplink data traffic enters the emulator on the eth0 network interface, and is shaped before leaving the emulator on the eth1. Downlink traffic follows the reverse route. There are two running instances of the 3G Emulator qdisc, one which shapes uplink traffic and one which shapes downlink traffic. TC is used to setup the two qdiscs as well as to modify their parameters at runtime.

Emulation Module

As mentioned above, the 3G emulator software consists of the 3G emulation qdisc and the patch to the TC command line tool. This section will explain the 3G emulation qdiscs.

When TC is used to setup a new 3G emulation qdisc, TC first sanity checks the configuration, and then calls the 3G emulation qdisc to setup the emulator. The 3G emulation qdisc now maps the settings entered into TC to the internal variables in the qdisc. The qdisc is now ready to handle the network queue.

The qdisc contains many functions for queuing, such as functions for dropping packets, changing settings during runtime, destructors and functions for logging etc. The most important functions, however are the functions which handle enqueuing and dequeuing. These functions are explained in detail below.

Packet Enqueuing When a packet is going to be sent over the network interface which has the 3G emulator qdisc attached, the packet is sent to the function rrc_enqueue where the scheduling of packets occurs. The enqueuing function also handles much of the RRC state machine, the function consists of eight parts explained below:

1. Fetch the state and state tail timers from the shared variables.
2. Check the inactivity timers and downswitch the state if an inactivity timer has expired. See Figure A.1 for details.
3. Next, the upswitch state transition checks occur. The emulator always upswitch when it is in the idle state. Which state the emulator upswitches to depends on whether the buffer exceeds the threshold or not. See Figure A.2 for details.
4. The time at which the packet should be sent (time_to_send) is calculated based on the current time, the delay in the current RRC state and an extra delay in case a state transition is currently in progress. Also, a random jitter is added. See Figure A.3 for details.
5. Update the T_1 and T_2 (dch_end and fach_end) inactivity timers. See Figure A.3 for details.
6. Remove transition flags if the transition has ended, see Figure A.4 for details.
7. The shared variables are updated with the new state and the tail times as previously updated.
8. Enqueue or drop the packet depending if the backlog limit is exceeded. See Figure A.4 for details.

Packet Dequeuing When the network interface is ready to send, the function rrc_dequeue is called to dequeue a packet from the send queue. Similarly to the enqueuing function explained above, rrc_dequeue can be explained in three phases.

1. Fetch the state from the shared variables.
2. The token bucket algorithm updates the amount of tokens available. This is done by multiplying the time elapsed since we last sent a packet with the configured bandwidth. Since the tokens is only used in the dequeuing step 3, we only need to update the amount of tokens at this point.
3. The third phase takes a peek on the packet to be sent, and checks if it is time to send it yet. If it is time to send the packet, the function checks if there are enough tokens to sent. Then the packet is either sent or requeued if there are not enough tokens available. See Figure A.5 for details.

4.4 3G Emulator Setup

This section instructs the installation and usage of the 3G emulator.

Bridge Setup

The computer running the 3G emulator should be set up as a network bridge, with the uplink and downlink interfaces bridged. This will allow both uplink and downlink network traffic to be shaped when the 3G emulation qdisc is applied to both interfaces. The bridge can be set up using the brctl[1] application:

```
sudo ifconfig eth0 0.0.0.0
sudo ifconfig eth1 0.0.0.0
sudo brctl addbr br0
sudo brctl addif br0 eth0
sudo brctl addif br0 eth1
sudo ifconfig br0 up
```

The first and second lines clear the IP addresses of the network interfaces eth0 and eth1. The third line creates a new network bridge, and the fourth and fifth lines adds eth0 and eth1 to the bridge. The last line activates the newly created bridge. The 3G emulator is now forwarding all traffic, and is completely transparent to devices on both sides of the emulator.

[1] http://www.tldp.org/HOWTO/BRIDGE-STP-HOWTO/set-up-the-bridge.html

3G Emulator Instructions

As mentioned earlier, the 3G emulator needs to run two instances, one for uplink shaping and one for downlink shaping. An example of setting up the 3G emulator using the TC command line interface can be seen below:

```
sudo tc qdisc add dev eth0 root rrc dchulrate 15Mbps backloglimit
    65525 tokenlimit 65535 idlepromomu 1500000 fachdchmu 1500000
    rlcul 294

sudo tc qdisc add dev eth1 root rrc dchulrate 2Mbps backloglimit
    65525 tokenlimit 65535 idlepromomu 1500000 fachdchmu 1500000
    rlcul 294
```

Setting up the 3G emulator is quite straightforward regarding delay, state transition tail, state transition delays and buffer thresholds. The bandwidth emulation is a bit trickier to set up. By setting the dchulrate, you set a maximum throughput. The backlog limit and token limit might have to be increased to reach higher bandwidths, and lowered when emulating lower bandwidths to avoid excessive delays introduced by the token bucket algorithm.

The 3G emulation can be changed during runtime to provide more dynamic situations. This is done by the "qdisc change" command, with the same syntax as the "qdisc add" command. To change the download speed to 22 Mbit/s, one types:

```
sudo tc qdisc change dev eth0 root rrc dchulrate 22Mbps backloglimit
     65525 tokenlimit 65535 idlepromomu 1500000 fachdchmu 1500000
    rlcul 294
```

The 3G emulator can be disabled by deleting the qdisc, just as one deletes any qdisc using TC:

```
sudo tc qdisc del dev eth0 root
```

This will delete all qdiscs and filters on the eth0 interface.

Chapter 5

Emulator Parameter Collection

In this chapter, a real 3G network is measured to find the parameters necessary to emulate it with the 3G emulator. The impact of the WiFi network is also measured. To make accurate evaluations, we need to estimate the 3G network parameters, which are later used to configure the 3G emulator. By measuring the 3G network parameters as accurately as possible, we will get more accurate results in the evaluation. The step of estimating 3G network parameters will also be an important step for the end-user of the 3G emulator since this will be important to accurately emulate a 3G network.

5.1 Measuring 3G Network Parameters

In the evaluation, we want the parameters of the 3G emulator to reflect the real 3G network we want to emulate. Therefore, prior to evaluation, the 3G network parameters are estimated by measurements. The estimated parameters are presented in Table 5.1, and we explain how we obtain the parameters in this chapter.

5.1.1 Measuring Inactivity Timers

Both inactivity timers are measured by sending ping packets with varying delay to see which intervals cause a transition back to the previous state, and which do not. The methods used in this section are inspired from the methods presented by Vergara [16].

Inactivity Timer T_1

The T_1 inactivity timer denotes the time the device must be inactive until a demotion from DCH to the FACH state occurs. To measure T_1, we send 600

Inactivity timer	**Measured Value**
DCH to FACH	3.6 - 3.8 sec
FACH to idle	4.9 - 5.6 sec
Transition time	
Idle to DCH	999 ms
FACH to DCH	617 ms
Idle to FACH	515 ms
DCH to FACH	981 ms
RLC Buffer threshold	
Idle to FACH	Always
Idle to DCH (Uplink & Downlink)	219 - 223 B
FACH to DCH (Uplink & Downlink)	222 - 261 B
Downlink Bandwidth	
FACH	3.45 kbit/s
DCH	21.9 Mbit/s
Uplink Bandwidth	
FACH	3.45 kbit/s
DCH	3.20 Mbit/s

Table 5.1: Measured parameters for the Telia 3G network in Stockholm, Sweden.

large ping packets (1300 B) with a delay varying between 0 and 10 seconds between the packets while recording the packets with Tcpdump[1]. We send large packets to make sure to trigger a transition to DCH each time we send or receive a packet. The RTT is measured by each ping packet, and by analyzing this RTT we can determine if a state promotion takes place or not. The RTT is measured with the ping tool, which sends a packet to a given destination. The computer which receives the ping packet sends an answer back to the sender. A ping measures the time between the packet is sent and the answer is received. This time is called RTT. This is the method we use when measuring RTT in this thesis.

Figure 5.1 shows the results of the test, where we can see that there is a clear separation between packets causing a transition and packets not causing a transition. We can see that when the interval between ping packets is large, the resulting RTT is also large, which means that a state transition from FACH to DCH occurred. On the other hand, when the interval between packets is small, the resulting RTT is also small, which means that the RRC state machine is still in DCH, and did not need to make a transition.

In Figure 5.1, we can see that in the delay interval 3.6 to 5.2 second, we have a significantly higher RTT than the rest of the delays which caused a transition. This is because the RRC has just started transitioning from DCH to FACH when the next packet arrives. This leads to the RRC state

[1] http://www.tcpdump.org

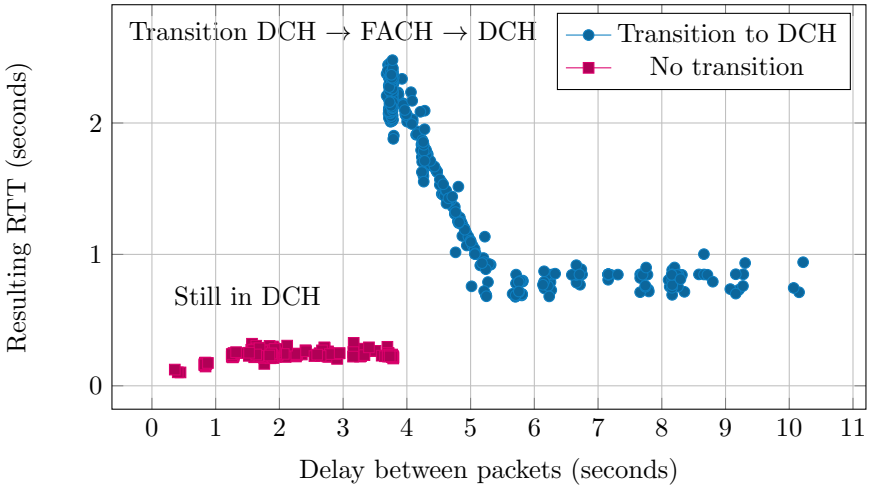

Figure 5.1: Estimation of inactivity timer T_1.

machine transitioning down to FACH then up again to DCH before the packet can be sent.

To get the lower bound of T_1, we find the minimum ping interval that caused a transition, which is 3.6 s and to find the upper bound, we find the maximum ping interval that did not cause a transition, which is 3.8 s. Therefore, a reasonable estimate for T_1 is obtained: 3.6 s $< T_1 <$ 3.8 s.

Inactivity Timer T_2

The T_2 inactivity timer denotes the time the device must be inactive for a transition from FACH to idle to occur. The estimation of the T_2 timer is quite similar to the estimation of the T_1 inactivity timer. However, this time we send 600 small ping packets (24 B) with varying intervals between 1 and 10 seconds, in order to not trigger a DCH transition, but instead trigger an idle to FACH transition. Similar to the T_1 test, we can see that when the interval between ping packets is large, the resulting RTT is also large, and vice versa. A small resulting RTT means that a transition to FACH did not occur, since we are already in FACH, and a large RTT means that a transition from idle to FACH occurred. Figure 5.2 shows the results of the test. A similar pattern as in Figure 5.1 can be seen here as well. This occurs between the delay interval 5 to 6 seconds, where a similar state transition pattern occurs, this time from FACH to idle and back to FACH.

