HELSINKI METROPOLIA UNIVERSITY OF APPLIED SCIENCES

Information Technology

Telecommunications

BACHELOR'S THESIS

AUTOMATION OF A QUALITY MEASUREMENT SYSTEM FOR WIRELESS @450 BROADBAND NETWORK

Author: Kimmo Kekkonen
Instructors: Ville Jääskeläinen
Mikko Huttunen

Approved: __.__. 2009

Ville Jääskeläinen
Principal Lecturer

Metropolia

PREFACE

The objective of this study was to implement an automated quality measurement system. The study turned out to be a challenging one and along the study many technical solutions where created giving me a motivation to go on.

I would like to thank Anne Suomi and Mikko Huttunen from Digita for this challenging and interesting subject. Also I would like to thank my supervisor Ville Jääskeläinen from Metropolia University of Applied Sciences, who gave me a hint of this subject. In addition Mr. Jääskeläinen and Marjatta Huhta gave me a lot of useful directions for writing this study.

Special thanks I would like to give to my wife and children for supporting me to get this study finished.

Helsinki, December 11, 2009

Kimmo Kekkonen

Metropolia

ABSTRACT

Name: Kimmo Kekkonen

Title: Automation of a Quality Measurement System for Wireless @450 Broadband Network

Date: December 11, 2009 **Number of pages:** 45 + 4 appendices

Degree Programme: Information Technology

Specialization: Telecommunications

Instructor: Ville Jääskeläinen, Principal Lecturer

Instructor: Mikko Huttunen, System Designer

The study implements an automated quality measurement system to be used in the wireless @450 broadband network. The main focus is to implement and test a system which can measure the quality of the customers Internet connection in a reliable manner. This study was carried out for Digita Oy as a part of a quality measurement project.

Digita had made a research how the quality measurement in the @450 broadband network could be carried out. Based on those findings a partly automated quality measurement system existed. The main elements were adopted from this existing system. To be able to automate a quality measurement system the Python programming language was used. The study shows how the each part of the quality measurement systems can be automatically controlled by the computer program.

The implementation of the automated quality measurement system was successful using the Python programming language. Python provided a solution for automation without of use of any other programming language. As a result the system fulfilled the issued requirements and was identified to be effective and reliable.

This study can be used by Digita Oy to implement an automated quality measurement system in their Flash-OFDM based network. The implementation offers a fully automated quality measurement system which reduces the work time spent on quality measurement by the employees and speeds up the processing of results.

Key words: @450, Quality Measurement, Python, FMDM, FMLP

Metropolia

OPINNÄYTETYÖN TIIVISTELMÄ

Työn tekijä: Kimmo Kekkonen

Työn nimi: Langattoman @450-laajakaistaverkon laadunmittausjärjestelmän automatisointi

Päivämäärä: 11.12.2009 **Sivumäärä:** 45 s. + 4 liitettä

Koulutusohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Tietoliikennetekniikka

Työn ohjaaja: Yliopettaja Ville Jääskeläinen

Työn ohjaaja: Järjestelmäsuunnittelija Mikko Huttunen

Tämän insinöörityön tarkoituksena oli toteuttaa automatisoitu laadunmittausjärjestelmä langatonta @450-laajakaistaverkkoa varten. Päätarkoitus oli toteuttaa ja testata laadunmittausjärjestelmä, joka pystyisi mittamaamaan asiakkaan Internet-yhteyden laadun luotettavasti. Työ tehtiin Digita Oy:lle osana laadunmittausprojektia.

Digita oli tehnyt tutkimuksen kuinka laadunmittaus voitaisiin toteuttaa @450-laajakaistaverkossa. Tutkimustulosten pohjalta oli toteutettu laadunmittauksen seurantajärjestelmä, joka oli osittain automatisoitu. Kyseisestä järjestelmästä hyödynnettiin tärkeimpiä osa-alueita uutta kokonaan automatisoitua laadunmittausjärjestelmää varten. Jotta laadunmittausjärjestelmän automatisointi voitaisiin tehdä, täytyi perehtyä Python nimiseen ohjelmointikieleen. Työssä on tuotu esille kuinka jokainen osa-alue laadunmittausjärjestelmästä voidaan toteuttaa automaattisesti. Automatisointi tapahtuu tietokoneohjelman avulla.

Insinöörityön edetessä osoittautui, että laadunmittausjärjestelmän automatisointi voidaan toteuttaa Python ohjelmointikielellä. Python mahdollisti ratkaisun, jossa ei ollut tarvetta käyttää muita ohjelmointikieliä. Tuloksena oli järjestelmä, joka täytti Digitan asettamat vaatimukset. Testeissä järjestelmä todettiin tehokkaaksi sekä luotettavaksi.

Digita voi hyödyntää tämän insinöörityön tuloksia toteuttaessaan automatisoitua laadunmittausjärjestelmää heidän Flash-OFDM pohjaista verkkoa varten. Työn tuloksena rakennettu järjestelmä on täysin automatisoitu, mikä vähentää työntekijöiden työajan tarvetta laadunmittauksessa sekä nopeuttaa tulosten käsittelyä.

Avainsanat: @450, laadunmittaus, Python, FMDM, FMLP

TABLE OF CONTENTS

LIST OF ACRONYMS

3G	Third Generation (Mobile Communication System)
ASCII	American Standard Code for Information Interchange
CLI	Command Line Interface
CMD	Command Prompt
CSV	Comma Separated Values
Flash-OFDM	Fast Low-latency Access with Seamless Handoff, Orthogonal Frequency Division Multiplexing
FMDM	Flarion Mobile Diagnostic Monitor
FMLP	Flarion Mobile Log Processor
FTP	File Transfer Protocol
GUI	Graphical User Interface
Iperf	Internet Performance Working Group
NMT	Nordic Mobile Telephony
MHz	Mega Hertz
Miperf	Modified Iperf
RF	Radio Frequency
Telnet	Telnet

1 INTRODUCTION

This study arises from the need to investigate and automate several operations which are needed to measure the quality of the wireless @450 broadband network. In order to implement an automated quality measurement process it is relevant to investigate how several programs and routine tasks can be controlled automatically without manual work. These operations will be represented in a general level to provide a clear and logical structure of the whole automation process.

Wireless @450 broadband network uses 450 MHz frequency previously used by the NMT (Nordic Mobile Telephony) network. *NMT* was the first cellular phone system for mobile phones used from year 1981 to year 2002. After 2002, when *NMT* was ramped down the 450 MHz frequency was released and within three years the frequency was allocated to wireless broadband network. Digita Oy won the tendering of the frequency in 2005 and was the only company from the seven candidates who had rights to build up a 450 MHz frequency network [1:1]. 2005 Digita started to maintain and develop the network, which was named as Wireless @450 broadband network. The technology behind @450 network is *Flash-OFDM*.

This study was carried out by Digita Oy, a Finnish wireless communication network operator, as part of a quality measurement project. Digita provides transport services for radio programs, television programs and wireless broadband in whole Finland [2]. In the wireless broadband area Digita develops and maintains wireless @450 broadband network. @450 network has been built by the terms of an open network model. In this model the role of Digita is to act as a network operator which provides capacity to all service operators. These service operators will then provide wireless broadband access to customers [3]. Digita has done quality measurements in their @450 network since the establishment of the network, but this process has been manual, and therefore errors may have occurred. Many of the steps needed in the quality measurement process have to be made by hand. These steps produce unnecessary load to employees and generates high operating expenditures. Because the tasks are mainly routine tasks, there is no need for advanced intelligence, they should be just automated to be handled by a computer program.

This study aims to offer an implementation to automate the quality measurement process in wireless @450 broadband network. Basically the quality measurement composes of four parts which are: *taking samples from the wireless connection*, *arranging the collected data in a logical way*, *transmitting the collected data to the processing server* and *processing the collected data at the server side*. Samples are taken using *FMDM* (Flarion Mobile Diagnostic Monitor) program. *FMDM* measures few *RF* (Radio Frequency) parameters to diagnose the quality of the wireless connection. Data is then further processed by using *FMLP* (Flarion Mobile Log Processor) at processing server. *FMLP* modifies the raw *FMDM* output data into reports and graphs. Reports and graphs can be then utilised to monitor the quality of the wireless connection.

In order to automate the measurement process it is relevant to program a script, which handles all the parts and tasks of the process. This is done by the use of a Python programming language. Python is an open source programming language, that can be used for different kinds of software development [4]. Because it is open source anyone can use it for free without license fees. Python has an ability to control programs that run on operating system eg. on Microsoft Windows [5] and to handle the Internet protocols such as *FTP* and *Telnet* [6]. *FTP* (File Transfer Protocol) is used to transmit collected data from the measurement device to the processing server. *FTP* has been developed to transfer data from a computer to an another [7:1]. *Telnet* is used to command *FMDM* program on Microsoft Windows operating system and it provides a bi-directional communication facility [8:1]. These are the major requirements of the programming language to implement the required automation process.

There is already a study made which concentrated on how the @450 broadband network works and what kind of quality measurement options were available. Therefore this study concentrates on the implementation and testing of the automated quality measurement process. However, some information is used from the previous research as a background. The quality measurement can be split into two types of measurements, which are *RF parameter* and *Performance measurements.* The aim of the quality measurement project is to implement a solution for both types of measurements. However, this study mainly concentrates on RF parameter measurement and therefore

performance measurement is not fully covered. The implementation of the automated quality measurement system is introduced so that also the second type of measurement, performance measurement, could be easily adopted into the existing system. The first process called RF parameter measurement is used to measure *Radio Frequency* parameters from the radio path. This quality measurement process begins when the customer reports of a problem in certain area. RF parameter measurement process helps to diagnose whether there are problems in the wireless connection or not. Finnish Ministry of Transport and Communications has introduced a service quality requirement for a provided Internet connection. Therefore also the second process called performance measurement must be taken into account while the automated system is developed [19: 30]. Basically the service quality requirement means that the customer should get an Internet connection capacity of which he or she pays for. Digita provides all the needed equipment to do the automation of the quality measurement.

Because quality measurement is a mandatory process and the current process loads too much employees it is relevant to automate the process. The aim of the quality measurement project is to deduct the work time spent on the quality measurement by the employees and to speed up the processing of the results. When the customer reports about a poor wireless connection Digita should have a way to diagnose the customers' connection. Since wireless @450 broadband network is quite new there have not been any implementations which offer fully automated quality measurement system. This study is needed to implement and test an automated process to handle quality measurement in @450 network.

The study is written in six sections. The second section covers background information of the Python programming language. The section gives the basic tools to understand the operation of Python and lists the needed add-ons to implement automated quality measurement. The section 3 introduces the equipments, environment and how quality measurement is done in @450 broadband network. Section 4 includes the implementation to automate the quality measurement process. Section 5 introduces the results of the implementation which are used to analyse the operation of the system. The main emphasis of this study is placed on implementation (Section 4) as it is the most challenging part of the project.

2 PYTHON PROGRAMMING LANGUAGE

This section describes the functionality of the Python programming language. The main features of the Python are derived to indicate how Python works. The installation of the Python will be covered in section three and section four concentrates on the implementation of the automated quality measurement process.

Python is, as mentioned in the introduction section, an open source programming language that can be used for different kinds of software development [4]. At the time of writing Python has its third version Python 3.0 available but the older version 2.5 was chosen because it supports the needed 3rd party modules that are not yet customized for the 3.0 version of Python. In order to automate the quality measurement process five elements, which are variables, strings, control structures, functions and modules are introduced. First four elements covers the use of Python and the final element includes the needed advanced modules to be used in the quality measurement system.

2.1 Variables

Variables are one kind of identifiers. Identifier indicates a name which is used to individualize information [9:5]. In practice we can say that the information, data, is stored to variables. Data is stored as characters or numbers. Variables are stored in the computer memory while program is run.

In Python there is no need to first define variables before they can be used. Variables are defined when the first value is given [9:6]. An example is as follows

```
# -*- coding: cp1252 -*-
# Python example program

number = 5
```

The first two rows are comments. These commented commands does not usually affect the program, they are used for purpose of keeping notes. But in this case the first row of the file defines the encoding used for the file. Cp1252 enables the use of *Non-ASCII* characters. If Scandinavian characters are used this row is mandatory. Otherwise the first row can be left out [9:3]. The third row shows how number 5 is stored to variable named num-

ber. This variable can then be called for instance with the help of print function as follows.

```
# -*- coding: cp1252 -*-
# Python example program

number = 5
print number
```

As a result we get number 5. It is good to notice that when variables are called quotation marks are not needed. In Python there is also no need to define whether the value is a number, character or string. Exception is if we want to convert number to character. The following example shows the procedure.

```
# -*- coding: cp1252 -*-
# Python example program

number = 5
str(number)
```

Str() function is used to convert numbers to characters. By calling str() function the converted value is returned as return value. This feature is used for instance when numbers and strings are concatenated. Because numbers and strings cannot be concatenated, numbers first need to be converted to strings. Then when numbers are string values they can be concatenated with other strings.

The second element which is strings will be covered next. Strings are needed to pass information from one part of the program to another. Strings are one of the fundamentals of programming languages.

2.2 Strings

Strings are characters in a sequence. Strings can include what ever characters, so basically they can be said to be just words. Dealing with strings is very common in programming. For instance in this study strings are used every now and then.

Defining Strings

Strings can be defined in three ways. The first way is to use single quotes (‘). The second way is to use double quotes. The string is written between quotes. As an example lets use (“) double quotation marks.

“Example string”

There is no difference between the first and the second way of defining strings. Despite the fact that these expressions are the same in Python they can not be mixed. For example “Example string’ returns an error.

The third way to define string is to use triple single quotes (‘’’) or triple double quotes (“””). In this case the string is written in multiple rows as follows.

‘’’Example string which is written in two rows. This is the first and this is the second row’’’

Escape Sequence

Control characters are needed when there is a need to use special characters. These are for instance back slash (\), quotation mark (“) and new line (\n). In order to use these special characters the control character must be put in front [9:18]. The following example demonstrates the usage.

’1. row: \”Quotation mark\” \n 2: row. Back slash is as follows \\’

If we would print the typed string we would get the following:

1. row: “Quotation mark”
2. row: Back slash is as follows \

Concatenation

Concatenation is useful when there is a need to combine several strings to one string. Strings can be combined by using ‘+’ operator, which does the concatenation. This operation is needed when variables are inside the strings. For instance if we have a string as follows.

“Text can be written in” + str(*number)* + “ways.”

The *number* variable must be separated from the string. This is done by using single quotes or quotation marks in both sides of the variable. Here it is good to notice that we need str() operator like explained previously in order to combine characters and numbers. If the number has the value of 3 we would get the following print result.

Text can be written in 3 ways.

To add more intelligence to the program, there is a need for decision making. In this thesis there is a wide need to make decisions and comparisons and therefore control structures will be discussed next.

2.3 Control Structures

To make more comprehensive program there is a need for control structures to do more than execute commands one after another. These control structures are used to make comparisons and repetitions. Python includes the three basic control structures which are *if*, *for* and *while*. The first one, *if*, is used to make comparisons. Next two, *for* and *while*, are used to repeat something in a loop. This study covers if, for and while, because these control structures are used in the automated quality measurement process.

If Structure

The *if* structure is used to make comparisons as earlier mentioned. These are now detailed in the following manner. Comparison is made using an *if-else* structure. If the first condition is obtained then the program will do what *if* item defines. If the first condition is not obtained then the program will do what the *else* item defines.

A good example is a number checker. This kind of program checks if a number is greater than 5. The program would be as follows:

```
# -*- coding: cp1252 -*-
# Python example program

number = 6

if number > 5:
    print "Number is greater than 5"
else:
    print "Number is equal or less than 5"

print "Program completed successfully"
```

The number variable gets the value of 6. Then the program executes the *if* comparison. Now when 6 is greater than 5 the *if* item will be printed out. *If* item is the next intended row after colon. The last row “Program completed successfully” will also be printed because it is outside of the if-else structure.

While Structure

While structure allows to make repetition loops in Python. The while loop is performed until the given condition is true [10:26]. It is used in the following way:

```
i = 1
while i < 3:
    print i
    i = i + 1
print "Loop completed successfully"
```

From the example we can obtain that i receives the value of one. Then while loop is performed while i is less than three. As a result the loop would be performed two times and then exited.

For Structure

The *for* structure is an other option to make repetition structures in Python. For structure can be used to go through certain part of the program until the issued criteria is obtained. For loop has a requirement that the number of loops must be expressed as a constant value [9:39].

The following example introduces *for* structure:

```
for i in range(1, 5):
    print i
print "Loop completed successfully"
```

In the first row the *range* function creates a sequence of numbers [1, 2, 3, 4]. This means that when for loop goes through the created sequence of numbers the loop is repeated four times. At first time *i* has a value of 1 and number 1 is printed. Then for structure increases value of *i* by one to value of 2. Next for loop starts from the beginning. Now *i* has a value of 2 instead of 1. This procedure is repeated until the last number in the sequence of number is handled.

To implement a program which is simple, logical and can be flexibly changed there is a need for functions. In this study there is a need to use function to keep the program simple and clear. The next part of the second section introduces how the functions work.

2.4 Functions

Functions are a peace of program, which means that they are basically some rows of code. They work as a block of statements which are named for

instance according to what they do. This allows functions to be called by name and to be used as many times as programmer wants. Functions are also used in this thesis to make the program of the quality measurement more simpler and flexible [10:32].

Defining and Using Functions

Definitions are located in the beginning of the program. Once they are introduced they can be used afterwards. The following example demonstrates the usage:

```
def sayHello():
    print 'Hello!' # block inside the function
# End of function

sayHello() # call the function
```

Functions are defined using the keyword *def*. After that comes an identifier, the name of the function. Note that the name of the function must be followed by the parenthesis. After parenthesis comes colon which tells to compiler that next indented rows belong to the function. After the definition of the function comes the function calling. Function is called simply by the name of the function. Now when the function is called program will go through statements inside the sayHello() function and print “Hello”.

Parameters

Parameters are values which are sent to the function. They are sent when the function is called. The benefit of using parameters is that the function can then utilise those sent parameters for own internal use [10:32-33]. The following example shows how this works.

```
def printMax(a, b):
    if a > b:
        print a, 'is maximum'
    else:
        print b, 'is maximum'

printMax(3, 4) # directly give values
```

The first statement defines a function named *printMax*. It can have two parameters, which are internally used by using variable names *a* and *b*. Next four rows of statements belong to the *printMax* function. After the definition comes the function calling. Now when the *printMax* function is called also two parameters, 3 and 4, are sent. The function then stores these values as

internal values 3 as *a* and 4 as *b* and makes the if comparison. The output would be as follows:

```
4 is maximum
```

The other way is to send parameters with the help of variables. The only difference would be then that it will allow more wider and flexible use.

Global Variables

As a default variables are local, which means that they can not be used in other function as they are. But if there is a need to use so called global variables, like in this thesis, Python offers a solution for that. Global variables are variables which can be used all over the Python program. The following example shows how this is done.

```
def func():
    global x

    print 'x is', x
    x = 2
    print 'Changed global x to', x

x = 50
func()
print 'Value of x is', x
```

In a logical order *x* gets the value of 50. Then the *func* function is called. Now this local *x* variable of *func* function is declared to be a global variable. Then the value of x is printed and it is 50. Next *x* gets new value of 2. After finishing function on the last row of the program *x* is printed and it has the value of 2, even if it is outside the *func* function The output would be as follows:

```
x is 50
Changed global x to 2
Value of x is 2
```

When using global variables programmer must be aware of that they are unique in the whole program and complications does not occur.

Return Values

To achieve more flexibility and error handling there is a need for return values. If we have to make decision which depends on how function worked we can make use of return values. The following example illustrates the situation.

```
def func():
```

```
i = 1

if i == 1:    # eg. something went ok
    return 0
else:
    return 1  # eg. something went wrong
```

If the function *func()* succeeds to do what we wanted it would return for instance 0 and we can continue executing program. But if it would return 1 meaning something went wrong we would like act in differently.

Four previous parts of the second section introduced how to use Python for different kinds of operations. The final part of the section will introduce the modules which are needed for advanced use of Python. With the help of next introduced modules it is possible to automate the quality measurement system.

2.5 Modules

Now that the idea of function is covered it is easy to understand modules. Modules are files containing functions and variables which can be used by any Python program. Modules can be imported by another Python program to make use of its functionality [10:41]. This is a central feature in this thesis. Because modules include small programs there is no need to do everything from scratch, which saves a lot of time and thinking. First we take a look at how modules are imported to the program. Then we will go through the most important modules needed for automating the quality measurement process.

Importing Modules

A module is imported using an *import* statement. This needs to be done in the beginning of the program in order program to know what modules are included. Import is done as follows:

```
# -*- coding: cp1252 -*-
# Python example program

import time

time.sleep(10)
```

The first line shows how to import a module. Firstly *import* statement is written and right after that the name of the imported module. Now we have all the functions and variables of *time* module in use. Secondly we can call the functions inside the *time* module. Lets take *sleep* as an example function. *Sleep* is also commonly used in this thesis. First we write the name of the

module then separate the function name by dot and finally give the parameter of 10: *time.sleep(10)*. When the program is run it just waits for 10 seconds before its finished.

Datetime

Datetime module is a module which can handle basic date and time types [11: 91]. In this thesis two of the *datetime* module functions are used. First is the *date.today()* function, which returns the current local date [11: 94], for instance *2009-06-17*. Second is the *timedelta()* function, which represents a duration, the difference between two dates or times [11: 93].

Time

Time module provides many time-related functions [11: 394]. In this study the function *sleep()* of *time* module is used. Python documentation defines *sleep* function as follows

> Suspend execution for the given number of seconds. The argument may be a floating point number to indicate a more precise sleep time. The actual suspension time may be less than that requested because any caught signal will terminate the sleep() following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount because of the scheduling of other activity in the system [11: 396]

Sleep function takes seconds as parameters, which determine how long the execution is suspended.

Os

Os module provides a portable way of using operating system dependent functionalities [11: 375]. In this study five functions of the *os* module was used. These functions are *chdir(), makedirs(), listdir(), path.join(), path.isdir().*

The first function, *chdir(),* is defined according to the Python documentation as follows: "Change the current working directory to path." [11: 382] This function is needed when changing the current working directory to other path.

The second function, *makedirs()*, is defined as follows: "Recursive directory creation function. Like mkdir(), but makes all intermediate-level directories

needed to contain the leaf directory. Throws an error exception if the leaf directory already exists or cannot be created.” [11: 383] In order to make folders *makedirs* function is needed. Making directories is a fundamental operation in this study, because the collected data must be in order. This function is also useful because it makes all intermediate-level directories.

Listdir(), the third function, is defined as follows: “Return a list containing the names of the entries in the directory. The list is in arbitrary order.” [11: 383] This function is used when there is a need to find out what directories and and files the current directory includes.

Fourth function, *path.join()*, is defined as follows: ”Join one or more path components intelligently. If any component is an absolute path, all previous components (on Windows, including the previous drive letter, if there was one) are thrown away, and joining continues.” [11: 315] As a result the return value is the concatenation of sent paths, for instance path1 and path2. To concatenate paths and filenames to get absolute paths *path.join()* function is used. Using this function it reduces the error probability by automatically placing correctly slashes etc.

Last function, *path.isdir()*, is used in order to get information if a directory already exists. This function returns *True* value if path is an existing directory [11: 314]. To avoid overwriting files *path.isdir()* is used. If directory exists it will be left as it is and program continues execution.

Subprocess

Subprocess module allows new processes to be run and connecting to their input, output and error streams [11: 543]. However in this study *subprocess* module is used only to start new processes, for instance running programs on Windows operating system.

To run external programs a function called *Popen()* in *subprocess* module needs to be executed. *Popen()* is called as follows:

```
subprocess.Popen(arg, stdin=None, stdout=None)
```

The program name is typed as a first argument, in this case by replacing *args* by the executable file of the program. *Stdin* and *stdout* have the *None* value in this study because there is no need to have pipes to and from the

program. With the help of pipes it would be possible to for instance record error messages and saving them as a log file.

Telnetlib

The *telnetlib* module provides a class that implements the Telnet protocol [11: 623]. In this study three functions are used. These are *Telnet(), write()* and *close()*.

The first function *Telnet()* is used to open a Telnet connection to server [11:623]. It is used as follows:

```
telnetlib.Telnet(host, port)
```

Where the host is replaced by the host machine's address and the port by the host machine's port.

The second function *write()* is used to write string to the host machine [11: 624]. *Write* function is used in the following way:

```
telnetlib.Write(buffer)
```

Buffer is replaced by the string wanted to send to the host machine.

The last function *close()* is used to close the Telnet connection. It does not take any arguments. Function is called simply by writing *telnetlib.close().*

Ftplib

The *ftplib* module implements the client side of the FTP protocol. This can be used to write Python programs that need to perform automated FTP tasks [11: 604]. In this study there is a need for *FTP()*, *cwd(), mkd(), storbinary()* and *quit()* functions. The first function *FTP()* is needed to open a FTP session. It is used in the following way:

```
ftplib.FTP(host, username, password)
```

Host is the address of the host machine, *username* is the used FTP username on the server and *password* the FTP users password.

The second function *cwd()* is used to set the current directory on the server [11: 607]. It is used as follows:

```
ftplib.cwd(pathname)
```

Where the *path* is the wanted path on the server. This is used to move from directory to another on the server.

The third function *mkd()* is used to make a new directory on the server side [11:607]. It is used in the following way:

```
ftplib.mkd(pathname)
```

Where the *path* is the name of the directory wanted to be made on the server.

The fourth function *storbinary()* is used to transfer files from local, client side, computer to the server. This function stores a file in binary transfer mode [11: 606]. It is used in the following way:

```
ftplib.storbinary(command, file)
```

Command have to be an appropriate FTP transfer command. Available commands are *STOR* and *APPE. STOR* rewrites the file and *APPE* skips the file if it already exists. *File* is the path and name of the file wanted to be transferred. File must be opened before it can be transferred, this is discussed later in this section.

The final function *quit()* is used to close the FTP connection [11: 607]. It sends a “quit” command to the server and connection is closed. Function is called simply by writing *ftplib.quit()*.

Socket

The socket module in this study is used to find out what is the computers current IP address. Two functions of the module are used which are *socket.gethostbyname()* and *socket.gethostname()*.

```
socket.gethostbyname(socket.gethostname())
```

The command retrieves the current IP address of the system.

Numpy

The module called *Numpy* is the fundamental package for Python to make scientific calculations. In this study there is a need to use Numerical Python as a one step to plot diagrams. The only functions needed is *arange()*. NumPy documentations defines it as follows: “Return evenly spaced values within a given interval.” [20]

PyWin32

PyWin32 is a package that includes extensions to access Windows Operating Systems functionalities. These functionalities include modules such as *win32com, win32api, win32qui*. These modules are also used in this study to achieve an programming interface to make certain tasks within Windows XP [21]. Useful tasks are for instance sending characters to programs and closing application windows. *PyWin32* was previously known as *Win32all*.

Matplotlib

The *matplotlib* module allows to plot graphs and figures using Python. Official homepage defines matplotlib as follows: ”matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hard-copy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala matlab or mathematica), web application servers, and six graphical user interface toolkits.” [22].

In this study *matplotlib* is used to plot graphs from the processed information of the radio path. It allows wide use and modification of figures but we are concentrating only on basics.

Logging

The *logging* module defines functionalities which implements flexible logging system for Python programmed applications [23]. In case of errors and malfunction it is mandatory to have logging feature. From the log files it easy to follow program execution and trace the error situation.

The introduced functionalities allow a wide use of Python programming language. However there are a lot of more functionalities that Python can do. Therefore Python has gained popularity among software developers and administrators. Wide use of Python also makes it relevant to use for this study where many functionalities are needed. Next section will introduce the quality measurement environment including what is needed before scripted program which takes care of the automation of the quality measurement can be implemented.

3 MEASUREMENT ENVIRONMENT AND MEHTOD

This section covers the hardware and the preparations needed to do quality measurements in @450 broadband network. All required devices and software installations are introduced. The aim is to set up the environment so that the automated quality measurement can be performed.