To get the lower bound of T_2, we find the minimum ping interval that caused a transition, which is 4.9 s and to find the upper bound, we find the maximum ping interval that did not cause a transition, which is 5.6 s. Thus, we estimate T_2 as: 4.9 s $< T_2 <$ 5.6 s.

The reason that there is a relatively large range between 4.9 and 5.6

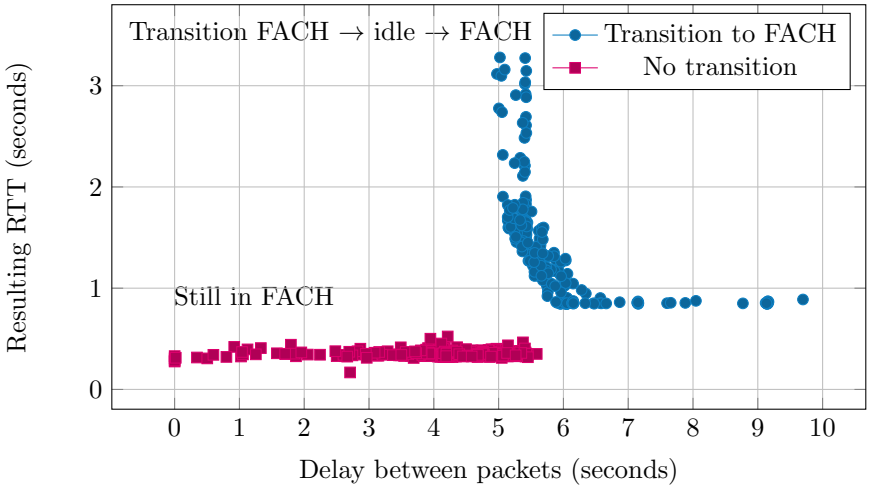

Figure 5.2: Estimations of inactivity timer T_2.

seconds could imply that there are errors in the estimation, or that the actual T_2 is not an exact number, but actually a variable number which can change within the range. The fact that the T_1 range is not as wide (3.6 to 3.8 seconds) suggests that the estimations are correct, and we conjecture that the inactivity timers can are set by the operator dynamically.

5.1.2 Measuring Buffer Thresholds

Buffers thresholds are tested by repeatedly sending packets of known size to find the instant at which a state transition will happen at each packet size.

Buffer Threshold B_1

The buffer threshold B_1 denotes the threshold for the RRC to transition from idle to DCH. To test this threshold, we first send a packet of a known size, followed by two seconds delay, then we measure the RTT. The RRC state machine will trigger a transition to FACH in cases when the amount of data sent is lower than the threshold, and triggers a transition to DCH if the amount sent is higher or equal to the threshold. A high RTT implies we are in the FACH state, and a lower RTT implies we are in the DCH state. Between each packet sent, the device will have to rest until the idle state is entered. On the network and device we used in this test, based on the sum of the inactivity timers, 20 seconds was enough rest time for the device to enter the idle state each time.

The test repeated 500 times on packets sizes in the range of 210 to 230 B, the results can be seen in Figure 5.3. The idle to DCH buffer threshold is within the range of 219 to 223 B, where a transition is more likely to

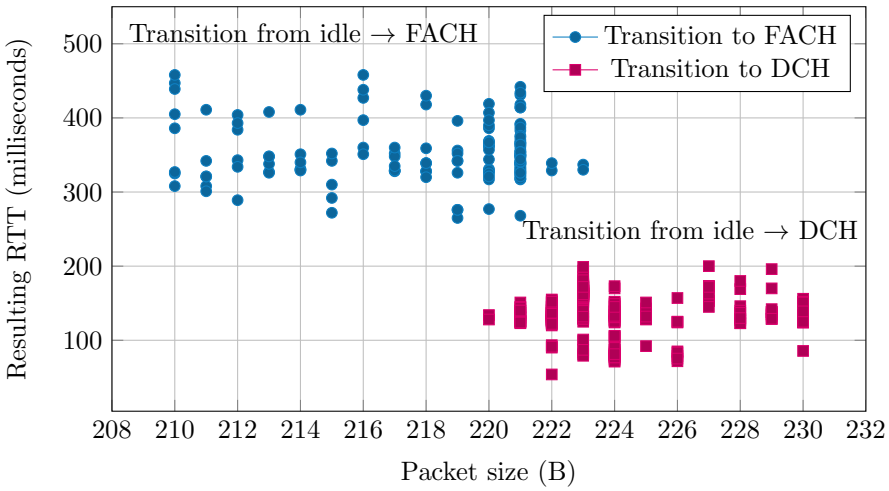

Figure 5.3: Estimation of the idle to DCH buffer threshold.

be triggered when sending larger packets, which is shown in Figure 5.4. Thereby, we estimate B_1 to: 219 B $\leq B_1 \leq$ 223 B.

Buffer Threshold B_2

The buffer threshold B_2 denotes the buffer threshold for the RRC to transition from FACH to DCH. The estimation of this threshold is quite similar to the previous threshold. During the entire test, we send small ping packets every 2 second to keep the RRC state machine in FACH state, but not trigger a transition to DCH. Every 10 seconds, we send a packet of known size to trigger a transition from FACH to DCH. The RTT is measured two seconds after the packet is sent to determine which state the device is in. As in the previous test, a high RTT indicates FACH and a low RTT indicates DCH. Figure 5.5 shows the size of the packets and the chance this size has to trigger a FACH to DCH state transition. The result is that the FACH to DCH buffer threshold is estimated as: 222 B $\leq B_2 \leq$ 261 B.

5.1.3 Transition Durations

Transition Duration $T_{Idle-DCH}$ When measuring this delay, we send large packets (1300 B) to trigger the DCH transition by exceeding the idle to DCH buffer threshold. The packet is send every 20 seconds to give the RRC state machine time to be demoted down to idle before the next packet is sent.

To account for the RTT in the DCH state, we measure the RTT in the DCH state by sending the 1300 B ping packet 100 times when the device is in the DCH state. The mean RTT in DCH is calculated to 181 ms, and the

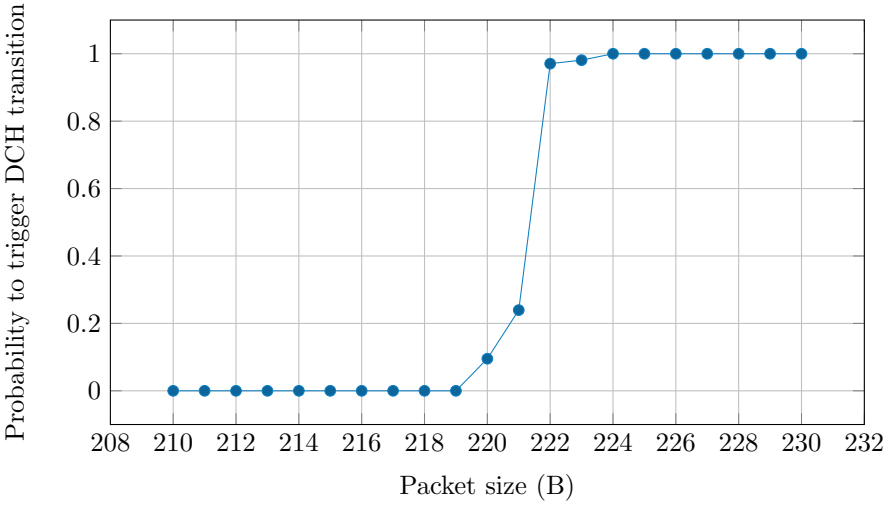

Figure 5.4: Probability to transition to from idle to DCH.

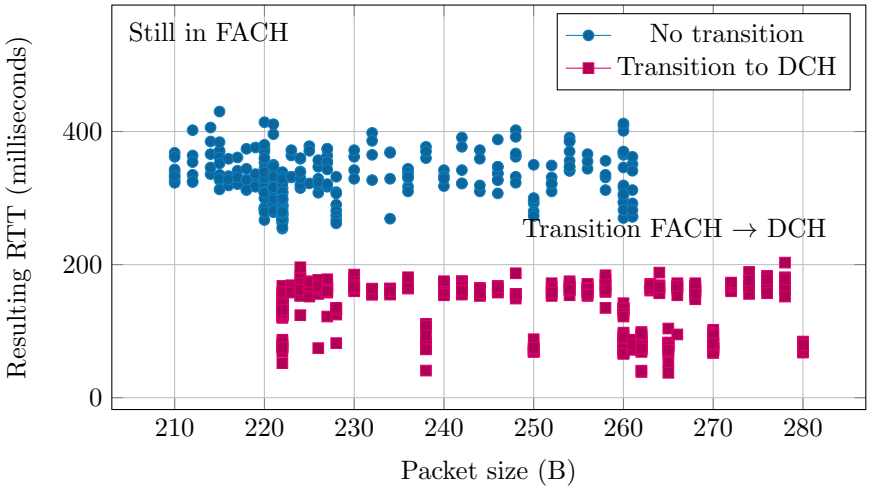

Figure 5.5: Estimation of the FACH to DCH buffer threshold.

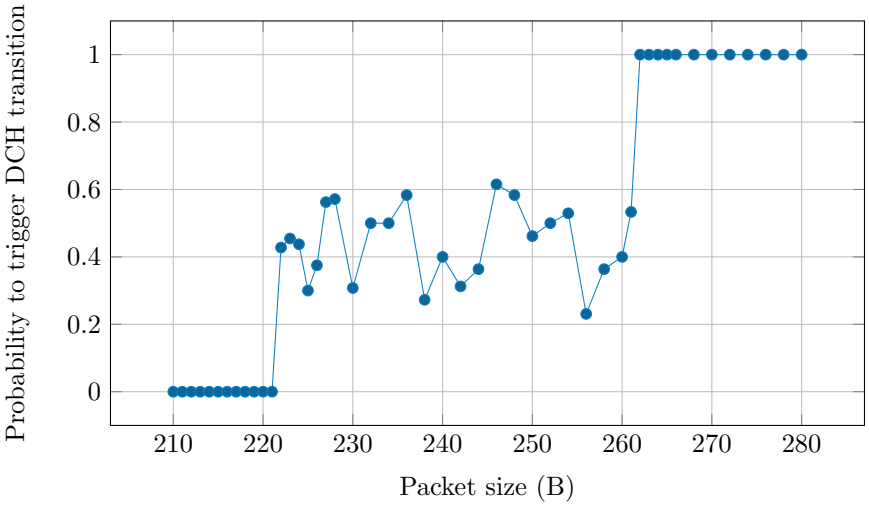

Figure 5.6: Probability to transition to from FACH to DCH.

mean RTT measured in the idle to DCH transition is 1180 ms. Therefore the approximate transition duration $T_{Idle-DCH}$ = 1180 - 181 = 999 ms.