As described earlier in the introduction section there are two kinds of measurement arrangements *RF parameter measurement* and *performance measurement*. The automation of the quality measurement process is different in RF parameter measurement than in performance measurement. The RF parameter measurement process is fully automated to operate days or months, whereas performance measurement is manually started and ends after 24 hours of measurements. The quality measurement time of the performance measurement is set by the Finnish Ministry of Transport and Communications [19:30]. However the network arrangements are the same in both processes. The only difference is in the scripts used for automating the process.

In order to make automation process to work there is a need for two devices. The first device is the measurement device and the second the processing server. The network arrangement is described next, then the measurement device and finally the processing server.

3.1 Network and Measurement Arrangement

Customers are connected to the @450 broadband network by @450 Terminal. The Terminal forms a wireless connection to the Radio Router which are then connected to the Internet cloud. Measurement device works in the same way and therefore quality measurement must take place similarly to the customer connection. This arrangement is illustrated in figure 1.

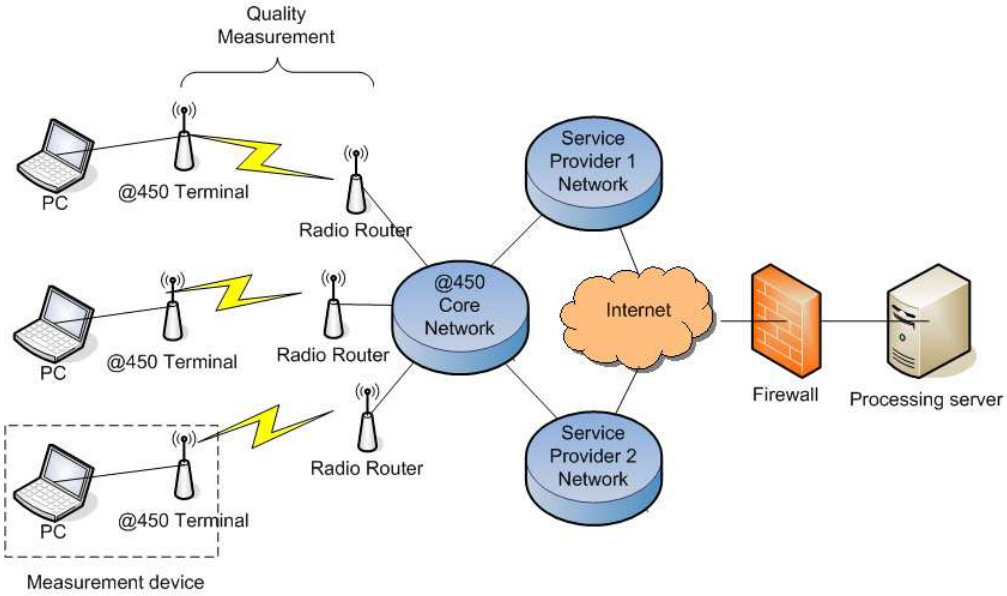


Figure 1 The quality measurement network diagram

The quality measurement occurs between the @450 Terminal and the Radio Router. The measurement device measures the radio path to the Radio Router. Once the measurements have been completed, the device transfers the collected data to the processing server through the network. The data is transmitted through @450 core network and through the service provider network, in this case to Digitas own network and through Internet to the processing server. Processing server is located at the Digitas laboratory behind a firewall. Server then processes the collected data of the radio path into a usable format.

In the case of the performance measurement the scripts are run manually in both of the devices, in the measurement device and in the processing server. These scripts then automate the actual performance measurement process. Once the measurements have been done, the script in the processing server has to be stopped manually.

3.2 Measurement Device

Measurement device is used to make measurements from the wireless connection. The device consists of a computer in which there is a suitable @450 broadband terminal connected. Also the computer must have an appropriate air testing program installed. In this study Acer Aspire one P531h-06k mini-PC was used and it was connected to Flarion Wireless Broadband Terminal. The air testing program installed in the PC was Qualcomm's FMDM. In order

to do not only RF parameter measurement but also performance measurement a program called Miperf was installed.

3.2.1 Features

In order to decide what kind of PC to use for measurements two requirements must be met. The first one is that the computer meets the requirements released by Qualcomm to use FMDM. Compatibility requirements for using FMDM are as follows [12, 2-1]: *Windows 2000® or Windows XP®*, *Pentium IV 1.2 GHz or above* and *512 MB of RAM memory*. The second criteria is that the computer should be small enough to fit into a small box. This box can be used to include all parts of the measurement device.
The computer was chosen according to the previous two requirements and the result was as mentioned before, Acer Aspire one P531h-06k mini-PC, which is illustrated in figure 2.

Figure 1 Acer Aspire One 531h-06k

Acer computer has features as follows: *Windows XP Professional Edition*, *Intel® Atom™ processor N270 1.6 GHz* and *2048 MB of DDR2* memory. These features meet the requirements for using FMDM. PC is also small enough to fit into a box which would include all parts of the measurement device.

The computer must be connected to Flash-OFDM wireless terminal to be able to connect @450 broadband network. For this purpose Flarion Wireless Broadband Terminal was used. Terminal is shown in figure 3.

Figure 2 Flarion wireless broadband terminal

Flarion Wireless Broadband Terminal includes an antenna, Ethernet port and power input port. Antenna can be a small indoor antenna like in figure 2 but it can also be a bigger rake antenna. Ethernet port is used to connect terminal into the measurement device.

3.2.2 Initialization

In order the measurement device to work properly few folders have to be made. For RF parameter measurement the following folders have to be created.

- C:\FMDM\Logs
- C:\FMDM\Measurements

The first folder is used to store logging files and the second folder to store measurement information. The script which automates the operations is copied to C:\FMDM folder. The script is named as measure_script_lab.py and it is included in appendix B.

The measurement device is adjusted to do measurements between 1.00 and 23.00 a clock. This was done using Windows own Scheduled Tasks which is shown in figure 4.

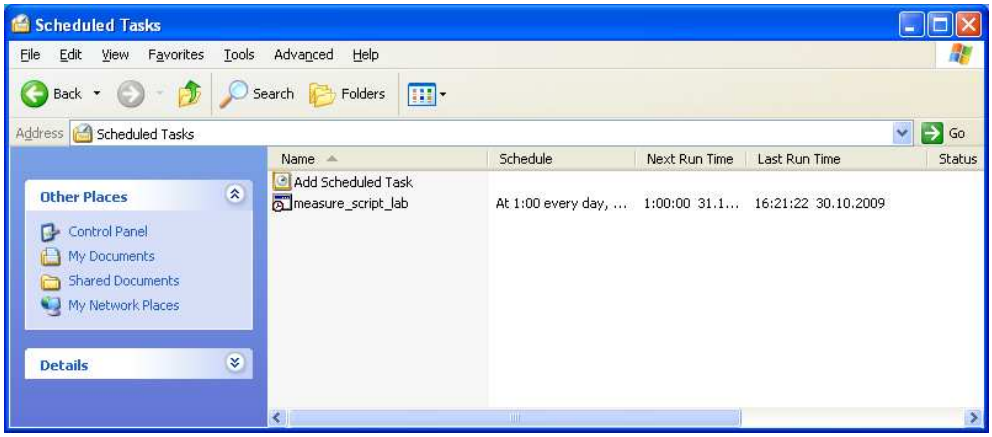


Figure 3 Windows scheduled tasks

The name of the script appears in the window and information about when it is scheduled to be run, when is the next run time and what was the last run time. Scheduled task starts the measurement script at 1.00 a clock automatically. Once the script have been started it does all the operations which are programmed into it. Finally at 23.00 a clock the script ends the measurement process and does few closure operations, which are introduced in section 4. After that the computer waits next morning to start the measurements. This will happen next day at 1.00 a clock.

Performance measurement

For performance measurement the following folders have to be made.

- C:\Miperf\Logs
- C:\Miperf\Measurements

Like in the RF parameter measurement in performance measurement the folders have the same purpose. The first folder is used to store logging files and the second folder to store measurement information. The script which automates the performance measurement is copied to C:\FMDM folder. In addition the script which generates the traffic was stored to the same folder. This had to be done this way because within the measurement script generation of the traffic did not work.

3.2.3 Installation of FMDM

The second step was to install appropriate programs into the measurement device. At first the air testing program FMDM version 3.3.1, which is illus-

trated in figure 5 was installed using Windows installer. After successful installation the USB serial key called USB dongle was plugged in and FMDM was started. It is important to notice that FMDM won't start without the USB dongle.

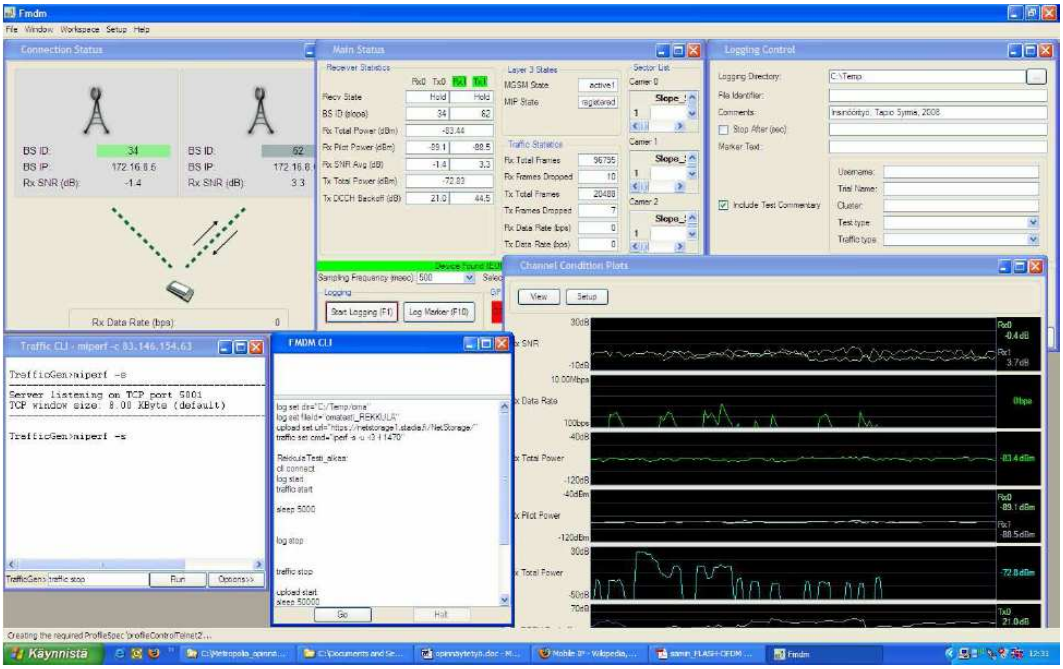


Figure 4 FMDM program [27:19]

After installation FMDM needed few minor changes. Firstly the correct windows were opened. These were Main Status, Channel Condition Plots, Logging Control and Connection Status windows. Secondly at installation path two filenames were renamed. File named fmdmConfig was renamed as fmdmConfig-FRR3.0 and fmdmConfig-FRR2.0 was renamed as fmdmConfig. Thirdly the previously renamed fmdmConfig file was loaded as FMDM Model. This was done from the Workspace > Load FMDM Model File then fmdmConfig was chosen. And finally the wanted workspace was saved as follows: Workspace > Save workspace file > _workspaceAuto_

3.2.4 Installation of Miperf

Miperf installation was a simple process. The Miperf program file called miperf.exe was copied to the C:\Miperf folder. Miperf was used so that it operated in a server mode. This means that the processing server generates traffic to the measurement device. Miperf has its own script which is called within the measurement device script. The command to use Miperf was as follows:

```
C:\Miperf\miperf.exe -s -p 65000 -u -l 1440
```

The commands –s parameter tells Miperf to work in a server mode. The second parameter –p indicates the port used for traffic. The third parameter –u is used to generate UDP traffic. The last parameter –l indicates the window size of 1440 bytes.

3.2.5 Installation of Python and Modules

The final step of the preparation was to install Python programming language and the appropriate modules. Python version 2.5.4 was downloaded at Python official website and installed using the Windows installer [14]. After installation of Python the required modules were installed. The only modules which did not come with the standard library was PyWin32. PyWin32 module was downloaded at the website of the SourceForge [15]. Filename of the appropriate version was pywin32-214.win32-py2.5.exe. After the module was downloaded it was installed using the Windows installer.

3.3 Processing Server

The second device is the processing server. The processing server has two major functions. Firstly it works as a FTP server, where measurement device transfers all the collected data. Secondly the server processes the data into graphs and useful values. At pretesting phase another Acer Aspire one P531h-06k was used as a server. In order computer to work as a FTP server a program called FileZilla was installed. For processing Qualcomm's own log processing program FMLP was used. In order to do not only RF parameter measurement but also performance measurement a program called *Iperf* was installed.

3.3.1 Features

FMLP is available only for Microsoft® Windows Operating Systems such as Windows XP[13, 9]. It is the only requirement set by Qualcomm for using FMLP, but according to tests computer should also have enough memory and processor power to run FMLP. At least 1024 MB of memory and 1.2 GHz processor turned out to be enough. Acer Aspire one P531h-06k meets these requirements. Acer's features were introduced in chapter 3.1.1.

3.3.2 Initialization

In order processing server to work properly few folders have to be created. For RF parameter measurement they are as follows.

- C:\FMLP_v2.4.5-2 \CSV
- C:\FMLP_v2.4.5-2 \Graphs
- C:\FMLP_v2.4.5-2 \Logs

The first folder will include *CSV* (Comma Separated Values) data which is generated by a specific script. The script collects five *RF* parameters within 30 days time period and combines them into one file. These parameters are *active_rx_pilot_pwr, active_rx_snr_avg, active_tx_dcch_backoff, rx_rate_kbps and tx_rate_kbps*. The second folder is used to store daily plotted graphics of the radio path. The third folder is used by processing server to store logging files of the RF parameter measurement.

Also the processing server scripts have to be copied into correct paths. For RF parameter measurement two scripts fmlp_script_lab.py, which is included in appendix C and handle_data_lab.py, which is included in appendix D are copied into C:\FMLP_v2.4.5-2 folder. The first script fmlp_script_lab.py automates the FMLP which is used for processing the collected data. The second script handle_data_lab.py automates the collected data further treatment which produces graphs.