Transition Duration $T_{FACH-DCH}$ To measure the $T_{FACH-DCH}$ delay, we send small packets every two seconds to keep the device in the FACH state. We also send a large (1300 B) ping packet every 8 seconds to trigger the FACH to DCH transition and record the RTT for this packet. The reason for sending the packet every 8 second, is because we want the device to transition back to FACH before sending the next ping. The test is repeated 100 times. To account for the usual delay in the DCH state, we must reduce the measured RTT with the RTT of a ping when in the DCH state for the 1300 B large packets. The mean RTT for the DCH state was previously measured to 181 ms, and the mean time measured for the FACH to DCH transition is measured to 798 ms. Therefore the approximate transition duration $T_{FACH-DCH}$ = 798 - 181 = 617 ms.

Using a similar approach as the ones above, we estimated $T_{Idle-FACH}$ as 515 ms and $T_{DCH-FACH}$ as 981 ms.

5.1.4 Measuring Bandwidth

DCH Bandwidth The uplink and downlink bandwidth in the DCH state was measured using the Ookla Speedtest.net service [1] by using the available Android application. Just before we measure the bandwidth, a large packet (1300 B) is sent to trigger a transition to DCH. The test was repeated 30 times for uplink bandwidth and 30 times for downlink bandwidth. Figure 5.7

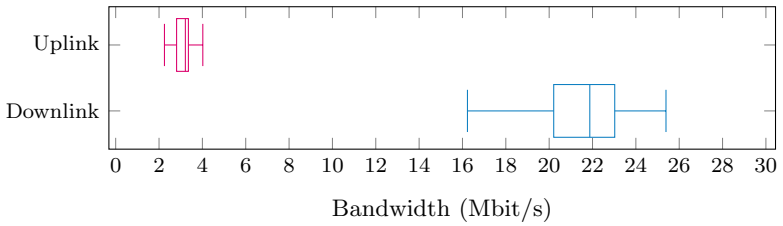

Figure 5.7: 3G bandwidth in DCH

shows a box plot for the test. The median downlink bandwidth is calculated to 21.9 Mbit/s and the median uplink bandwidth is 3.20 Mbit/s.

FACH Bandwidth The FACH bandwidth is determined by the FACH to DCH buffer threshold and by the speed which the buffer is emptied. Therefore, since the FACH to DCH buffer is the same size on both uplink and downlink, the bandwidth will also be the same. The Bandwidth is measured by sending ping packets, and calculating the maximum bandwidth we can achieve. The highest throughput we were able to achieve while still in the FACH state was when sending 220 B packets (just below the FACH - DCH threshold) every 510 ms, which translates to $220/0.51 \approx 431.25$ B/s = 3.45 kbit/s.

5.2 Impact of WiFi on 3G Emulator

The WiFi network restricts the 3G emulation to some degree. We have to consider the latency introduced in the WiFi network, as well as the bandwidth of the WiFi interface since these will interfere with the emulation. In this section we will study these factors to see how they will impact our 3G emulation.

5.2.1 Impact of WiFi RTT

To measure the RTT characteristics of WiFi, we want to capture the characteristics during the course of a complete day to consider the variations which occur during the course of a full day. We use the ping tool on an Android phone connected to the WiFi network where Snigel is installed. We sent 10 pings/second to a nearby external server with stable connection (www.su.se) over the course of roughly 24 hours, which resulted in 800,000 RTT samples. The same test was also conducted on the Telia 3G network. To keep the device in the DCH state during the entire test, we sent one large packet (2000 B) each second to prevent state downswitch.

The frequency 10 pings per second was empirically verified to be the highest multiple of 10 which did not deteriorate the result. Higher frequen-

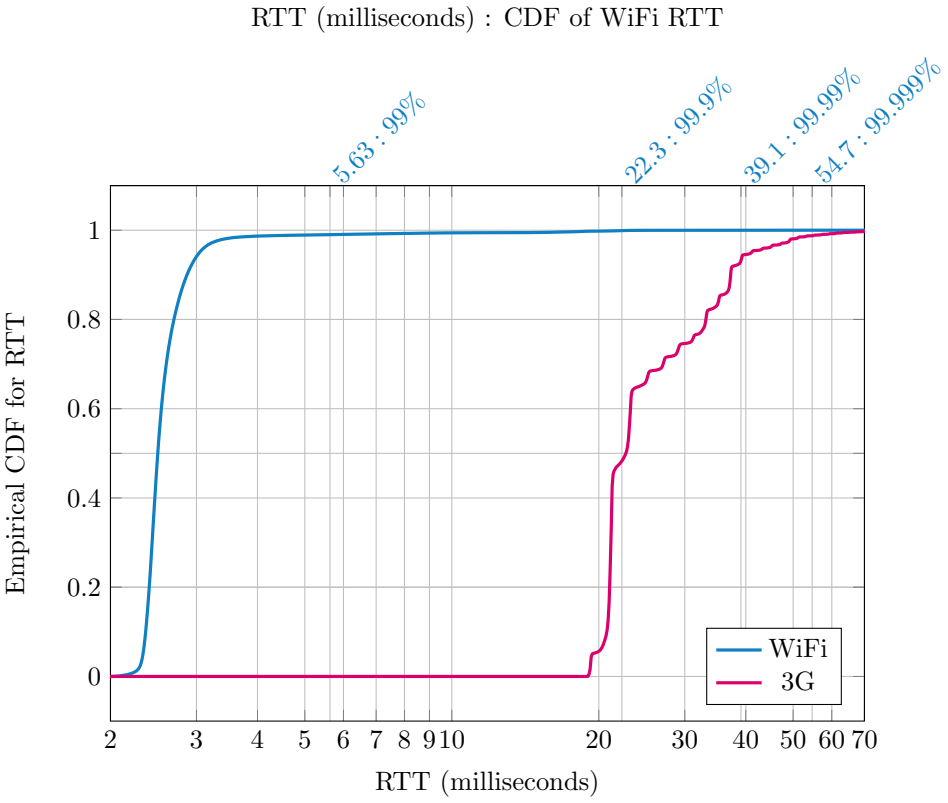

Figure 5.8: Empirical CDF for RTT over WiFi and 3G based on 800,000 samples for each connection.

	WiFi	3G
Mean RTT	2.68 ms	26.4 ms
Median RTT	2.50 ms	22.8 ms
Standard deviation	1.39 ms	8.67 ms
Minimum RTT	1.90 ms	18.9 ms
Maximum RTT	81.2 ms	258 ms

Table 5.2: Statistical values for the WiFi and 3G RTT tests.

cies were tried, but this propagated in more inconsistent results. Therefore, 10 pings per second is used hereinafter used when measuring RTT.

The results from the test can be seen in Figure 5.8, where a cumulative distribution function has been calculated using the 800,000 ping samples. One can see that the WiFi connection is quite stable and adds only a small delay of around 2.68 ms when only one user is connected as in this test. If we assume that we want at least 99.9% chance to correctly emulate a certain RTT, we need to look at the distribution of RTT over WiFi. We can see that 99.9% of the WiFi RTT samples are equal to or lower than 22.3 ms. By accepting 22.3 ms as the baseline, and given the RTT values of the 3G network we want to emulate, the risk of an error when emulating a 22.3 ms RTT (a packet is delayed more than what we expect) is 0.01%

As Table 5.2 shows, the maximum RTT measured during the test was 81.2 ms. This means that there are a few outliers among the samples, but this should not impact the performance of the emulator significantly since the vast majority of samples are close to the median, as seen in Figure 5.8 and in the standard deviation of 1.39 ms. Keep in mind that these measurements corresponds to the WiFi network we have set up at the Spotify office, and should not be interpreted as a general performance measure of WiFi networks.

The mean RTT for the 3G samples as shown in Table 5.2 is 26.4 ms, this RTT is more than the baseline of 22.3 ms, which means that this 3G network should be possible to emulate in our test environment without any performance issues. Also, from the statistics Table 5.2, we can see that the median RTT of 3G is close to the lowest 3G RTT measured. In Figure 5.8 we can see that the 3G distribution looks much like a folded normal distribution. This means that most samples are close to the median, and the remaining samples are higher than the median. This is a characteristic that we have taken into consideration when developing the 3G emulator by randomly adding an extra delay on only a selection of the packets. This means that around 6% of the packets gets delayed by a random amount of time in addition to the usual emulated RTT.

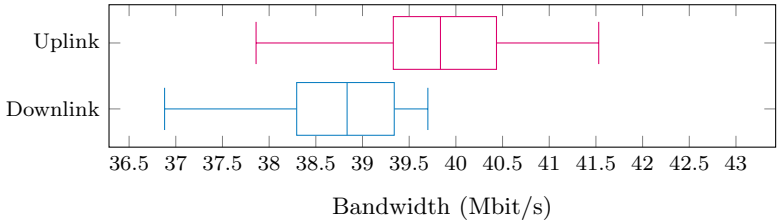

Figure 5.9: WiFi bandwidth.

5.2.2 WiFi Bandwidth

The WiFi bandwidth is measured using the Ookla Speedtest.net service [1], using the Android application. The test is repeated 30 times for both uplink and downlink bandwidth. The results are presented in Figure 5.9, where the median downlink bandwidth is measured to 38.8 Mbit/s, and the median uplink bandwidth is 39.8 Mbit/s. This will be the absolute maximum throughput which the 3G emulator can emulate with the current equipment. We can thereby expect to see degraded performance when the emulator tries to emulate bandwidths close to 39.8 Mbit/s.

Chapter 6

3G Emulator Evaluation

In this chapter, the 3G emulator is evaluated using combinations of synthetic network traffic and realistic usage of the Spotify Android client. The emulator uses the network parameters measured in chapter 5.

The purpose of this chapter is to evaluate the 3G emulator's similarity to a real 3G connection. The real 3G connection is our ground truth, which the emulator would ideally mimic perfectly. However, we do not expect to achieve a perfect emulation and will therefore also make comparisons between the 3G emulator and Snigel, the currently active network testing system at Spotify.

6.1 Metrics

There are three aspects we take into consideration in the evaluation of the 3G emulator: RTT, throughput and data communication pattern.

The main metric used in the evaluation of latency and packet inter-arrival times is the Pearson product-moment correlation coefficient (PPMCC) of the Cumulative Distribution Function (CDF). The PPMCC provides the linear correlation between two variables describing the similarity. 1 is a total positive correlation, whereas 0 represents no correlation. The PPMCC between different variables can then be compared to each other to determine the similarity between different network connections. The formula for calculating the PPMCC for two variables $\{x_1, ..., x_n\}$ and $\{y_1, ..., y_n\}$ is:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Where:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

When performing the tests, it is clear that different sessions in the same network show differences, this means that the variations in the network will have to be considered in the baseline. Having the 3G as baseline is hard to handle since it is not a stable connection. Each measurement we do on 3G is different from the next measurement. Therefore, we make five measurements on the inter-arrival times on 3G, create the CDF graphs for each measurement, and calculate the PPMCC for the first 3G CDF to the second 3G CDF, then we do the same for all combinations of the 3G measurements to finally calculate the average PPMCC, which is considered our baseline.

When regarding throughput evaluation, we use the relative error to the baseline 3G throughput, where the relative error is calculated as:

$$\delta x = \frac{\Delta x}{x}$$

Where x is the 3G throughput (baseline) and Δx is the absolute error:

$$\Delta x = |x_0 - x|$$

Where x_0 is the measured throughput of the 3G emulator, or Snigel.

6.2 Operational Parameters

During the evaluations in this thesis, we use the network parameters presented in section 5.1. To get the emulator working, we also need some extra parameters for the bandwidth restriction in DCH state to behave as we expect. These parameters are presented in Table 6.1. The backlog limit is the amount of bytes that the emulator can store before they are sent. If the backlog is full, additional packets will be dropped. The token limit is the maximum amount of tokens that the emulator can store. One token represents sending one byte.

The values of these parameters are empirically derived. A too low backlog limit will result in bandwidth being throttled, and a too high backlog limit will result in increased RTT since packets wait in the queue for a long time. The value of this parameter is determined by gradually increasing it while measuring the bandwidth until the bandwidth is close to the expected value. For any bandwidth that is entered in the emulator, the value of this parameter will have to be reevaluated. The token limit determines the "burstiness" of the emulator. We do not want the emulator to allow any excessive bursts, and this parameter should therefore be as low as possible while still providing the full bandwidth.

6.3 Functional Testing of RRC States

We verified the buffer sizes which cause RRC state transitions and state inactivity timers of the 3G emulator by sending packets of controlled sizes

Emulator Parameter	Value
Backlog Limit	500000 B
Token Limit	25000 B

Table 6.1: Parameters used by the 3G emulator bandwidth restriction.

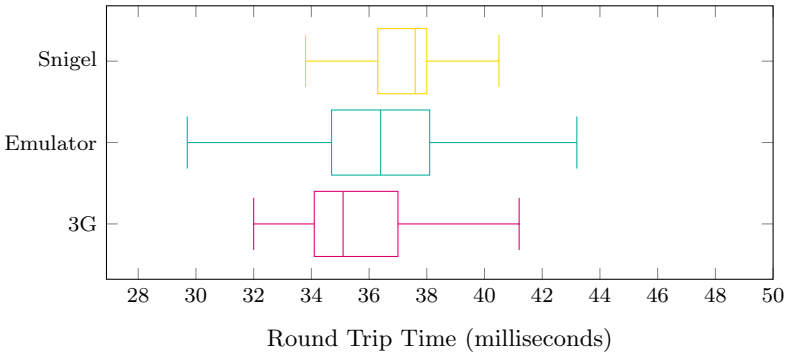

Figure 6.1: Box plot over 3000 RTT samples.

according to the same algorithmic methods used for finding the 3G network parameters in chapter 5.1. The results are positive, and the emulator behaves as expected.

6.4 Round Trip Time

The RTT has a quite significant impact on the user experience. The high RTT of the 3G network might decrease the responsiveness of the application under test even when very small amounts of data are sent or received. The RTT emulation of the 3G emulator is therefore evaluated and compared to the real 3G network, our baseline. This section also includes comparisons with Snigel.

As we want to measure the RTT at a specific moment, but also want to make an accurate measurement, we measure sample the RTT during 60 seconds. For the same reason as in section 5.2.1, we use the same frequency of 10 pings per second, which is enough to keep the device in DCH state during the entire test. During the second period we send 600 pings using the ping tool in a similar way as we perform the tests described in section 5.2.1. To obtain the baseline, we first measure the RTT to a nearby server with a quick connection (www.su.se). The test is repeated 5 times, which results in 3000 RTT samples. We measure the mean RTT of all 3000 samples to be 37.9 ms.

To evaluate the accuracy of the RTT in the emulator, the RTT we mea-

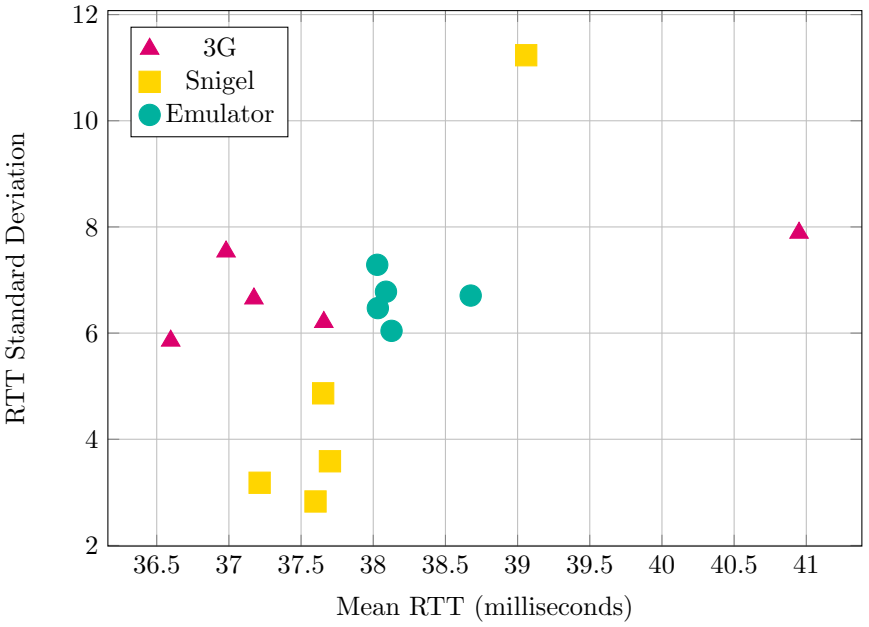

Figure 6.2: Compilation of RTT metrics: Mean RTT and RTT Standard Deviation.

sured for our baseline, the 3G network is used as configuration setting to the 3G emulator and Snigel. All other configuration parameters are the ones we measured in chapter 5.1. Now, we measure the RTT on the 3G emulator and Snigel in the same way as we measured our baseline RTT.

The distribution of all RTT samples are presented as a box plot in Figure 6.1. The mean RTT for the 3G emulator is measured to 38.2 ms and Snigel was measured to 37.6 ms. 3G, Snigel and 3G emulator shows very similar results in regards to mean RTT. The evaluation is further visualised as a scatter plot in Figure 6.2 where the mean RTT and standard deviation is presented, grouped in sample sizes of 600 (one test iteration).

Furthermore, Figure 6.2 shows that there is a significant difference in regards to standard deviation within the test iterations, where Snigel generally shows lower standard deviation than 3G and 3G emulator, median at 3.6 ms for Snigel versus the median standard deviation of 3G and 3G emulator which is almost the double, at 6.6 and 6.7 ms respectively.

To capture the entire characteristics for the RTT, we create an empirical Cumulative Distribution Function (CDF) shown in Figure 6.3, where all 3000 samples for all network connections are considered. The empirical CDF plots the chance that a RTT sample is below a given value. This figure gives a representation of the RTT distribution on the different network connections, and is therefore interesting from the evaluation standpoint.

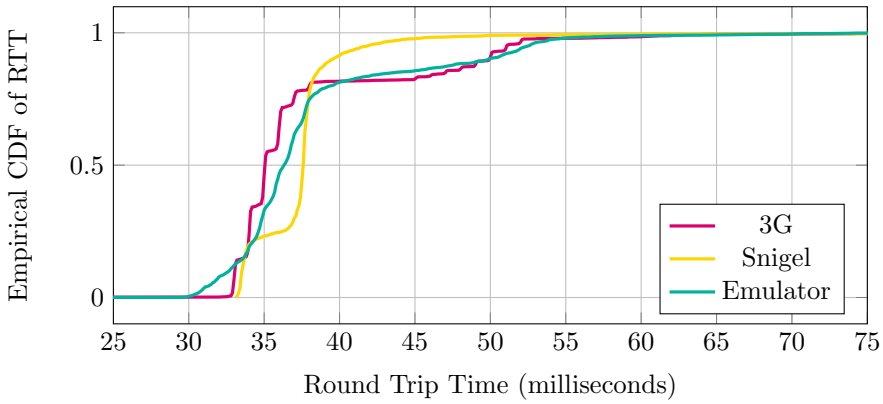

Figure 6.3: Empirical CDF for the RTT using 3G, Snigel and the 3G emulator.

PPMCC	3G
Emulator	0.997
Snigel	0.959

Table 6.2: PPMCC of CDF of RTT between 3G emulator and Snigel compared to 3G.

Snigel seems to capture the RTT between 30 to 33 ms better than the 3G emulator, while the 3G emulator seem to handle RTT from 33 to 55 ms better than Snigel. Also interesting to note is the staircase shape of the real 3G connection, with one step every millisecond. A similar shape is barely noticeable in the 3G emulator.

To make comparisons between the CDF's of different network types, we use the PPMCC explained in section 6.1. The PPMCC between 3G and 3G emulator and between 3G and Snigel are presented in Table 6.2. We can see that the correlation between 3G and 3G emulator is slightly higher than the correlation between 3G and Snigel. The increase in correlation we get from using 3G emulator instead of Snigel is calculated to $(0.997 - 0.959)/0.959 \approx 4\%$.

In conclusion, both the 3G emulator and Snigel can simulate RTT accurately. However, the 3G emulator achieves slightly better results in terms of correlation of the RTT distribution.

6.5 Throughput

The throughput emulation is an important aspect of the 3G emulator. It must be able to emulate both the downlink and uplink throughput of the

3G network. We therefore consider the median throughput of the real 3G connection to be our baseline that we want to mimic with the 3G emulator.

6.5.1 Throughput Emulation

The first part of the throughput evaluation is to evaluate the accuracy, and compare it to a real 3G connection as well as Snigel. We first configure the 3G emulator and Snigel with the parameters presented in Table 5.1 to reflect the 3G network. The most important parameters being the downlink bandwidth on 21.9 Mbit/s and uplink bandwidth on 3.2 Mbit/s. Again, we use the Ookla Speedtest.net service to measure the bandwidth. We repeat the measurement 30 times on each network type (3G, Snigel and 3G emulator).

Downlink The results from this test are presented as a box plot in Figure 6.4, where the whiskers represent the minimum and maximum recorded bandwidths. We can see that the real 3G network has a median bandwidth of 21.9 Mbit/s, but also has a quite large interquartile range (IQR) of 3.0 Mbit/s. However, this spread was not taken into consideration when emulating the throughput. The goal of the bandwidth emulation is to provide an accurate and stable emulation to provide an environment where tests are repeatable. This means that we value a stable emulation with IQR much higher than an emulation which has similar spread as the baseline. The 3G emulator provides a stable emulation with an IQR of just 0.4 Mbit/s and throughput median of 21.8 Mbit/s, which gives us a relative error compared to the baseline of 0.46%.

Snigel on the other hand has a median of 25.5 Mbit/s, which gives a relative error to the baseline of 14%. Snigel also has a higher IQR of 4.5 Mbit/s.

Uplink The median 3G uplink throughput is 3.2 Mbit/s. The 3G emulator has a slightly higher median throughput, 3.6 Mbit/s, which gives a relative error of 11%. The 3G emulator achieves a small IQR of 0.2 Mbit/s.