The scripts have to be scheduled also correctly. This is done from the Windows Scheduled Tasks where the two scripts are adjusted. The first one is the fmlp script and the second is the data handling script.

Performance measurement

For performance measurement the following folder needs to be created

- C:\Iperf\Logs

The folder is used by processing server to store performance measurement logging files into it. The iperf_client_script_lab.py script which automates the performance measurement is copied into C:\Iperf folder.

3.3.3 Installation of FMLP

FMLP is a compiled MATLAB script and therefore it needs some MATLAB component libraries to be installed [13, 9]. To install a MATLAB library two steps were needed. Firstly the files were copied to *C:\FMLP_v2.4.5-2* directory and secondly the file named *MCRInstaller.exe* was run and the installation was done with the default settings

After these steps it was possible to install FMLP itself. Installation was simple consisting only few steps. Firstly the Zip file was extracted to *C:\FMLP_v2.4.5-2* directory and secondly the file named *FMLP.exe* was run.

The first time FMLP.exe was run it took some time to start FMLP. This was because FMLP unpacked its contents. FMLP DOS window appeared and described the extraction of CTF archive. After the extractions was completed FMLP main window appeared: FMLP was now installed successfully.

3.3.4 Installation of Iperf

Iperf installation is done similarly to Miperf. The Iperf program file called iperf.exe is copied to the C:\Iperf folder. Iperf works in the client mode to generate traffic to measurement device. The command to use Iperf is as follows:

```
C:\Iperf\iperf.exe -c 192.168.50.6 -p 65000 -u -b
1024k -l 1440 -t 3600
```

The commands –c parameter tells Iperf to work in a client mode. The second parameter –p indicates the port used for traffic. The third parameter –u is used to generate UDP traffic. Parameter –b indicates the bandwidth used for traffic. Parameter –l tells Iperf what window size to use. The last parameter –t indicates the time how long traffic is generated at a time.

3.3.5 Installation of FileZilla FTP Server

FileZilla FTP server was downloaded at FileZillas own website and it was installed with the basic windows installer [18]. Installation settings were left as default. After a successful installation FileZilla settings had to be adjusted correctly.

User settings

The first step was to create a username and a password for the measurement device to be able to log in. This username and password is used by measurement device to transfer collected information of the radio path to the processing server. Secondly few general settings were changed. The maximum number of users was not limited to allow as many connections as possible. Therefore several measurement devices can be connected simultaneously to the processing server. All timeout settings were set to zero meaning there is no timeout. This allows measurement devices to be connected for long periods of time without data transfer. This is needed if the radio path is poor and data hardly goes through. Otherwise measurement device would not be able to transfer collected information.

Server side settings

Three settings were changed in the FTP server. Firstly the server was set up to be in a passive mode. In passive mode server opens a port where the client i.e. measurement device can connect [24]. The server waits until the client connects and starts the file transfer. The server was set to use its own external IP address and custom port from range of 50 000 – 50 100 from the passive mode settings. Previous port range was preferred because those port numbers are not reserved by other protocols. Secondly the IP filter was set to disallow all IP addresses except the IP addresses of measurement devices. Therefore all other connections are refused. This setting was considered as a mandatory setting, otherwise there were tens of tries within a week by hackers trying to guess usernames and passwords to get into the server. Thirdly the logging feature was turned on. The server was configured to save logging files for 180 days and to limit the size of the logging file to 1000 kB.

3.3.6 Installation of Python and Modules

Installations of Python and PyWin32 module were made in the same way as in the measurement device. However the processing server needed few external modules more. These modules were Numpy and MatplotLib. Numpy was downloaded at SourceForge [16]. As described in section two Numpy was needed to plot diagrams. MatplotLib was also downloaded at the website of the SourceForge and the file name was matplotlib-0.98.5.3.win32-py2.5.exe. As mentioned in section two MatpotLib was used to plot graphs and figures using Python.

3.3.7 Setting Up the Firewall

The last preparation task was to set up the servers firewall. Because the collected data does not contain any classified information it was decided to use the Windows own firewall.

As the FTP server was set to passive mode and port range was 50 000 to 50 100 there was a need to configure the firewall accordingly. From the Windows firewall settings all ports from 50 000 to 50 100 were opened. However there is no feature where you can put port ranges. Therefore all 100 port openings would have to be made manually unless there is a script. A script was made so that it opens ports one by one in a loop. The command was put in a Command Prompt and was as follows

```
for /L %i in (50000,1,50100) do netsh firewall add
portopening TCP %i "Port-range %i" [25]
```

The script goes through the port range and sends a command to open the corresponding port in the firewall. The FTP service must also be enabled from the Windows firewall settings. Otherwise FTP connections from the measurement device are refused by the firewall. FTP service is enabled as follows: Advanced > Local Area Connections > Settings > Services > FTP service > input servers IP address

After the servers IP address is put into the field the server allows FTP connections to be established.

Once all the preparations and installations are done the devices can be automated. The next section, section four, introduces the implementations of the scripts. These scripts are needed to automate the quality measurement system in @450 broadband network.

4 AUTOMATION OF QUALITY MEASUREMENT

This section describes the automation of the quality measurement process. The aim is to offer an automated system, which does all the tasks from the beginning to the end ie. from the measurements to the data processing. As the scope of this study was only in the RF parameter measurement, only the implementation which automates the quality measurement of the RF parameter measurement is introduced. To implement this kind of an automated

system Python programming language was used. The devices are firstly described in general to get a general view of the features. Then each of the functionalities are described in more detail.

The quality measurement system consists of two devices. The first one is the measurement device and the second is the processing server. The functionalities of these devices are represented in flow charts to offer more general view without interpreting the source code in detail. In case there is a need for a closer look at the Python language the source codes are included in the appendix B, C and D. The implemented Python scripts are run locally in the devices.

4.1 Measurement Device

The script which controls the measurement devices consists of five main operations. These operations are initialization, preparation, measuring, transfer and closure. This general functionality of measurement device is represented in figure 6.

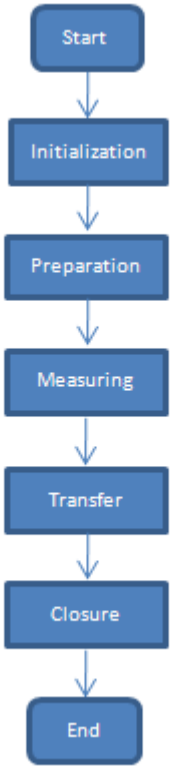


Figure 5 General functionality of the measurement device

When the script is started it firstly makes an initialization. In the initialization process, for instance, global variables and logging are introduced. After that the script starts the preparation process which takes care of the preparations needed to do the measurements. Then the measuring process starts to measure the radio path. When all the measurements are done the script calls the transfer process to transfer all the collected data to the processing server. Once the all files have been transmitted the script calls closure process which closes the measurement software.

4.1.1 Initialization

Initialization process introduces the fundamental variables needed in the program and enables logging. The process is illustrated in figure 7.

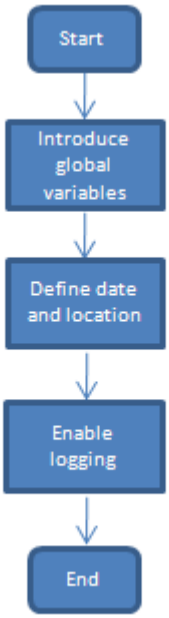


Figure 6 Initialization process

When the initialization process starts it firstly introduces global variables. Then it defines the current date and stores it to a variable. Location variable is manually set by the user. The value of the location variable is the location where measurement device is located. Finally the process enables logging and ends.

4.1.2 Preparation

The preparation process takes care of the software startups and of the correct data organization. The process is shown in figure 8.

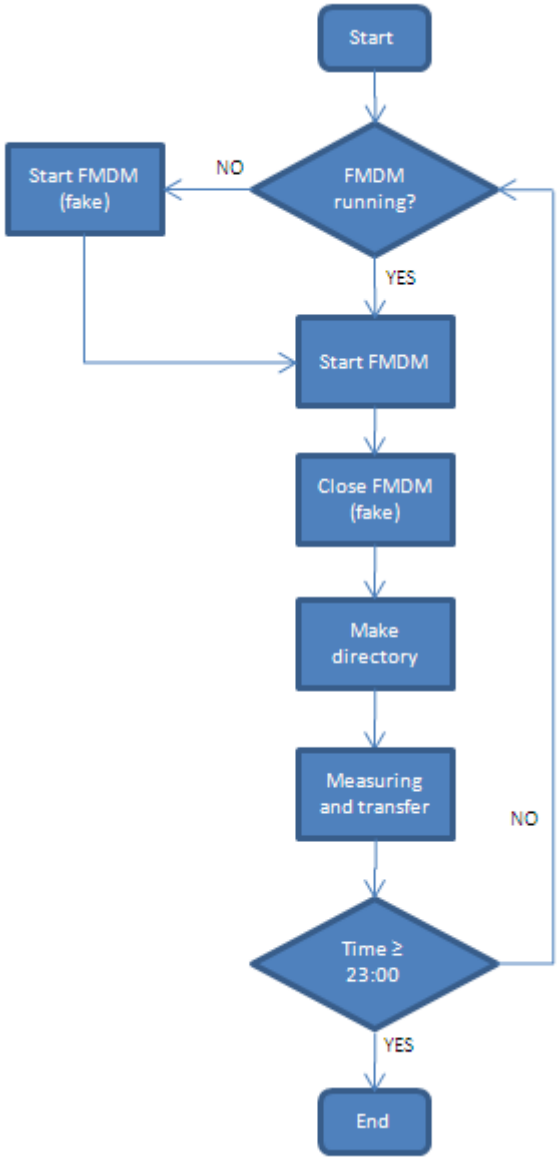


Figure 7 Preparation process

Preparation process starts by checking whether FMDM is running or not. Here takes place a solution that was invented after months of work. After few months when FMDM was installed it started to slow down. The startup took hours. However, this was solved by starting another FMDM and closing the first one. With this solution FMDM was in operation within a minute. After FMDM was operating correctly a directory according to current date is made. When the directory is made the script does the measurements and the file transfer which are explained next. If the measurements or transfer interrupts before the time is 23:00 preparation process starts the process again.

4.1.3 Measuring

The measuring process enables FMDM to log traffic in the air interface. It establishes a Telnet connection to FMDM, runs the FMDM scripts and shuts the Telnet connection. The operation is illustrated in figure 9.

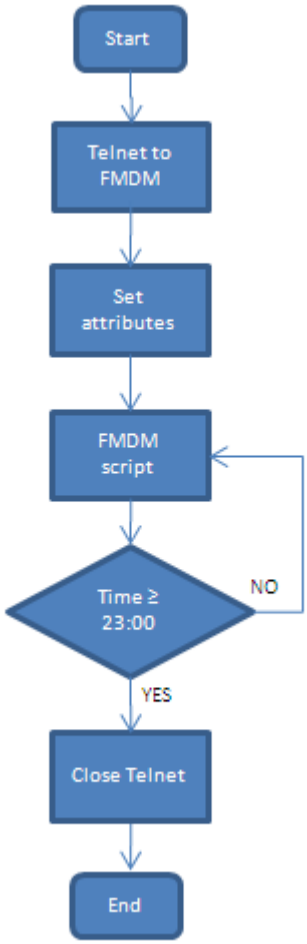


Figure 8 Measuring process

At first a local Telnet connection to FMDM is established. Once this has been done the script sets two attributes needed by FMDM. After the attributes have been set starts the measuring process. The FMDM measurement script is run for 30 seconds and then there is a halt period, which lasts for 5 minutes. So the total time for one sample data is 5 minutes and 30 seconds. When the time is more than 23:00 the scripts closes the Telnet connection and starts the transfer process.

4.1.4 Transfer

The transfer process takes care of the data transmission. It transfers all the collected data of the current date before midnight to the processing server. The operation of transfer process is illustrated in figure 10.

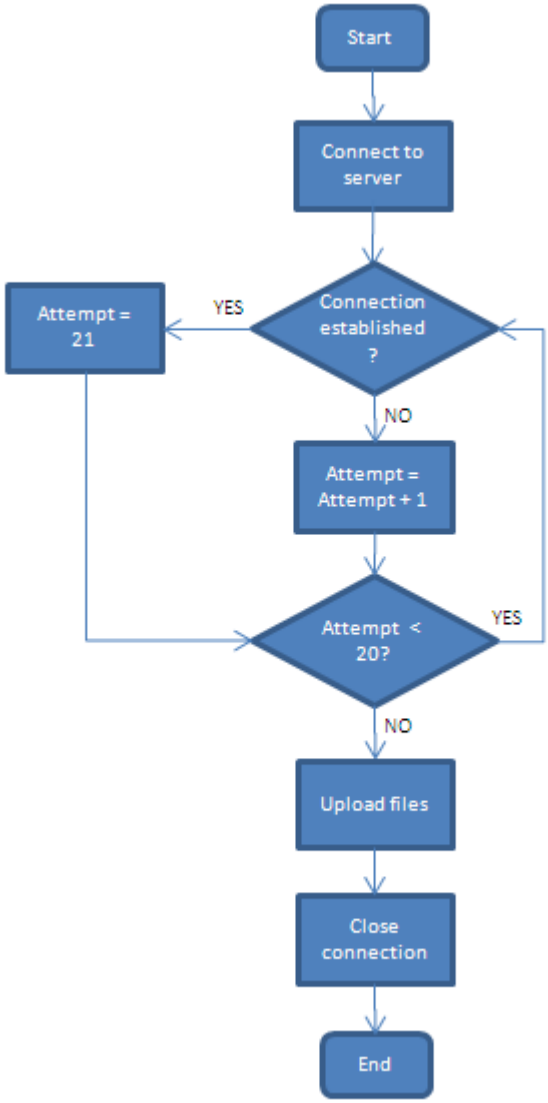


Figure 9 Transfer process

The first task of the transfer process is to connect the measurement device to the processing server where FTP server is installed. If the first attempt does not succeed it will try to connect again. If after 20 tries there is no connection, the file uploading is considered failed and the script terminates. If the measurement device was able to connect the file upload will start. File upload is made by a modified FTP client, which is based on a code published by Mark Lutz [26:602-603]. File upload goes through all the files in the

folder where the measured data is located and transfers them one by one to the processing server into corresponding folder. After all files have been transferred the close connection function is called to close the FTP connection and the script moves to closure process.

4.1.5 Closure

The closure process terminates the FMDM program process. The operation is represented in figure 11.

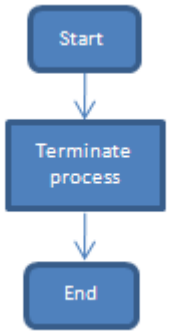


Figure 10 Closure process

The handle of the FMDM program is delivered to the closure process which then sends an termination order using a handle. This process forces program to quit and no program inside quit commands are needed. If the program freezes or there are any Graphical User Interface (GUI) malfunctions it would be hard to verify that the program was successful. With the help of handle it is possible to check if it does not exist anymore. When the handle function is empty the program was terminated successfully.

4.2 Processing Server

The script which controls the measurement devices consist of three main operations. These operations are initialization, data processing and graph plotting. This general functionality of processing server is represented in figure 12.

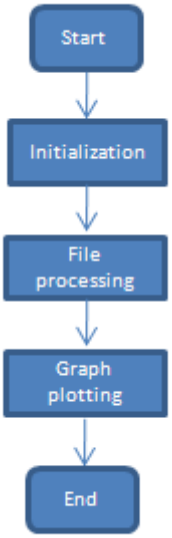


Figure 11 General functionality of the processing server

When the Windows scheduled task feature starts the script it firstly makes an initialization. The initialization function is almost the same as was in measurement device. After the initialization the script starts the data processing. Data processing is done using the described log processing program called FMLP. When the time for data processing has elapsed the script starts to calculate mean values and plot graphics. When finished the script quits.

4.2.1 Initialization

The initialization process deals with the fundamental variables that are globally used within the scripted program. The operation is illustrated in figure 13.

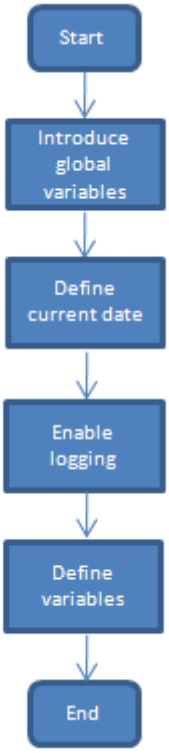


Figure 12 Initialization process

The initialization process is similar to the previously described one in measurement device. The only exception is that the processing server needs to store two date values. The first one is the value of the current date when the data processing process is executed. Another value of the date is the date of the preceding day to process the measurement data of the preceding day.

4.2.2 Data Processing

The data processing will take care of the most important part of the processing server script. It processes the collected data sent by the measurement device, analyses it and plots graphics well as stores several *RF* parameter mean values. The operation is represented in figure 14.

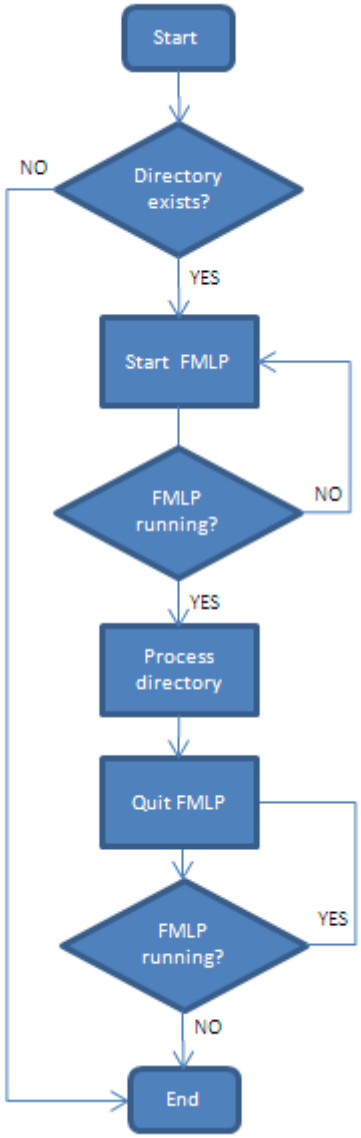


Figure 13 Data processing

The data processing starts with a directory check. If there is not a directory named according to yesterday the program will quit immediately. If the directory exits the script starts FMLP. Then it is checked that the FMLP did start, if not it would be tried to start again. If FMLP started accordingly the script starts to process the data in the directory. Data processing uses FMLP from the command prompt to input paths that need to be processed. Then the script waits for certain amount of time to continue. The time is given by the user. The data processing took with the explained equipment for about 90 minutes. If the user gives to the script a time which is less than the FMLP uses for data processing part of the collected data is not processed. After

the given time has elapsed the script starts to quit FMLP. It also verifies that FMLP was quit before it goes to graph plotting process.

4.2.3 Graph Plotting

The graph plotting takes care of plotting graph from the calculated mean values of five *RF* parameters. This procedure is illustrated in figure 15.

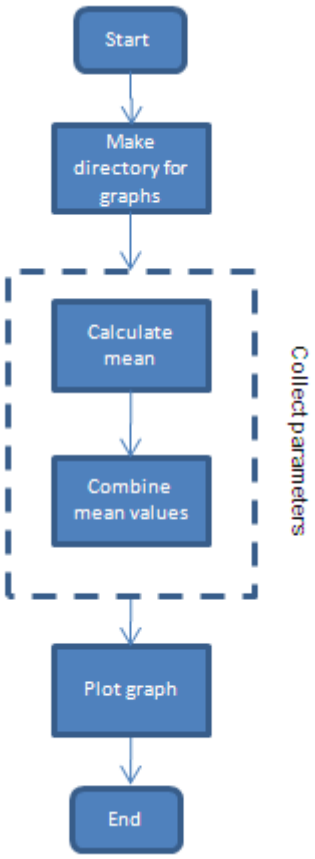


Figure 14 Graph plotting process

At first the process creates a directory to C:\FMLP_v2.4.5-2 \Graphs path. The directory is named according to yesterday functions value and locations stored in measurements folder e.g. C:\FMLP_v2.4.5-2 \Graphs\2009-10-23\pasila. If there are several measurement devices in the field, which have transmitted data to the processing server i.e. there are several folders named according to the measurement device location, the server is able to take into account also this situation. Then the script starts to calculate *RF* parameters. The collect parameters function is called five times because

there are five *RF* parameters that Digita is interested in. For the each parameter it firstly calculates the mean value per day as follows

$$Mean\ value\ (x) = \frac{\sum_{Number\ of\ rows} x}{Number\ of\ rows} \quad (1)$$

where x is the number of the column. Each column includes only values for one parameter. The values of the parameter are summed over the number of rows and then it is divided by the number of rows. The result is then the mean value. When the mean value is calculated the script combines seven days mean values in to one list. That list is then used to plot graphs where information from the seven days is represented.

The logical structures of the scripts are now introduced. These scripts are used to automate the quality measurement in wireless @450 broadband network. To analyze whether the automation works or not there is a need to put the system under test. Therefore the next section introduces the test results which are achieved by the previously introduced system.

5 TEST RESULTS OF THE QUALITY MEASUREMENT

This section introduces the results of the automated quality measurement process. The final version of the process was put into test in Digitas laboratory to determine whether it is ready or not. If everything goes as expected the process is noted to be ready for a real use.

The measurement device and the processing server was located in Digitas laboratory in a way that they would demonstrate the real but error-free measurement environment. The real measurement environment was introduced in chapter three. The test was made to RF parameter measurement which was in operation for three days. After test environment was set up the process was started.

5.1 Measurement Device

Firstly the measurement script was manually started. This was done only in the first time to test if the manual start also works. However this is not a mandatory in real situation when the operating system starts the script at 1.00 a clock. However, when the script was started it opened the FMDM air testing program.

From the FMDM window it is possible to follow the measurement process in real time. Main Status window shows the technical information about the connection. Logging Control window informs about air interface measurements. Measurement device generated ping traffic to performance server at address 192.168.20.2 which can be seen from the Traffic CLI window. The real time values for transmit and receive power etc. can be observed from the Channel Condition Plots window.

The script also opens a *CLI* window called *CMD* where it is easy to follow what the measurement device is doing at the moment on the script level. The *CLI* window operations are shown in figure 17.

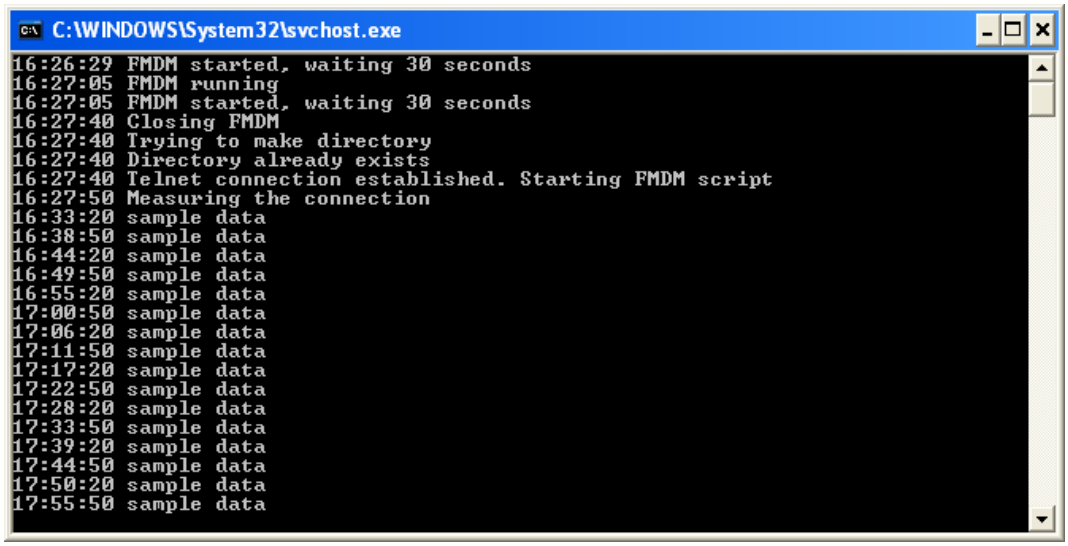


Figure 15 Measurement script in the CMD

From the *CMD* window we can observe what the measurement device have done and what it is doing at the moment. At the beginning FMDM have been started, also the fake program operation is executed, directory for the days measurements is made and the measurement itself is started.

The rest of the operations can be observed from the log file that measurement device has created. The logging file is represented in figure 18.

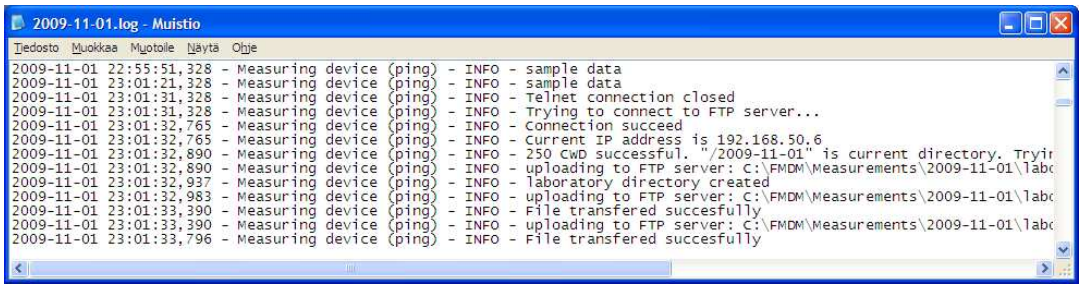


Figure 16 Mar 1, 2009 logging file, FTP connection

After the clock is over 23.00 the file transfer process starts. From the log files we could observe that the precise time in this test was 23:01. Next the measurement device tries to connect to the FTP server in processing server device. Few seconds after the connection was successfully established and the uploading begun. Firstly the laboratory folder is made under the C:\Measurements\2009-11-01\ path and then the file transfer starts. Final operations are represented in figure 19.

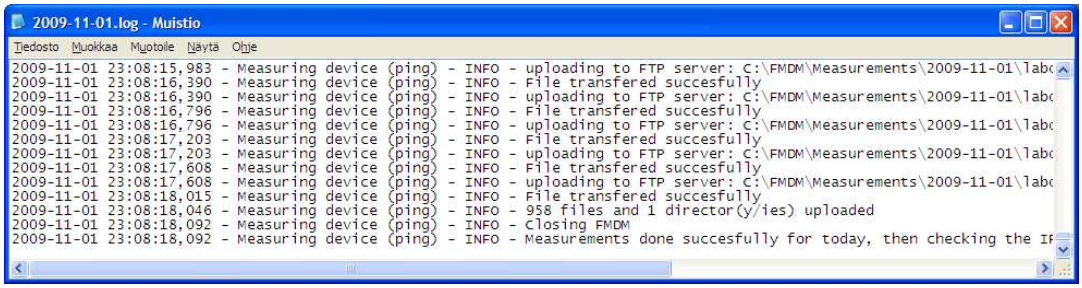


Figure 17 Mar 1, 2009 logging file, the end

From the logging file we can observe that total of 958 files was transferred and FMDM was successfully closed at 23:08.

5.2 Processing Server

The processing server was adjusted to start data processing at 5.00 a clock and graphic plotting at 7.00 a clock. In real use those times will vary depending on the situation. From the logging file we can observe that the server works as it should. Logging file is shown in figure 20.