Snigel provides similar results, with a median of 3.7 Mbit/s, which gives a relative error of 14%. Snigel achieves an IQR on 0.04 Mbit/s, which is very good. Both the 3G emulator and Snigel manage to emulate the real 3G network bandwidth quite well, but the 3G emulator behaves much more accurately on the downlink throughput, while the uplink throughput is managed equally.

We observed that the uplink bandwidth emulation was much worse than the downlink emulation. We did not, however search for the reason to this differentiation. It would be interesting to try the same tests with a different WiFi hardware, to see if the problem might be in the WiFi switch, or even in the phone's WiFi interface.

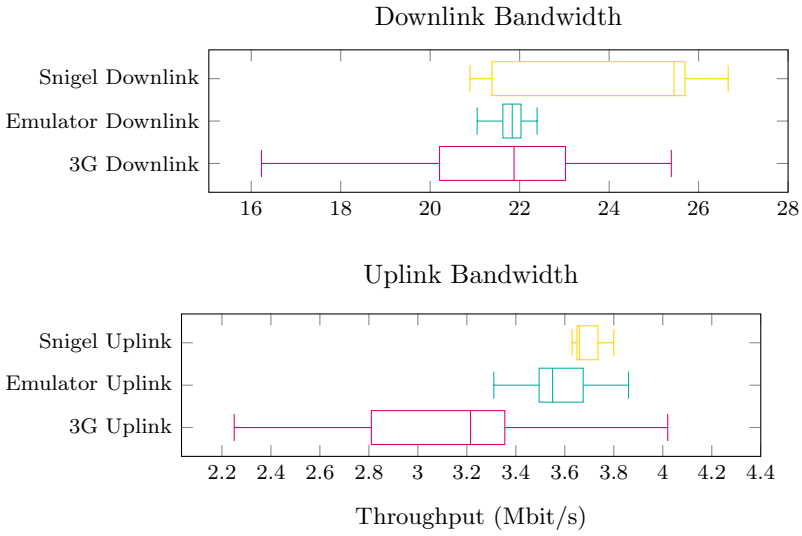

Figure 6.4: Box plots over 30 throughput samples for uplink and downlink.

6.5.2 Operational Range of Throughput

To compare the bandwidth restriction accuracy and the operational range between Snigel and the 3G emulator more closely in a large range of settings, we perform a test where we make measurements on both Snigel and the 3G emulator using the Ookla Speedtest.net service on eight selected bandwidths between 0.5 and 64 Mbit/s. The baseline in this test is the expected value of the test, e.g if 4 Mbit/s is used as input to the emulator, then 4 Mbit/s is the expected result and baseline.

This test was performed by entering a desired bandwidth in the 3G emulator and Snigel, then measuring the resulting throughput with Ookla Speedtest.net, and recording the average of five measurements. The results from the test are presented in Figure 6.5, where each mark represents the average of the measurements at the desired throughput. Figure 6.6, depicts the relative error to the desired throughput. The relative error is calculated as explained in chapter 6.1.

Downlink We can see that the downlink performance is very good up to 32 Mbit/s on both systems. After 32 Mbit/s, neither of the systems can keep up with the speed. The reason for this is the WiFi network that bottlenecks the throughput performance. When the emulator is disabled and the WiFi switch is used on its own, we get a median downlink throughput of 38.8 Mbit/s as was measured in chapter 5.2.2, a throughput limit which the emulation is getting close to. This explains the performance degradation

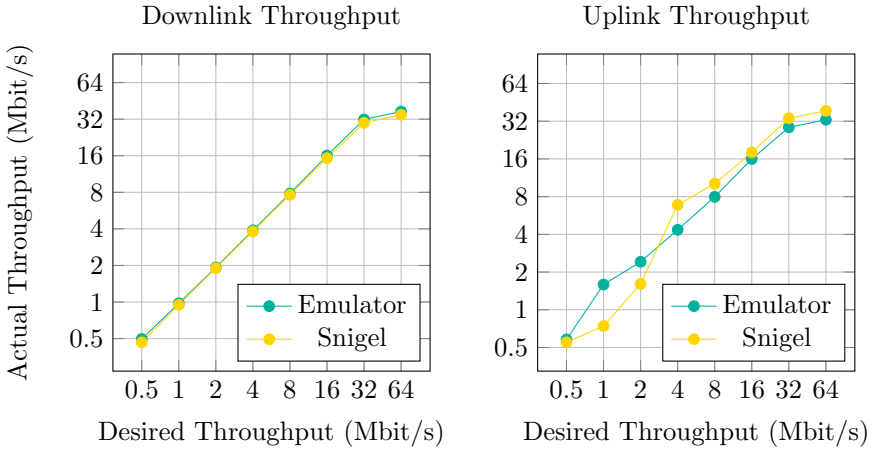

Figure 6.5: Desired throughput and the measured throughput for the 3G emulator and Snigel.

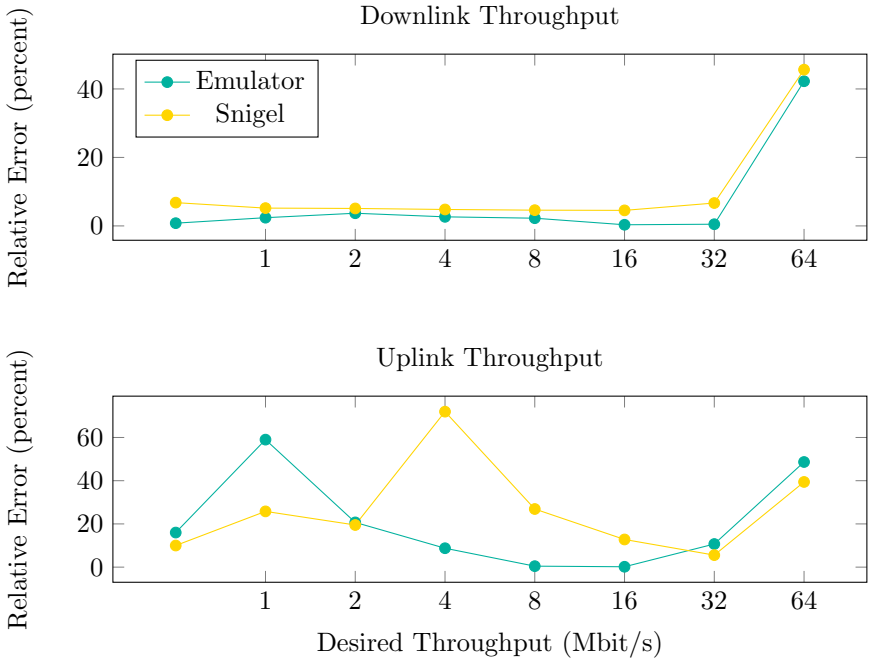

Figure 6.6: Desired throughput and the relative error for the 3G emulator and Snigel.

after 32 Mbit/s.

Uplink The uplink performance is generally worse than the downlink performance. As in the downlink emulation, we experience a degradation after 32 Mbit/s, as we are getting close to the limit of the WiFi network. The uplink emulation also performs worse than the downlink emulation at low bandwidths. As seen in Figure 6.5, the Snigel emulation results in almost the double desired throughput as the downlink throughput results in 6.9 Mbit/s when it should have emulated 4 Mbit/s. The same kind of performance degradation as in Snigel is also measured in the 3G emulator.

Both Snigel and the new 3G emulator experience similar problems in uplink throughput restriction, and the 3G emulator use the same code for both uplink and downlink emulation. This makes us believe that the problem which cause this degradation lies elsewhere than in the software. The problem probably is the WiFi interface in either the smartphone or the WiFi switch which interferes with the emulator. We did not investigate this any further, but it is an interesting aspect of the bandwidth emulation to investigate further in future development.

6.6 Data Communication Patterns

RTT and throughput plays a large part in the 3G characteristics, but these factors do not take the RRC state transitions into consideration. The delays caused by the state transitions have a huge impact on the characteristics as well, especially in situations with intermittent network traffic when state transitions are frequent. Previously, we have only used synthetic traffic to evaluate the emulator. This chapter aims to prove that the suggested solution emulates 3G realistically even when used on a real application with use-cases based on real world usage. Realistic in this context, is the notion of having similar distribution of inter-arrival times compared to 3G, when the same automatic test is conducted on 3G as well as on the 3G emulator.

To evaluate the 3G emulator in a realistic scenario, the packet traces from three Spotify use-cases are recorded. Spotify is a streaming music streaming service which uses the relatively high bandwidth of a mobile connection to download small music files. The files are downloaded in batches, starting with a couple of seconds of the song at first, then gradually increasing the burst sizes until the entire song is downloaded. To make sure the tests are performed in the same way each time, we use the monkeyrunner tool which is delivered as part of the Android SDK. Monkeyrunner lets us simulate screen touches on a real devices to perform deterministic GUI tests.

The phone has a Facebook account logged in to allow login using Facebook in Spotify. The Spotify account has a premium subscription to prevent commercials interfering with the test. The test is performed on a “rooted” Samsung Galaxy S3 (gt i9305) with no other applications than Spotify and Facebook installed (except for the default apps still present after factory

reset). The data network traffic from Spotify is isolated by restricting the background data on all installed apps except for Spotify using the built in Android feature.

Unfortunately, it is impossible to perform these tests in a completely deterministic manner since we are not in control of the load on our Internet service providers network or the load on the Spotify server. For these reasons, we perform the tests five times on each network connection to see which variations we encounter.

We first measure the 3G network parameters as explained in chapter 5.1, and apply these to the 3G emulator and Snigel to emulate the 3G network as closely as possible. All tests in this section are to five times on 3G, Snigel and on the 3G emulator. The packet traces are captured using the tcpdump tool installed on the Android phone. This means that outgoing data is captured as they leave the phone and incoming data is captured when it arrives to the phone. The same measurement setup is used in all tests, regardless of which network connection is used.

The characteristics from the test are presented as empirical CDF of packet inter-arrival time, which is the time between two consecutive packets. This graph gives an overview over the patterns in the network traffic. The data that is sent in the tests, regardless of which connection is used should have the same packet sizes. What differs is therefore the frequency that the packets are sent and received. The inter-arrival time metric captures the time between packets, but also aspects such as bandwidth, since a lower inter-arrival time suggests high throughput as the packet size is the same. The baseline in these tests is the empirical CDF of the 3G connection. To determine the similarity between two iterations of the test, we calculate the PPMCC of all combinations of CDF. The RRC should have more impact on test cases with intermittent traffic since then the RRC will make more state transitions. We therefore expect to see an increase in accuracy compared to Snigel when intermittent traffic cases are considered.

The three different use-cases are described in text in the following sections.