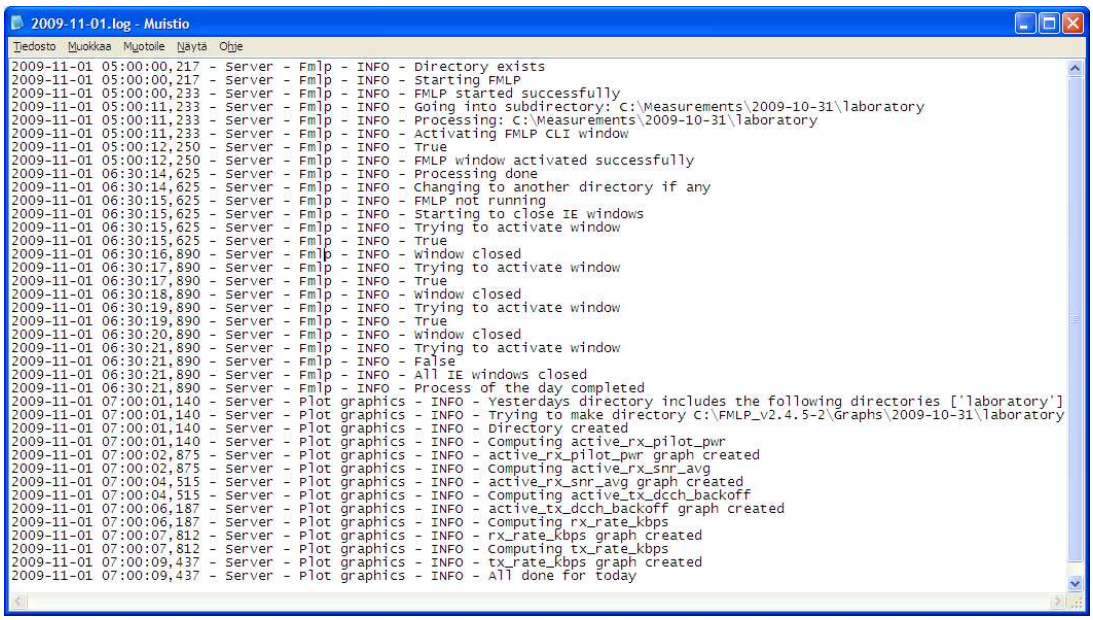


Figure 18 Mar 1, 2009 logging file of the processing server

The server has started data processing at 5.00 a clock. It checks if the file transfer have succeed by checking if last days folder exists. Then it starts the FMLP for data processing. The given path C:\Measurements\2009-10-31\laboratory is a correct path for data processing because the log is made 1st of November 2009. After the processing is done for the laboratory location, the script checks if there are any other locations. Because there are no

other locations it prepares to shut down. At 7.00 a clock the server starts to plot graphics for the five *RF* parameters. Firstly the script makes a directory for graphs to C:\FMLP_v2.4.5-2\Graphs\2009-10-31\laboratory path. Then it starts to calculate the *RF* parameter values and one by one plots the graph. After the five graphs are successfully made, server stops operation.

All five *RF* parameter graphs are included in the appendix A. The viewer must note that the test measurements occurred between 10th of October and 1st of November so the values before 10th of October are noted as zeros. As this study was concentrating on automation of quality measurement process the graphics are not explained in detail. However the purpose is to demonstrate that the automated process can produce useful graphs for the user.

6 CONCLUSIONS

Before this study a research had been made by Digita on how the quality measurement in the @450 broadband network could be carried out. The aim of this study was to implement a software based automation of the quality measurement system which was based on the previous research. The research question was how to implement an automated quality measurement system, to also fulfill the quality measurement requirements. The solution was to implement the system using Python programming language. In this study the quality measurement system of the wireless @450 broadband network was automated. This implementation reduces the work time spent on quality measurement by the employees and speeds up the processing of the results.

Based on the tests the quality measurement system, which was implemented using Python worked automatically as Digita requested. All the tasks from the beginning to the end i.e. from the measurements to the graph plotting were successfully accomplished. The measurement device made measurements between 1.00 and 23.00 a clock, and once the measurements were made it transferred the files to the server. The server then processed the measured data and in addition plotted five separate graphs. The system did not need user while it was in operation. Tests addresses that the system is able to operate automatically as a working wholeness in an error-free environment.

Although the automated quality measurement system can operate in error-free environment it does not fully quarantine that the system can operate in highly erroneous environment. The measurement device was developed in highly erroneous environment part by part for two months. As a result the measurement device could cope with all the issues it had during that time. If there are any other errors during the measurement period than were during the development stage the operation of the measurement device is not fully guaranteed. Even though there might be some disturbances that are not taken into account, they are not overly probable and most likely do not affect to the operations of the measurement device in a long period of time.

Problems to be tackled

During the design process there were some challenges which were not fully solved or had to be evaded. These challenges concerned mainly problems caused by the software or hardware providers. First, the Acer computer which was working as *FTP* server had some issues. For instance the Windows firewall crashed after few days if it were turned on. Once the firewall was turned off the computer worked without transfer issues. Secondly, the measurement software *FMDM* showed some unexpected behavior. That is why the victim process was implemented in order to get the *FMDM* program started. The process went so that two *FMDM* programs were started and the first one, called victim *FMDM* process was shut down. Then the secondly started *FMDM* worked without problems. In addition *FMDM* also reported from version mismatches even though nothing was changed. Thirdly, the *FMLP* program had problems with the window focus and command line interface. The *Flash-OFDM* measurement and the log processing software *FMDM* and *FMLP* are owned by Qualcomm and their further development relies on Qualcomm. Therefore there are no other choices than wait for bug fixes or patches which will repair the detected issues. Both of the Acer computers had also an uncommon problem. If the screen was closed up the load of the processor jumped to 100 %. The computer did not shut down because it was defined to stay open when the lid was closed. At the end the processor high load prevented both the measurement device and the processing server from normal operation. The only solution was to keep the lid open.

For further study

The implemented system could be improved by adding features like *GUI* (Graphical User Interface) and alternative Internet connection for the measurement device and advanced graphics for the processing server. The first improvement, *GUI*, would ease the use of the measurement device. *GUI* would make it possible to control the measurement script without the knowledge of Python programming language. Secondly an alternative connection could be used as a backup connection, which could be implemented e.g. using *3G* network. If there are serious problems in the wireless @450 connection and the measured data can not be transferred using @450 connection,

the measurement device could use the *3G* connection to send the measured data to the server. Finally the processing server could be improved to produce several types of graphs. The information in the graphs would stay as they are but there could be information from shorter periods of time. For instance the values of the parameters could be plotted every hour instead of plotting only one plot per day.

All the requirements of the quality measurement system became possible to be implemented using Python. One major advantage was that with Python all the different kinds of tasks in the quality measurement were achieved, without of use of any other programming language. Besides the quality measurement working automatically using Python, Digita achieved two important other objectives. One of the objectives was to reduce the working time spent on quality measurement. This objective was certainly met as the system did not need manual work during the laboratory test, which lasted three days. Another objective was to speed up the processing of reliable results. This objective was also met as the production of graphics and tables automatic, and thus fast and reliable. The server accomplished the required tasks early in the morning, and when the employees came to office the results were available for scrutiny. As a final conclusion it can be said that Python proved to be the right programming language for automation and the automation itself proved to meet the requirements requested by Digita.

REFERENCES

[1] Finnish Ministry of Transport and Communications (2005) *License Resolution*, [WWW document] Available at: http://www.lvm.fi/fileserver/upl696-Toimilupapäätös.pdf (Accessed Sep 4, 2009)

[2] Digita Oy (2009) *Digita lyhyesti*, [WWW document] Available at: http://digita.fi/digita_dokumentti.asp?path=1840;2249;3795 (Accessed Aug 14, 2009)

[3] Digita Oy (2007) *Digita verkko-operaattorina,* (Jan 29, 2009) [WWW document] Available at: http://www.450laajakaista.fi/at450/9059, (Accessed Aug 14, 2009)

[4] Python Software Foundation (1990-2009) *Official Website*, [WWW document] Available at: http://python.org/ (Accessed Aug 14, 2009)

[5] Python Software Foundation (1990-2009) *What is Python*, [WWW document] Available at: http://www.python.org/doc/faq/general/#what-is-python (Accessed Aug 14, 2009)

[6] Python Software Foundation (1990-2009) *What is Python Good For*, [WWW document] Available at: http://www.python.org/doc/faq/general/#what-is-python-good-for (Accessed Aug 14,2009)

[7] Internet Engineering Task Force (IETF) *File Transfer Protocol (FTP)*, [WWW document] Available at: http://tools.ietf.org/html/rfc959 (Accessed Aug 14, 2009)

[8] Internet Engineering Task Force (IETF) *Telnet Protocol Specification*, [WWW document] Available at: http://tools.ietf.org/html/rfc854, (Accessed Aug 14, 2009)

[9] Jussi Kasurinen (2008) *Python ohjelmointiopas, versio 1.2*. Lappeenranta: Lappeenranta University of Technology

[10] Swaroop C H (2005) *Byte of Python*, [WWW document] Available at: http://www.dgplug.org/byteofpython/byteofpython_120.pdf (Accessed Aug, 2009)

[11] Guido van Rossum (2008) *Python Library Reference*, Release 2.5.4, Python Software Foundation

[12] Qualcomm Flarion Technologies (2008) *Flarion Mobile Diagnostic Monitor (FMDM) Reference Manual.* New Jersey, USA

[13] Qualcomm Flarion Technologies (2007) *Flarion Mobile Log Processor (FMLP) Reference Manual.* New Jersey, USA

[14] Python Software Foundation (2008) version 2.5.4 http://www.python.org/download/releases/2.5.4/ (Accessed Oct 23, 2009)

[15] Mark Hammond (2003) version 214 for win32 (Accessed Oct 23, 2009) http://sourceforge.net/projects/pywin32/files/pywin32/Build%20209/

[16] Numpy developers (2009) version 1.3.0 for win32 http://sourceforge.net/projects/numpy/files/NumPy/1.3.0/numpy-1.3.0-win32-superpack-python2.5.exe/download (Accessed Oct 23, 2009)

[17] John H., Darren D. and Michael D. (2009) version 0.98.5.3 for win32 http://sourceforge.net/projects/matplotlib/files/matplotlib/ (Accessed Oct 23,2009)

[18] Tim Kosse (2009) version 0.9.33 http://filezilla-project.org/download.php?type=server (Accessed Oct 23, 2009)

[19] Finnish Ministry of Transport and Communication (2009) *General Service Commitment*, [WWW document] Available at: http://www.ficora.fi/attachments/suomimg/5kfKjcMSP/MPS58.pdf (Accessed Oct 30, 2009)

[20] The Scipy community (2008-2009) *Array Creation Routines*, [WWW document] Available at: http://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html (Accessed Oct 16, 2009)

[21] Python Software Foundation (1990-2009) *Win32All,* (Nov 2008) [WWW document] Available at: http://wiki.python.org/moin/Win32All (Accessed Oct 16, 2009)

[22] John H., Darren D. and Michael D. (2008) *matplotlib Official Website*, [WWW document] Available at: http://matplotlib.sourceforge.net/ (Accessed Oct 2, 2009)

[23] Python Software Foundation (1990-2009) *Logging Facility for Python,* [WWW document] Available at: http://docs.python.org/library/logging.html (Accessed Oct 2, 2009)

[24] Tim Kosse (2009) *Network Configuration,* [WWW document] Available at: http://wiki.filezilla-project.org/Network_Configuration#Passive_mode_2 (Accessed Oct 23, 2009)

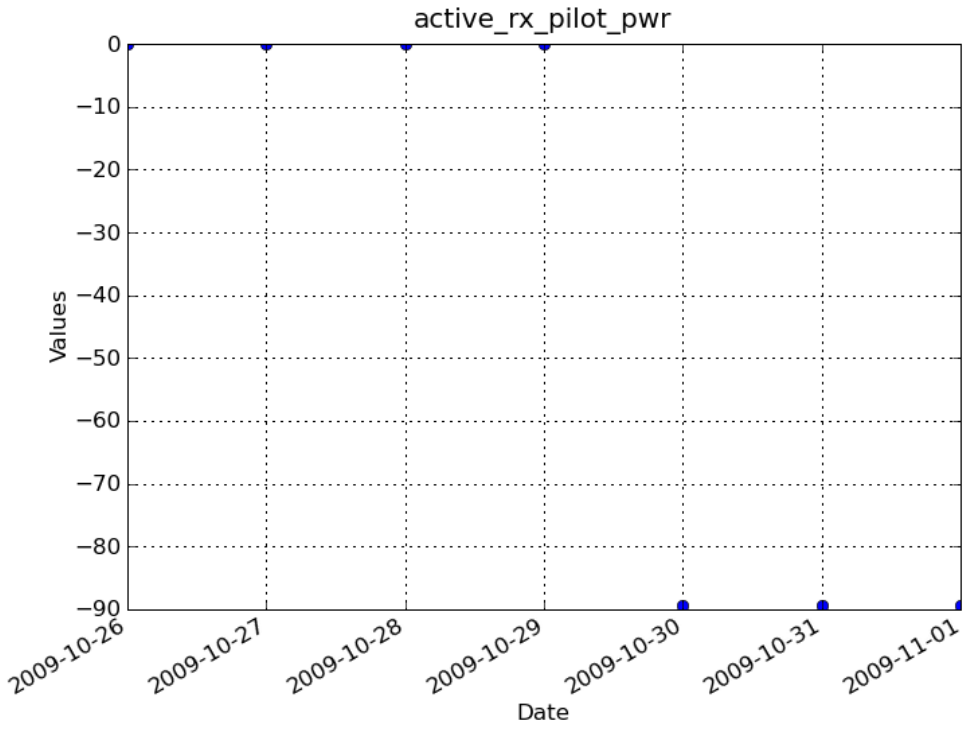
[25] TechArena (2008) *Windows Firewall Port Ranges,* [WWW document] Available at: http://forums.techarena.in/server-networking/890789.htm (Accessed Oct 23, 2009)

[26] Mark Lutz (2001) *Programming Python*. 2nd edition. California, USA: O'REILLY