6.6.1 Streaming Music

This Spotify session represents an active listening scenario. The user searches for a band, starts listening and skips some songs. Then searches for a new band and listens for a couple of minutes. The Spotify cache and data are deleted before each test to clear all cached songs and account information. This test took 9 minutes and resulted in around 35 MB data being transferred. This is the core use-case in the Spotify app, and should therefore be the most interesting in this evaluation. This use-case contains intermittent traffic, with some periods of high intensity traffic, and some periods with low intensity traffic. This intermittent traffic should make the RRC state machine trigger many state transitions, which the 3G emulator should

handle much better than Snigel. The Spotify session is described in detail below:

1. Start Spotify
2. Log in using Facebook
3. Search for “ARCTIC MONKEYS”
4. Open the Arctic Monkeys artist page
5. Start the first song (“Do I Wanna Know?”)
6. Open the album view
7. Swipe to the next song four times (1-5 seconds between each swipe)
8. Swipe to the next song four times quickly (1 second between each swipe)
9. Close the album view
10. Search for “DUNDERTAGET”
11. Open the Dundertåget artist page
12. Start the first song (”Ifrån mej själv”)
13. Keep listening for 5 minutes (the next song will start playing automatically)

6.6.2 Downloading Playlist

This Spotify session contains a download of a playlist with 10 songs. The songs are downloaded in normal quality. The Spotify cache and data are deleted before each test. This test took 2 minutes and resulted in around 45 MB data being transferred. This case is a very network intense, therefore the improvement over Snigel is not expected to be as significant as in the other two cases since the RRC state machine will not trigger as much transitions, the DCH state will be active most of the time.

1. Start Spotify
2. Log in using Facebook
3. Enter the settings screen and enable the “sync over mobile network” setting
4. Open the playlist
5. Press the “Available offline” button
6. Wait two minutes for the download to finish

6.6.3 Playing Offline Music

This session represents a use case which should not be as data traffic intense as the previous tests. Before we do the test, we download a playlist of five songs. The test is simply to listen to this playlist during 9 minutes, which corresponds to three complete songs, and just starting the fourth. Listening to a downloaded playlist does not require nearly as much data as streaming music, but low volumes of traffic such as log data and user statistics are still being sent as usual. The playlist is downloaded before the first iteration of the test, and auto login is enabled. This test took 6 minutes and resulted in around 300 KB data being transferred. This very low intensity case, where the RRC state machine will be in FACH or idle state much of the time, we expect to see some improvement over Snigel, credited to the RRC transitions and the FACH bandwidth restriction.

1. Start Spotify
2. Open the playlist
3. Start playing the first song (shuffle is disabled)
4. Wait nine minutes

6.6.4 Data Pattern Analysis

To see the difference of the three use-cases, the characteristics of the packet inter-arrival times and packet sizes are presented in Figure 6.7. In this figure, we can see that we expect to see much longer inter-arrival times when listening to offline music than when we are streaming or downloading. Also, the packet sizes are more diverse when listening to offline music, as opposed to streaming, where almost 92% of the packets are 1388 B, and when downloading, where 30% of the packets are 1388 B, and 65% of the packets are 1400 B.

It is important to understand that even sessions performed on the real 3G network differs each time we perform a test. Therefore, we perform several sessions of each test even on the 3G network, and use the 3G to 3G correlation coefficient as baseline.

The baseline for this test is calculated by calculating the PPMCC of 3G packet traces to other 3G packet traces for the same 3G connection.

The PPMCC to real 3G of the empirical CDF is calculated for all connections and presented in Table D.1, D.2 and D.3 in appendix D. However, to simplify the reading of these tables, we summarised the data by presenting the mean PPMCC to 3G for each connection in Table 6.3. We can see that the 3G emulator achieves a slightly higher correlation to 3G than Snigel. This means that the 3G emulator behaves more similarly to 3G than Snigel. For details of the empirical CDF:s for all three tests, see Figures C.1, C.2 and C.3 in appendix D.

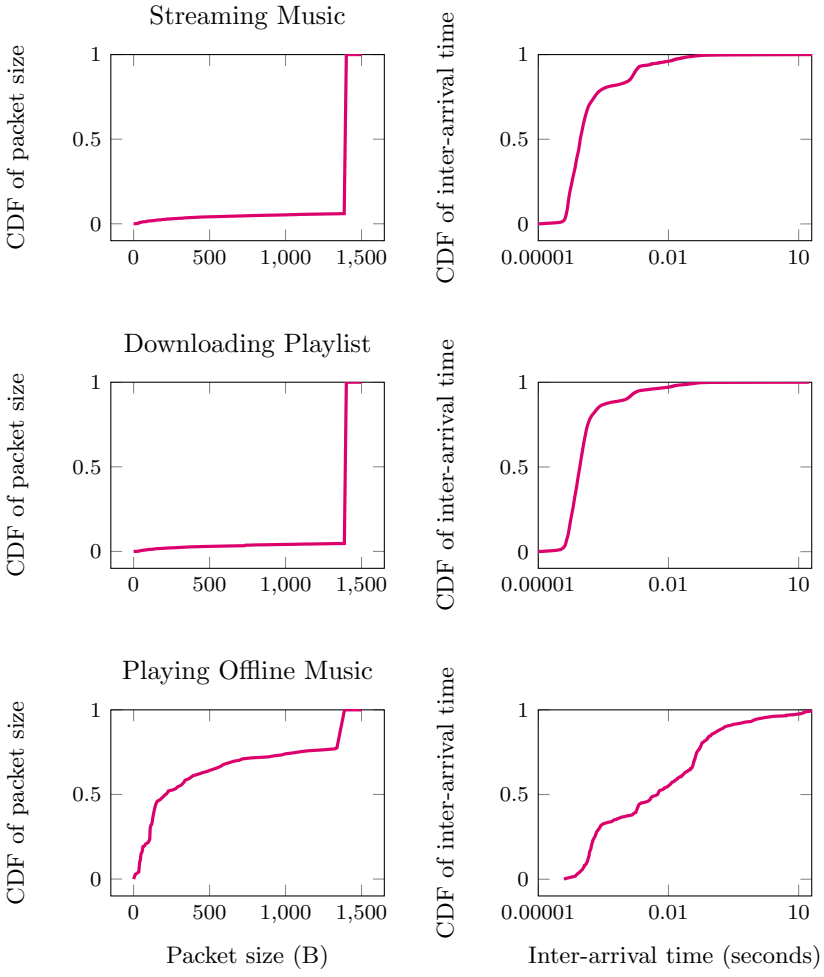

Figure 6.7: The data pattern and inter-arrival times for the three Spotify use cases measured using 3G in a mobile device.

PPMCC to 3G	Music streaming	Downloading playlist	Offline listening
3G	0.9982	0.9968	0.9979
Emulator	0.9731	0.9860	0.9708
Snigel	0.8775	0.9693	0.9343

Table 6.3: Mean PPMCC to 3G of inter-arrival time CDF in three use cases.

Relative Error	Music streaming	Downloading playlist	Offline listening
Emulator	2.6%	1.1%	2.8%
Snigel	14%	2.8%	6.8%

Table 6.4: Relative error of inter-arrival time CDF correlation.

We calculate the relative error to the baseline by calculating the absolute error of the PPMCC to the baseline and dividing this with the PPMCC. The relative errors are presented in Table 6.4. Snigel has an error of 2.8 to 14%, and by using the 3G emulator, we managed to get the error as low as 1.1 to 2.8%.

To summarise, the new 3G emulator provides an inter-arrival time distribution which has a relative error from a real 3G connection of 1.1 to 2.8%. This is a decrease from the current Snigel system, which achieved a relative error of 2.8 to 14%. This increase should provide Spotify the means to use a WiFi connection instead of the 3G network to performance test mobile apps.

As expected, the music streaming case shows a significant improvement in relative error from 14% in Snigel to 2.6% in the new 3G emulator. This improvement can be credited to the RRC state machine, which proves it's applicability to software testing. Since the playlist downloading case does not involve the RRC state machine as much, the result is not as significant, but still we see the relative error dropping from 2.8% to 1.1%, presumably due to better throughput restriction and RTT emulation. Lastly, the offline listening cases resulted in the relative error dropping from 6.8% to 2.8%, where again the RRC state machine have been involved in restricting the bandwidth during a period of low traffic by emulating the FACH state.

Chapter 7

Conclusions and Future Works

This chapter contains the final conclusions from the works performed in this project. The feasibility of the proposed solution is discussed, and suggestions on future work related to this thesis are proposed.

7.1 Conclusions

In this thesis, we have proposed a solution for emulating 3G over a WiFi connection for use during the testing of mobile applications at Spotify. The 3G emulator will provide Spotify means to emulate 3G networks of varying quality. By using the emulator, Spotify gets the means to test mobile apps on a 3G-like network with a predefined quality which allows for performing repeatable tests.

The solution was evaluated and compared to the previously active network testing system Snigel. The RTT emulation in the new 3G emulator increases the RTT accuracy by 4% compared to Snigel.

The bandwidth emulation in the new 3G emulator performs better than Snigel in both downlink and uplink emulation. The downlink emulation shows a significant increase in accuracy, and achieves a relative error of 0.46% compared to the 14% of Snigel. The uplink emulation performs slightly better than Snigel, with a 11% relative error compared to the 14% of Snigel.

The characteristics of the resulting traffic and the behavior of the 3G emulator was compared to the one of Snigel. Snigel showed a relative error compared to a real 3G connection of 2.8 to 14%, while the 3G emulator showed convincing results with a relative error as small as 1.1 to 2.8%. This is a compelling result and this new 3G emulator should therefore be a promising new test tool for the industry. We expect the tool to be a valuable asset to the testing environment at Spotify.

7.2 Future Work

In this section, we discuss interesting ways this thesis could be extended for future academic work or further development internally at Spotify.

7.2.1 LTE Support

The current emulator developed for this project does only support the emulation of 3G connections. To add LTE support, a new WiFi switch which supports higher downlink throughput than the current hardware must be installed. Also, the RTT will have to be re-evaluated since LTE under good conditions can have very low RTT. It has to be ensured that the WiFi connection has even lower RTT than the LTE connection that is to be emulated.

The state machine within the emulator will have to be extended to support the conceptually similar states of LTE, and the rules for switching between states will have to be discovered, together with tail times as done in this thesis.

7.2.2 Emulator Accessibility

By providing easier means of accessing the 3G emulator, chances that it is used will increase. One could integrate the emulator with the usual WiFi network already present at the office. By doing this, it would be possible for developers to test their network code directly at their desks instead of having to connect devices to a special WiFi network in a dedicated testing room.

Additionally, providing some standard emulation profiles which simulate a certain signal strength on a certain network with all the correct parameters could decrease the knowledge needed to perform tests on the 3G emulator. At first, the user interface of the 3G emulator can seem intimidating, since it requires the user to enter parameters from the 3G network, which are not known by the average software developer.

7.2.3 Simultaneous Connections

The proposed solution in this thesis only allows for one connected user at a time. To provide 3G emulation for several devices at the same time, the 3G emulator could be modified and installed on a more powerful server. A more robust WiFi network would also be necessary to allow several connections without degrading the performance. Most of the work in this project would be to evaluate the solution, and create models for how many Spotify sessions can be accurately emulated at the same time. One has to take into consideration that only one device can communicate on the WiFi network at any given time, while other devices will have to wait for their turn to communicate. This might cause RTT fluctuations because of the waiting time, which can be hard to control.