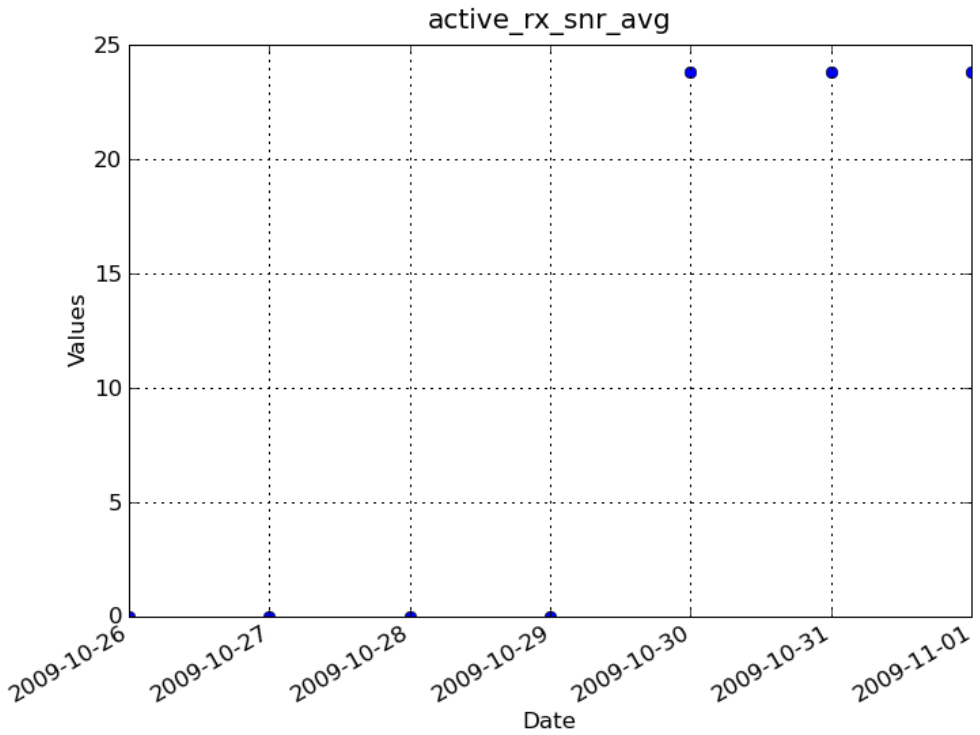
[27] Tapio Syrmä, Digita Oy (2008) *Mittalaite Flash-OFDM-tekniikalle*, [Thesis] Available at: https://publications.theseus.fi/bitstream/handle/10024/1286/Mittalaite%20Flash-OFDM-tekniikalle.pdf?sequence=1 (Accessed Dec 4, 2009)

Measurement period: **October 30, 2009 – November 1, 2009**
Measurement time: **1:00 – 23:00**
Location: **Laboratory**

Active Rx Pilot Power

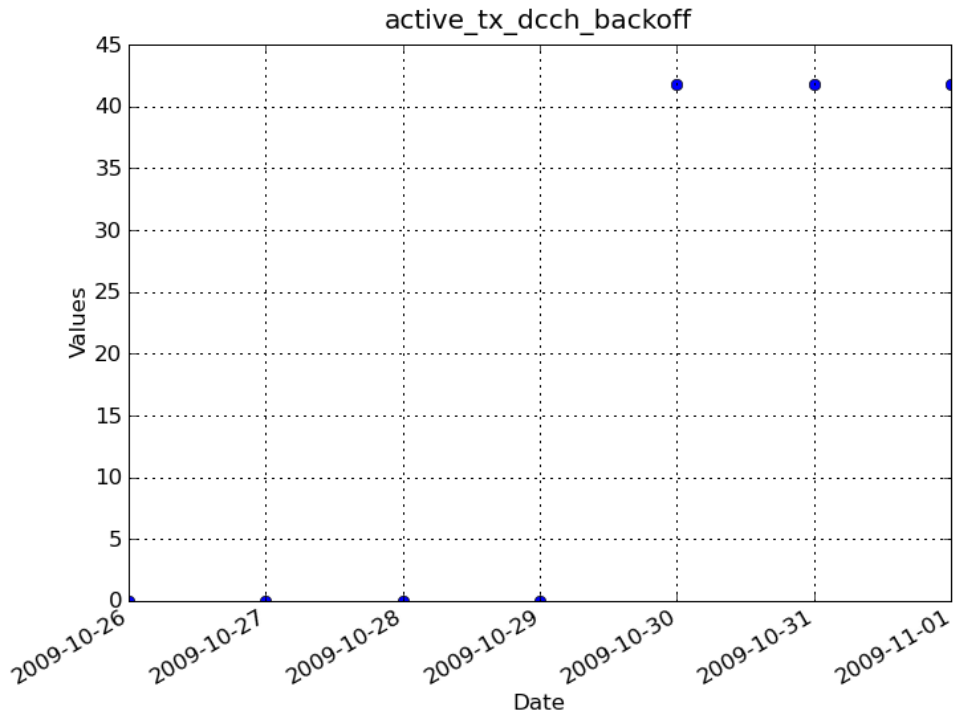


Active Rx SnR Average

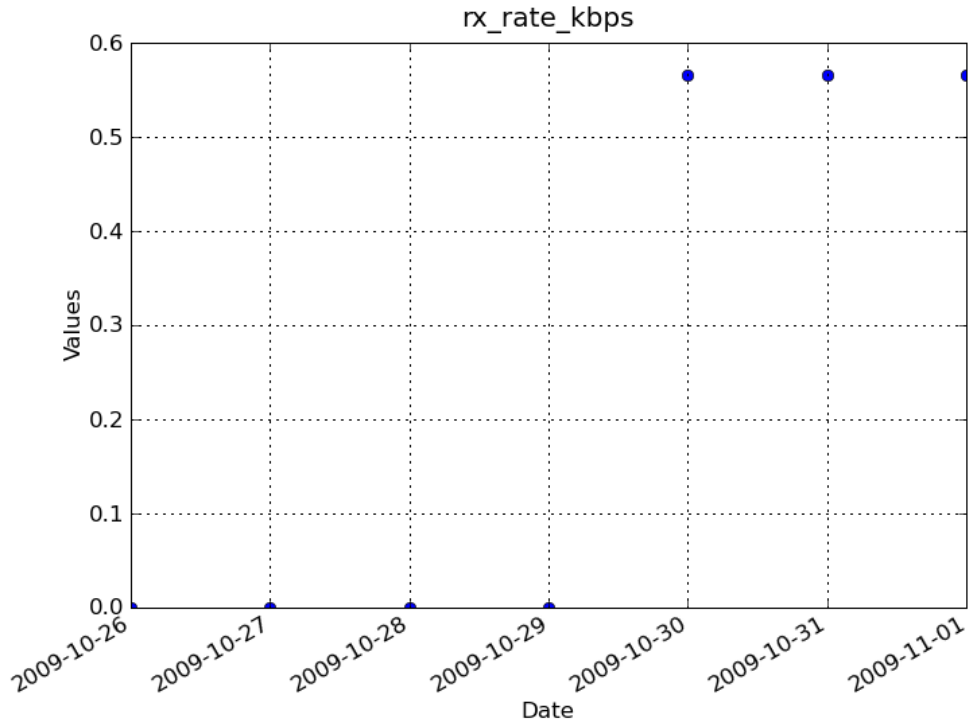


Measurement period: **October 30, 2009 – November 1, 2009**
Measurement time: **1:00 – 23:00**
Location: **Laboratory**

Active Tx DCCH Backoff

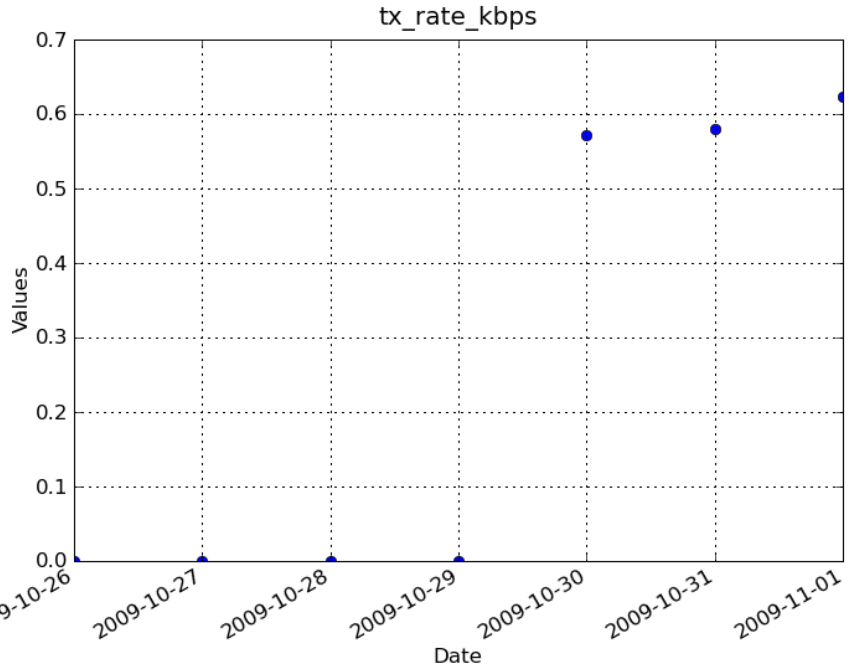


Receive data rate in kbps



Measurement period: **October 30, 2009 – November 1, 2009**
Measurement time: **1:00 – 23:00**
Location: **Laboratory**

Transmit data rate in kbps



Location: **Laboratory**
File name: **measure_script_lab.py**

The script to automate measurement device

```
# -*- coding: cp1252 -*-
# File: measure_script_lab.py

import telnetlib
import os
import datetime
import time
import subprocess
import ftplib
import logging
import socket
import win32com.client
import win32gui
import win32api

def initialize():
    # introduce global variables
    global location
    global date
    global logger

    # store values to variables
    location = 'laboratory'         # location
    date = datetime.date.today()    # current date

    # enable logging
    logger = logging.getLogger('Measurement device (ping)') # create logger
    logger.setLevel(logging.INFO)

    ch = logging.FileHandler('C:\\FMDM\\Logs\\' + str(date) + '.log') # create console handler and set level to info
    ch.setLevel(logging.INFO)

    formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s - %(message)s") # create formatter
    ch.setFormatter(formatter) # add formatter to ch
    logger.addHandler(ch) # add ch to logger

def fmdmStatus():
    # try to activate FMDM window
    try:
        success = win32gui.FindWindow(None, "Fmdm")
        time.sleep(1)
        if success != 0:
            print time.strftime("%H:%M:%S") + " FMDM running"
            logger.info("FMDM running, handle is %i " % success)
            return 0
        else: # needed when (None, "FMDM")
            print time.strftime("%H:%M:%S") + " FMDM not running"
```

```
            logger.info("FMDM not running, handle is %i " % success)
            return 1
    except win32gui.error: # needed when ("FMDM", None)
        print time.strftime("%H:%M:%S") + " FMDM not running"
        logger.info("FMDM not running")
        return 1

def startFmdm():
        path ="C:\Program Files\Flarion Mobile Diagnostic Monitor\FMDM"
        command = 'Fmdm.exe'
        os.chdir(path)

        process = subprocess.Popen(command, shell=None, stdin=None, stdout=None) # start FMDM

        print time.strftime("%H:%M:%S") + " FMDM started, waiting 30 seconds"
        logger.info("FMDM started, waiting 30 seconds")
        time.sleep(30) # wait 30 seconds
        return process

def makeDirectory():
    print time.strftime("%H:%M:%S") + " Trying to make directory"
    logger.info("Trying to make directory")

    # directory is made according to the date and location
    try:
        os.makedirs("'C:\\FMDM\\Measurements\\" + str(date) + '\\' + location) # we have to change the datetime output to string using str() -function
        print time.strftime("%H:%M:%S") + " Directory made according to date"
        logger.info("Directory made according to date")
    except WindowsError:
        print time.strftime("%H:%M:%S") + " Directory already exists"
        logger.info("Directory already exists")

def telnetToFmdm():
    # telnet-connection is opened
    host = 'localhost'
    port = 8100

    telnet = telnetlib.Telnet(host, port) # open a telnet-connection
    print time.strftime("%H:%M:%S") + " Telnet connection established. Starting FMDM script"
    logger.info("Telnet connection established. Starting FMDM script")
    telnet.write('log set dir=\"C:\/FMDM\/Measurements\/' + str(date) + '\/' + location +'\"\n') # have to use # to allow "s in the middle
    telnet.write('log set fileId=\"' + str(date) + '\"\n')
    telnet.write("sleep 10000\n")
    time.sleep(10)
    print time.strftime("%H:%M:%S") + " Measuring the connection"
    logger.info("Measuring the connection")

    # fmdm script
    running = True
    while running:
        if time.strftime("%H:%M:%S") >= "23:00:00":
```

```
            running = False
        else:
            telnet.write("'sleep 1000\n
            traffic set cmd="ping 192.168.20.2 -t"\n
            traffic start\n
            sleep 1000\n
            log start\n
            sleep 30000\n
            log stop\n
            traffic stop\n
            sleep 298000\n
            "') # pinging performance server

            time.sleep(330)
            print time.strftime("%H:%M:%S") + " sample data"
            logger.info("sample data")

    #print telnet.read_all()
    time.sleep(10)
    telnet.close()              # close a telnet-connection
    print time.strftime("%H:%M:%S") + " Telnet connection closed"
    logger.info("Telnet connection closed")

def transferFiles(toplocaldir):
    print time.strftime("%H:%M:%S") + " Preparing for transfer..."
    topremotedir = './'

    # ftp connection must be established and files to be transferred
    attempt = 1
    while attempt <= 20: # try to connect 20 times
        try:
            print time.strftime("%H:%M:%S") + " Trying to connect to FTP server..."
            logger.info("Trying to connect to FTP server...")
            connection = ftplib.FTP('192.168.50.5','username','password')
            print time.strftime("%H:%M:%S") + " Connection succeed"
            logger.info("Connection succeed")
            attempt = 21
        except ftplib.all_errors:
            print time.strftime("%H:%M:%S"), str(attempt) + ". attempt unable to connect to
FTP server"
            logger.error(str(attempt) + ". attempt unable to connect to FTP server")
            attempt = attempt + 1
        ip = socket.gethostbyname(socket.gethostname())
        logger.info("Current IP address is %s" % ip)

    connection.cwd(topremotedir)
    try:
        connection.mkd(str(date)) # make todays directory to FTP server
    except:
        time.sleep(0.1)
    topremotedir = connection.cwd(topremotedir + str(date)) # change top remote dir to
todays directory (./ -> ./YYYY-MM-DD)
    print time.strftime("%H:%M:%S") + " %s Trying to transfer files and directories" % to-
premotedir
    logger.info("%s Trying to transfer files and directories" % topremotedir)
```

```
    def uploadDir(localdir):
        global fcount
        global dcount
        fcount = dcount = 0
        localfiles = os.listdir(localdir)
        for localname in localfiles:
            localpath = os.path.join(localdir, localname)
            logger.info("uploading to FTP server: %s" % localpath)
            if os.path.isdir(localpath):    #if directory then recur into subdirs
                try:
                    connection.mkd(localname)
                    logger.info("%s directory created" % localname)
                except:
                    logger.info("%s directory not created" % localname)
                connection.cwd(localname)
                uploadDir(localpath)
                connection.cwd('..')
                dcount = dcount+1
            else:   # but if not directory then use binary mode transfer
                localfile = open(localpath, 'rb')
                connection.storbinary('APPE ' + localname, localfile, 1024)
                localfile.close()
                logger.info("File transfered succesfully")
                fcount = fcount+1

    uploadDir(toplocaldir)      # upload local top directory to the FTP server
    print time.strftime("%H:%M:%S") + " %i files and %i director(y/ies) uploaded" % (fcount,
dcount)
    logger.info("%i files and %i director(y/ies) uploaded" % (fcount, dcount))
    connection.quit()           # close the connection

def closeFmdm(process):
    print time.strftime("%H:%M:%S") + " Closing FMDM"
    logger.info("Closing FMDM")
    win32api.TerminateProcess(int(process._handle), -1) # terminate process

def main():
    initialize()                        # initialize

    toplocaldir = "C:\\FMDM\\Measurements\\" + str(date)

    victim_process = startFmdm()                # start victim fmdm
    time.sleep(5)

    running = True
    while running:
        if time.strftime("%H:%M:%S") >= "23:00:00": # run program until the time is 23:00
            break
        else:
            status = fmdmStatus()               # is fmdm running
            if status != 0:
                victim_process = startFmdm()
                time.sleep(5)
            else:
```

```
            process = startFmdm()          # start fmdm
            time.sleep(5)
            closeFmdm(victim_process)      # close victim fmdm
            makeDirectory()               # make a directory where results are saved
            try:
               telnetToFmdm()             # telnet to fmdm
            except:
               time.sleep(1)
            try:
               transferFiles(toplocaldir)  # transfer files to FTP server
            except:
               time.sleep(1)
            closeFmdm(process)             # close FMDM

    ip = socket.gethostbyname(socket.gethostname())
    logger.info("Measurements done succesfully for today, then checking the IP: %s " % ip)

if __name__ == "__main__":
    main()
```