7.2.4 Automatic RTT Control

If the emulator could adapt to such a situation by continuously monitoring the RTT and compensating the RTT addition to match the target RTT, it would be possible to allow the user to input the desired RTT instead of the RTT addition as in this thesis.

7.2.5 RTT and Bandwidth From Distributions

Instead of just adding a random jitter to the RTT, an improvement would be the ability to tell the 3G emulator to let the RTT follow a certain distribution, for example a normal distribution with a standard deviation as input. This could further improve the accuracy of the emulator, but would also require more thorough studies of the RTT in 3G networks, to see which distributions would be viable options. The same addition could be added to the bandwidth emulation as well.

7.2.6 Automatic Regression Tests and Energy Consumption

By implementing automatic regression tests which make use of the 3G emulator, it will be possible for Spotify to monitor the performance on mobile networks in a controlled environment and automatically report regressions in the Spotify apps. It could also be possible to implement EnergyBox [17] to the test environment to automatically analyse the battery consumption during automatic tests.

Bibliography

[1] Bauer, Steven and Clark, David D. and Lehr, William, *Understanding Broadband Speed Measurements* (August 15, 2010). TPRC 2010. www.ssrn.com/abstract=1988332

[2] Marta Carbone and Luigi Rizzo, *Dummynet revisited*, SIGCOMM Comput. Commun. Rev. 40, no. 2, 12–20. 2010.

[3] Linux Foundation, *NetEm*. 2009. www.linuxfoundation.org/collaborate/workgroups/networking/netem

[4] Bert Hubert et al., *Linux advanced routing & traffic control howto*, 2002. www.lartc.org

[5] H.K. Kalitay and M.K. Nambiar, *Designing wanem: A wide area network emulator tool*, Third International Conference on Communication Systems and Networks (COMSNETS), pp. 1–4,TCS Innovation Lab Performance Engineering, Jan 2011.

[6] Andres Lagar-Cavilla, *UMTS RRC emulation module*. 2011. www.lagarcavilla.org/software/rrc.html

[7] Miguel López-Benítez, Francisco Bernardo, Nemanja Vučević, and Anna Umbert, *Real-time HSPA emulator for end-to-edge QoS evaluation in all-ip beyond 3G heterogeneous wireless networks*, Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, pp. 8:1–8:12, Networks and Systems & Workshops, ICST, 2008.

[8] Huawei Technologies Co. Ltd., *LTE small cell v.s. WiFi user experience*, 2013. www.huawei.com/ilink/en/download/hw_323974

[9] Anders Grotthing Moe, *Implementing rate control in NetEm: Untying the NetEm/TC tangle*, University of Oslo, August 2013. www.duo.uio.no/bitstream/handle/10852/37459/Moe-Master.pdf

[10] Lucas Nussbaum and Olivier Richard, *A comparative study of network link emulators*, Proceedings of the 2009 Spring Simulation Multiconference, pp. 85:1–85:8, SpringSim '09, Society for Computer Simulation International, 2009.

[11] P.H.J. Perala, A. Barbuzzi, G. Boggia, and K. Pentikousis, *Theory and practice of RRC state transitions in UMTS networks*, GLOBECOM Workshops, pp. 1–6, IEEE, Nov 2009.

[12] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, and Oliver Spatscheck, *Characterizing radio resource allocation for 3G networks*, Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, pp. 137–150, ACM, 2010.

[13] O.M. Teyeb, M. Boussif, T.B. Sorensen, J. Wigard, and P.E. Mogensen, *Respect: a real-time emulator for service performance evaluation of cellular networks*, 62nd Vehicular Technology Conference (VTC-2005-fall), vol.1, pp. 402-406, IEEE, Sept 2005.

[14] J.L. Valenzuela, A. Monleon, I. San Esteban, M. Portoles, and O. Sallent, *A hierarchical token bucket algorithm to enhance QoS in IEEE 802.11: proposal, implementation and evaluation*, 60th Vehicular Technology Conference (VTC-2004-fall), vol.4, pp. 2659-2662, IEEE, Sept 2004.

[15] Karima Velásquez and Eric Gamess, *A comparative analysis of WAN emulators*, Proceedings of the 7th Latin American Networking Conference, pp. 44–51, ACM, 2012.

[16] Ekhiotz Jon Vergara Alonso, *Exploiting energy awareness in mobile communication*, Licentiate thesis, Linköping University, 2013. `www.diva-portal.org/smash/get/diva2:663464/FULLTEXT01.pdf`

[17] Ekhiotz Jon Vergara Alonso, Simin Nadjm-Tehrani, and Mihails Prihodko, *EnergyBox: Disclosing the wireless transmission energy cost for mobile devices*, Sustainable Computing: Informatics and Systems **4**, no. 2, 118–135, 2014.

[18] Anthony I. Wasserman, *Software engineering issues for mobile application development*, Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, pp. 397–400, ACM, 2010.

[19] Per Willars, *Smartphone traffic impact on battery and networks*, Ericsson, 2010. `www.ericsson.com/research-blog/uncategorized/smartphone-traffic-impact-battery-networks/`

[20] Zheng Zeng, Yan Gao, and P.R. Kumar, *Sofa: A sleep-optimal fair-attention scheduler for the power-saving mode of WLANs*, 2011 31st International Conference on Distributed Computing Systems, pp. 87–98, IEEE, June 2011.

Appendices

Appendix A

3G Emulator Flowcharts

The flowcharts in this appendix describe the functionality of the developed 3G emulator qdisc.

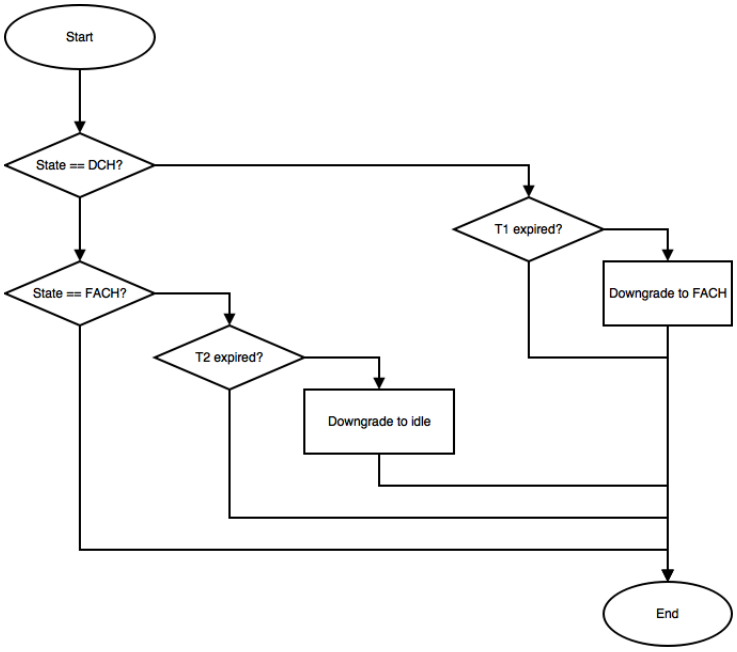

Figure A.1: Part 2 of enqueuing.

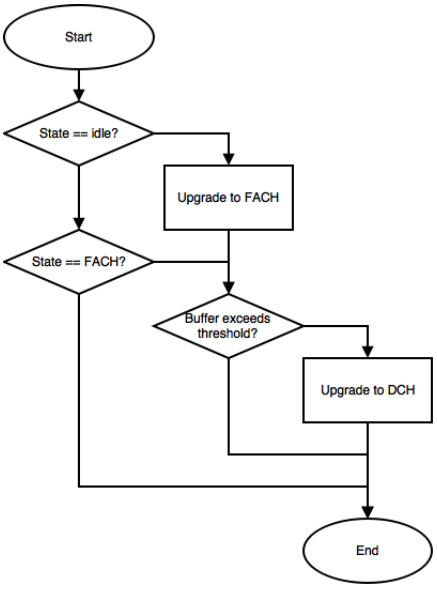

Figure A.2: Part 3 of enqueuing.

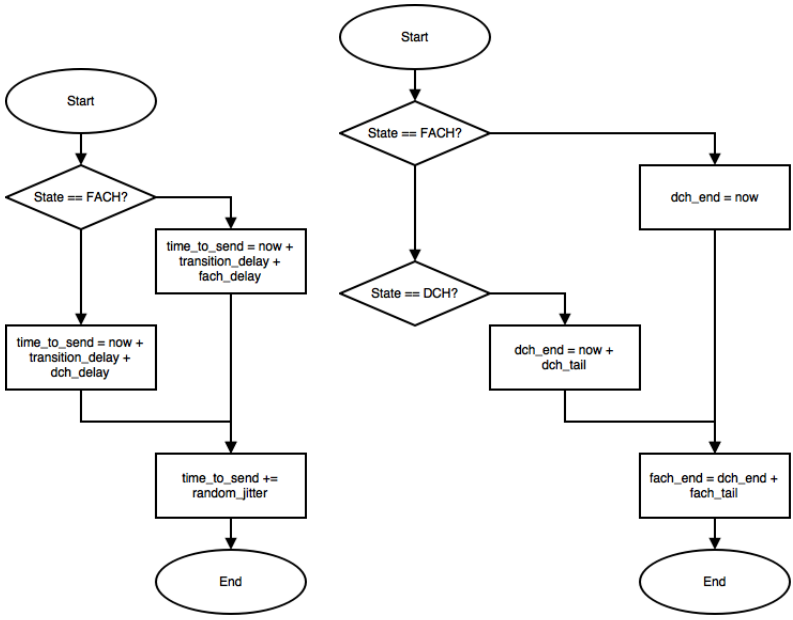

Figure A.3: Part 4 (Left) and 5 (Right) of enqueuing.

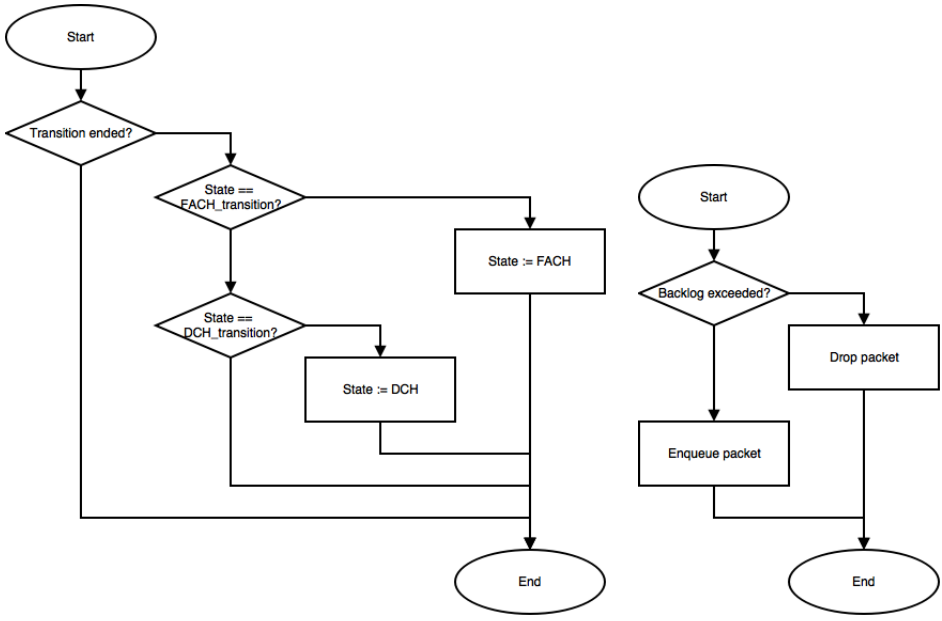

Figure A.4: Part 6 (Left) and 8 (Right) of enqueuing.