Location: **Laboratory**
File name: **fmlp_script_lab.py**

The script to automate data processing using FMLP

```
# -*- coding: cp1252 -*-
# File: fmlp_script_lab.py

import os
import datetime
import time
import subprocess
import win32com.client
import logging
import win32gui
import win32api

def initialize():
    # introduce global variables
    global date
    global logger
    global shell

    # store values to variables
    date = datetime.date.today()
    shell = win32com.client.Dispatch("WScript.Shell") #taking win32com in to operation

    # enable logging
    logger = logging.getLogger('Server - Fmlp') # create logger
    logger.setLevel(logging.INFO)

    ch = logging.FileHandler('C:\\FMLP_v2.4.5-2\\Logs\\' + str(date) + '.log') # create console handler and set level to info
    ch.setLevel(logging.INFO)

    formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s - %(message)s") # create formatter
    ch.setFormatter(formatter) # add formatter to ch
    logger.addHandler(ch) # add ch to logger

    # store values to variables, must be after logging
    date = date + datetime.timedelta(days=-1) # store yesterdays date to date variable

def dirStatus(Dir):
    try:
        os.chdir(Dir)
        logger.info("Directory exists")
        return 0
    except WindowsError:
        logger.info("Directory does not exist")
        return 1

def fmlpStatus():
    # check FMLP status
```

```
    try:
        success = win32gui.FindWindow(None, "FMLP Main Window")
#http://msdn.microsoft.com/en-us/library/ms633499(VS.85).aspx
        time.sleep(1)
        if success != 0:
            logger.info("FMLP running")
            return 0
        else: # needed when (None, "FMLP")
            logger.info("FMLP not running")
            return 1
    except win32gui.error: # needed when ("FMLP", None)
        logger.info("FMLP not running")
        return 1

def startFmlp():
    logger.info("Starting FMLP")
    path = r'C:\FMLP_v2.4.5-2'
    command = "fmlp.exe cli"
    os.chdir(path)
    process = subprocess.Popen(command, shell=None)
    logger.info("FMLP started successfully")
    time.sleep(1)
    return process

def quitFmlp(process):
    logger.info("Trying to close FMLP...")
    logger.info("Handle is ")
    logger.info(process._handle)

    win32api.TerminateProcess(int(process._handle), -1)
    logger.info("FMLP closed successfuly")

def quitIes():
    logger.info("Starting to close IE windows")
    logger.info("Trying to activate window")
    success = shell.AppActivate("C:\\Measurements\\")
    logger.info(success)
    while success == True:
        time.sleep(1)
        shell.SendKeys('%{F4}')
        logger.info("Window closed")
        time.sleep(1)
        logger.info("Trying to activate window")
        success = shell.AppActivate("C:\\Measurements\\")
        logger.info(success)
    logger.info("All IE windows closed")

def fmlpWindowFocus():
    logger.info("Activating FMLP CLI window")
    success = shell.AppActivate("C:\\WINDOWS\\") # must be done this way because oth-
erwise scheluded task wont run cause of the different title
    time.sleep(1)
    logger.info(success)
    if success == True:
        logger.info("FMLP window activated successfully")
```

```
        time.sleep(1)
        return 0
    else:
        logger.error("FMLP window could not be activated")
        time.sleep(1)
        return 1

def processDir(Dir):
    files = os.listdir(Dir)
    for name in files:
        path = os.path.join(Dir, name)
        if os.path.isdir(path): #if directory then go into subdirectories
            logger.info("Going into subdirectory: %s" % path)
            processDir(path)
        else: # but if not directory then
            logger.info("Processing: %s" % Dir)
            status = fmlpWindowFocus()
            if status != 0:
                #something went wrong while trying to activate FMLP window
                logger.error("Processing can not be done")
            else:
                time.sleep(1)
                shell.SendKeys("process_all \"" + Dir + "\"") # send FMLP process_all command
                time.sleep(0.1)
                shell.SendKeys('{ENTER}')
                time.sleep(5400) # must be adjusted correctly, otherwise processing fails, give enough time (5400 s = 1,5 h)
                logger.info("Processing done")
                break # if even one file found then break from the loop and process the directory. Otherwise the directory will be processed as many times as there are files
        logger.info("Changing to another directory if any")

def main():
    initialize()

    topdir = "C:\Measurements\\" + str(date)

    status = dirStatus(topdir)
    if status != 0:
        logger.info("Files have not been transfered")
    else:
        # Directory exists, everything allright, continuing normally
        process = startFmlp()   # start FMLP and store handle
        while status != 0:      # check status after every 2 seconds
            time.sleep(10)
            status = fmlpStatus()
        time.sleep(10)
        processDir(topdir)      # process yesterdays directory
        # Try to close FMLP
        status = fmlpStatus()
        while status == 0:
            quitFmlp(process)   # close fmlp
            time.sleep(5)
            status = fmlpStatus()
        # close IE windows
```

```
        quitIes()

        logger.info("Process of the day completed")

if __name__ == "__main__":
    main()
```

Location: **Laboratory**
File name: **handle_data_lab.py**

The script to automate the further processing of the data

```
# -*- coding: cp1252 -*-
# File: handle_data_lab.py

import os
import datetime
import time
import subprocess
import win32com.client
import logging
import win32gui
import csv

from matplotlib.pyplot import figure, show, plot_date
from matplotlib.dates import DayLocator, HourLocator, DateFormatter, drange, num2date
from numpy import arange, loadtxt

def initialize():
    # introduce global variables
    global date
    global logger
    global shell

    # store values to variables
    date = datetime.date.today()
    shell = win32com.client.Dispatch("WScript.Shell") #taking win32com in to operation

    # enable logging
    logger = logging.getLogger('Server - Plot graphics') # create logger
    logger.setLevel(logging.INFO)

    ch = logging.FileHandler('C:\\FMLP_v2.4.5-2\\Logs\\' + str(date) + '.log') # create console handler and set level to info
    ch.setLevel(logging.INFO)

    formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s - %(message)s") # create formatter
    ch.setFormatter(formatter) # add formatter to ch
    logger.addHandler(ch) # add ch to logger

    # store values to variables, must be after logging
    date = date + datetime.timedelta(days=-1) # store yesterdays date to date variable

def makeDirectory(path):
    logger.info("Trying to make directory %s" % path)

    # directory is made according to the date and location
    try:
        os.makedirs(path) # we have to change the datetime output to string using str() - function
```

```
        logger.info("Directory created")
    except WindowsError:
        logger.info("Directory already exists")

def calculateMean(n, date, location):
    dataPath = "C:\\Measurements\\" + str(date) + "\\" + location + "\\transformed_collected_combined.csv"

    try: # try except because otherwise process ends. Must be done this way because transfers are not sure
        csvReader = csv.reader(open(dataPath), delimiter=',', quotechar='|')
        valueList = []
        numberOfRows = 0 # this way we dont get "header" row taken into account
        for row in csvReader: # data is in lists
            if len(row) < 30: # don't care about the first row
                csvReader.next()
            else:
                value = row[n] # read the n:th (ännännes) value
                if value == "NaN": # put NaNs to zeros for calculation
                    value = 0
                valueList.append(value) # add values to list one by one
                numberOfRows = numberOfRows + 1 # 14208 total usually
        valueList = valueList[1:] # delete the second row element also

        i = 0
        valueSum = 0
        while i < numberOfRows-1: # while i is less than the number of rows, indexes start from 0 so there is a need to -1
            valueSum = valueSum + float(valueList[i])
            i = i+1
        valueMean = valueSum / numberOfRows
        return valueMean
    except:
        valueMean = 0
        return valueMean

def combineMeans(n, endDate, location):
    parameterList = []
    for i in range(1,8): # week (7 days) back
        date = endDate + datetime.timedelta(days=-(7-i)) # reverse order starting from past to current date
        meanValue = calculateMean(n, date, location)
        parameterList.append(meanValue) # add value of the date to list
        date = endDate   # back endDate  date
    return parameterList

def plotGraph(startDate, endDate, parameterList, parameter, location):
    endDateTemp = endDate + datetime.timedelta(days=1) # temporarily add one day to be able to plot. Otherwise x and y mismatch
    delta = datetime.timedelta(hours=24)
    dates = drange(startDate, endDateTemp, delta)

    fig = figure()
    ax = fig.add_subplot(111)
```

```
    ax.plot_date(dates, parameterList, xdate=True, ydate=False) #plot_date(x, y, fmt='bo', tz=None, xdate=True, ydate=False, **kwargs)

    ax.grid(True)          # show grid
    ax.set_xlabel('Date')   # set x-label
    ax.set_ylabel('Values') # set y-label
    ax.set_title(parameter) # set title
    #set_ybound(lower=None, upper=None) #Set the lower and upper numerical bounds of the y-axis. This method will honor axes inversion regardless of parameter order.

    # format date into YYYY-MM-DD format
    ax.xaxis.set_major_formatter( DateFormatter('%Y-%m-%d') )

    # x-axis dates in 45 degrees
    fig.autofmt_xdate()

    fig.savefig("C:\\FMLP_v2.4.5-2\\Graphs\\" + str(endDate) + "\\" + location + "\\" + str(endDate) + "_" + location + "_" + parameter + ".png", dpi=100)

def main():
    initialize()

    print "Program running..."
    endDate = date # endDate is yesterday
    Dir = "C:\\Measurements\\" + str(date)

    files = os.listdir(Dir)
    logger.info("Yesterdays directory includes the following directories %s" % files)
    for name in files: # go through all locations if locations does not exist no graphics are plotted
        path = os.path.join(Dir, name)

        graph = "C:\\FMLP_v2.4.5-2\\Graphs\\" + str(endDate) + "\\" + name
        makeDirectory(graph)

        # calculate mean values for active_rx_pilot_pwr (63)
        logger.info("Computing active_rx_pilot_pwr")
        parameter = "active_rx_pilot_pwr"
        parameterList = combineMeans(63, endDate, name) # combine mean values to one list
        # then the values must be plotted
        startDate = endDate + datetime.timedelta(days=-6) # lauantai 2009-10-03
        plotGraph(startDate, endDate, parameterList, parameter, name)
        logger.info("active_rx_pilot_pwr graph created")

        # calculate mean values for active_rx_snr_avg (65)
        logger.info("Computing active_rx_snr_avg")
        parameter = "active_rx_snr_avg"
        parameterList = combineMeans(65, endDate, name) # combine mean values to one list
        # then the values must be plotted
        plotGraph(startDate, endDate, parameterList, parameter, name)
        logger.info("active_rx_snr_avg graph created")

        # calculate mean values for active_tx_dcch_backoff (70)
```

```
        logger.info("Computing active_tx_dcch_backoff")
        parameter = "active_tx_dcch_backoff"
        parameterList = combineMeans(70, endDate, name) # combine mean values to one
list
        # then the values must be plotted
        plotGraph(startDate, endDate, parameterList, parameter, name)
        logger.info("active_tx_dcch_backoff graph created")

        # calculate mean values for rx_rate_kbps (81)
        logger.info("Computing rx_rate_kbps")
        parameter = "rx_rate_kbps"
        parameterList = combineMeans(81, endDate, name) # combine mean values to one
list
        # then the values must be plotted
        plotGraph(startDate, endDate, parameterList, parameter, name)
        logger.info("rx_rate_kbps graph created")

        # calculate mean values for active_rx_pilot_pwr (82)
        logger.info("Computing tx_rate_kbps")
        parameter = "tx_rate_kbps"
        parameterList = combineMeans(82, endDate, name) # combine mean values to one
list
        # then the values must be plotted
        plotGraph(startDate, endDate, parameterList, parameter, name)
        logger.info("tx_rate_kbps graph created")
    logger.info("All done for today")
if __name__ == "__main__":
    main()
```