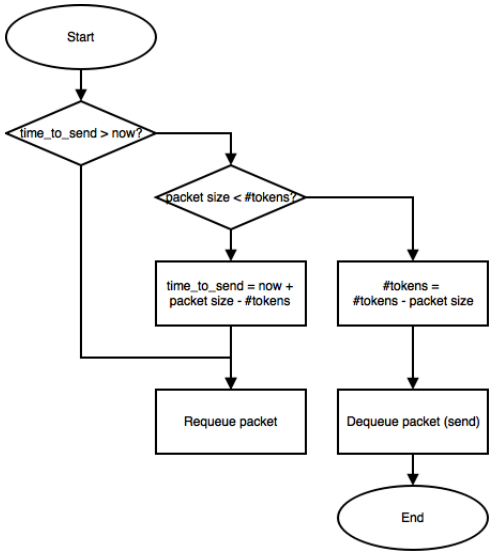

Figure A.5: Part 3 of dequeuing.

Appendix B

Playlists

The playlists presented here are the playlists used for in the evaluation of the 3G emulator.

Artist	Song
Red Hot Chili Peppers	Californication
Red Hot Chili Peppers	Otherside
Red Hot Chili Peppers	Under The Bridge
Red Hot Chili Peppers	Can't Stop
Red Hot Chili Peppers	Snow [Hey Oh]
Red Hot Chili Peppers	Scar Tissue
Red Hot Chili Peppers	By The Way
Red Hot Chili Peppers	The Adventures Of Rain Dance Maggie
Red Hot Chili Peppers	Dani California
Red Hot Chili Peppers	Soul To Squeeze

Table B.1: Playlist used for download test.

Artist	Song
The Offspring	Slim Pickens Does The Right Thing
Arctic Monkeys	Reckless Serenade
Iggy Azalea, Jennifer Hudson	Trouble
Sherlock Brothers	My Way
Tricky	Paranthesis

Table B.2: Playlist used for offline listening test.

Appendix C

CDF of Packet Inter-arrival Times

The three CDFs presented in this appendix depict the combined data from the evaluations of music streaming, playlist downloading and listening to a downloaded playlist.

As seen in Figure C.3, listening do a downloaded playlist results in higher concentration of high packet inter-arrival times. The reason for this is because there are much lower data traffic volume in this test, which results in longer times between each packet.

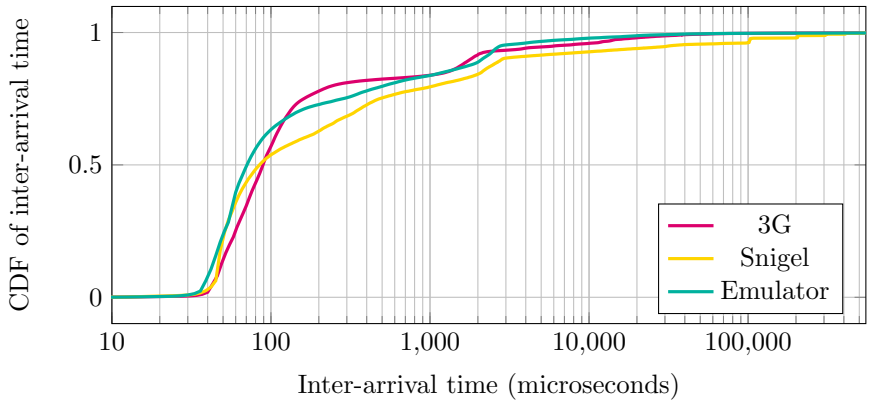

Figure C.1: Empirical CDF of packet inter-arrival times while streaming music on Spotify. Combined results of all five tests.

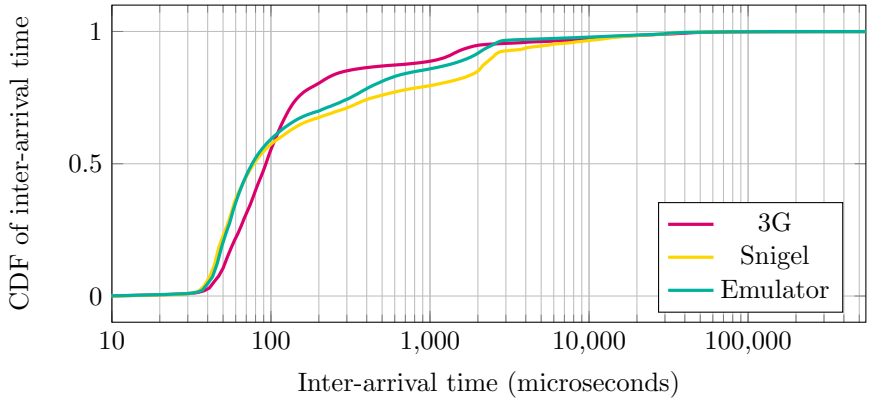

Figure C.2: Empirical CDF of packet inter-arrival times while downloading a Spotify playlist. Combined results of all five tests.

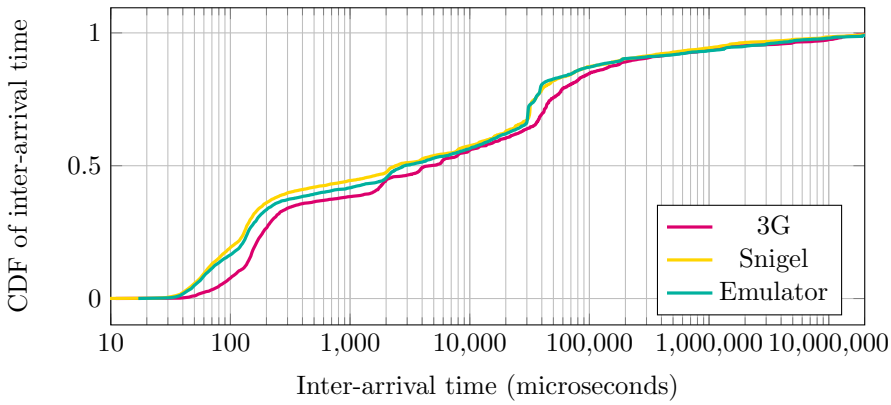

Figure C.3: Empirical CDF of packet inter-arrival times while listening to a downloaded playlist on Spotify. Combined results of all five tests.

Appendix D

Correlation Tables

The tables in this appendix are all combinations of PPMCC of the CDFs for the three use-cases evaluated in section 6.6. The averages of all 3G to 3G, 3G to emulator and 3G to snigel PPMCCs are used in the evaluation in chapter 6.

PPMCC	3G (1)	3G (2)	3G (3)	3G(4)	3G (5)
3G (1)	1	0.9974	0.9973	0.9995	0.9976
3G (2)	0.9974	1	0.9995	0.9972	0.9996
3G (3)	0.9973	0.9995	1	0.9971	0.9997
3G (4)	0.9995	0.9972	0.9971	1	0.9974
3G (5)	0.9976	0.9996	0.9997	0.9974	1
Emulator (1)	0.9798	0.9735	0.9759	0.9819	0.9748
Emulator (2)	0.9702	0.9622	0.9663	0.9717	0.9646
Emulator (3)	0.9713	0.9643	0.9677	0.9735	0.9663
Emulator (4)	0.9784	0.9716	0.9737	0.9809	0.9727
Emulator (5)	0.9807	0.9742	0.9755	0.9834	0.9749
Snigel (1)	0.8134	0.8155	0.8123	0.8123	0.8120
Snigel (2)	0.8325	0.8355	0.8310	0.8311	0.8312
Snigel (3)	0.9514	0.9490	0.9471	0.9471	0.9469
Snigel (4)	0.8917	0.8919	0.8890	0.8890	0.8889
Snigel (5)	0.9043	0.9025	0.8998	0.9046	0.8998

Table D.1: Pearson product-moment correlation coefficient between the packet inter-arrival time CDF while streaming music.

PPMCC	3G (1)	3G (2)	3G (3)	3G(4)	3G (5)
3G (1)	1	0.9979	0.9947	0.9998	0.9996
3G (2)	0.9979	1	0.9991	0.9981	0.9986
3G (3)	0.9947	0.9991	1	0.9951	0.9959
3G (4)	0.9998	0.9981	0.9951	1	0.9999
3G (5)	0.9996	0.9986	0.9959	0.9999	1
Emulator (1)	0.9667	0.9769	0.9830	0.9678	0.9691
Emulator (2)	0.9595	0.9722	0.9795	0.9608	0.9625
Emulator (3)	0.9633	0.9747	0.9814	0.9644	0.9659
Emulator (4)	0.9702	0.9790	0.9843	0.9713	0.9723
Emulator (5)	0.9622	0.9739	0.9808	0.9635	0.9650
Snigel (1)	0.9291	0.9458	0.9558	0.9284	0.9303
Snigel (2)	0.9176	0.9351	0.9458	0.9158	0.9181
Snigel (3)	0.9188	0.9366	0.9475	0.9177	0.9198
Snigel (4)	0.9197	0.9373	0.9482	0.9186	0.9207
Snigel (5)	0.9419	0.9566	0.9658	0.9417	0.9434

Table D.2: Pearson product-moment correlation coefficient between the packet inter-arrival time CDF while downloading a playlist.

PPMCC	3G (1)	3G (2)	3G (3)	3G(4)	3G (5)
3G (1)	1	0.9990	0.9970	0.9944	0.9983
3G (2)	0.9990	1	0.9953	0.9921	0.9965
3G (3)	0.9970	0.9953	1	0.9985	0.9990
3G (4)	0.9948	0.9921	0.9985	1	0.9982
3G (5)	0.9983	0.9965	0.9990	0.9982	1
Emulator (1)	0.9897	0.9924	0.9833	0.9817	0.9868
Emulator (2)	0.9915	0.9942	0.9840	0.9812	0.9873
Emulator (3)	0.9897	0.9912	0.9864	0.9873	0.9895
Emulator (4)	0.9922	0.9943	0.9870	0.9846	0.9898
Emulator (5)	0.9834	0.9874	0.9724	0.9660	0.9759
Snigel (1)	0.9728	0.9773	0.9569	0.9482	0.9620
Snigel (2)	0.9841	0.9874	0.9725	0.9665	0.9769
Snigel (3)	0.9798	0.9844	0.9665	0.9589	0.9707
Snigel (4)	0.9852	0.9882	0.9741	0.9676	0.9778
Snigel (5)	0.9408	0.9413	0.9483	0.9582	0.9489

Table D.3: Pearson product-moment correlation coefficient between the packet inter-arrival time CDF while playing offline music.