Aalto University

School of Electrical Engineering

Department of Communications and Networking

Degree Programme in Communications Engineering

Xi Chen

Automated Testing for Mobility Management Entity of Long Term Evolution System

Master’s thesis

Espoo, October 20, 2012

Supervisor:	Professor Jyri Hämäläinen, D.Sc
Instructor:	Risto Nissinen, M.Sc.

AALTO UNIVERSITY ABSTRACT OF THE MASTER’S THESIS
SCHOOL OF ELECTRICAL ENGINEERING
DEPARTMENT OF COMMUNICATIONS AND NETWORKING

Author: Xi Chen

Title: Automated Testing for Mobility Management Entity of Long Term Evolution System

Date: 20.10.2012 **Language:** English **Number of pages**: 7+72

Department of Communications and Networking

Professorship: Radio communications **code:** S-72

Supervisor: Prof. Jyri Hämäläinen

Instructor: M.Sc. Risto Nissinen

This thesis starts with a brief introduction about LTE system and Automated Testing, after which, it describes the implementation of test automation for Mobility Management Entity (MME) of 4G LTE core network. The thesis work is focus on testing of MME logical interfaces s1, s6 and s11. There is a test suite that is developed, and is used to remotely control the Tektronix G35 tester to generate network traffic for testing purpose. Besides, the test suite sends Man Machine Language (MML) commands to the MME hardware platform called ATCA, and periodically collects counters’ data, with which the test suite is able to track the performance of the three interfaces of MME. As a conclusion, automated testing for MME is found to be very important. It provides an efficient way to generate a clear picture of the performance of MME, which is helpful when improving the quality of MME.

Keywords

Test automation, Mobility Management Entity, test suite, Tektronix G35, Man Machine Language

Preface

This MSc thesis is done in Nokia Siemens Networks Oy (NSN), in the team of NWS CIC RD IP Control PET&PST. The thesis work is a research about test automation of Mobility Management Entity (MME) of LTE core network with the usage of Tektronix G35 tester. The purpose of the work is to track the performance of NSN's ATCA MME platform by implementing a test suite which could collect MME counters' statistics.

I would like to express my gratitude to Mr Risto Nissinen, my instructor, for providing me not only the opportunity to work on this project, but also many helps during my work. I would also like to thank Tanja Lindgren, Joni Turkki for their kind assistance.

Espoo, 20.10.2012

Xi Chen

Abbreviation and explanations

3GPP	3rd generation partnership project
FDD	Frequency Division Duplex
TDD	Time Division Duplex
eNB	evolved Node B
SAE	System Architecture Evolution
SGW	Serving Gate Way
GTP	GPRS Tunneling Protocol
HSPA	High Speed Packet Access
I-HSPA	Internet HSPA
SGSN	Serving Gateway Support Node
GGSN	Gateway GPRS Support Node
BTS	Base Transceiver Station
EPS	Evolved Packet System
PDN	Packet Data Network
QoS	Quality of Service
EMM	EPS Mobility Management
ECM	EPS Connection Management
TA	Tracking Area
IMSI	International Mobile Subscriber Identity
TAU	Tracking Area Update
RRC	Radio Resource Control
HSS	Home Subscriber Server
TA	Tracking Area
NAS	Non Access Stratum
S1AP	S1 Application Part
TEID	Tunnel Endpoint Identifiers
UTRAN	Universal Terrestrial Radio Access Network
E-UTRAN	Evolved UTRAN
EPC	Evolved Packet Core
ROHC	Robust Header Compression

ARQ	Automatic Repeat Request
PDU	Protocol Data Unit
H-ARQ	Hybrid ARQ
ATCA	Advanced Telecommunications Computing Architecture
G35	A protocol tester device
LTE	Long Term Evolution
MME	Mobility Management Entity
SAE	System Architecture Evolution
WCDMA	Wideband Code Division Multiple Access
ATLM	Automated Test Life-cycle Methodology
PICMG	PCI Industrial Computer Manufactures Group
CPPU	Control Plane Processing Unit
SMMU	Signaling and Mobility Management Unit
MMDU	Mobility Management and Database Unit
IPDU	IP Director Unit
MCHU	Marker and Charging Unit
OMU	Operation and Maintenance Unit
CORBA	Common Object Request Broker Architecutre
IDL	Interface Definition Language
ORB	Object Request Broker
DPE	Distributed Programming Environment
SSH	Secure Shell
MML	Man Machine Language

Table of Contents

Preface………………………………………………………………………..i

Abbreviation and explanations……………………………………………..ii

Table of Contents...iv

Chapter 1 Introduction

1.1 Background

During the last decades, mobile communications has evolved fast. It has been a long way from the first radio communication experiment to the current cellular system. The development of mobile technologies nowadays is mainly conducted by global standards developing organizations like 3GPP [1] which involves a lot of people. The evolvement of mobile communications can be characterized by different generations. The so called 1G was the analog mobile radio system in the 1980s. The first digital mobile system was 2G system which was followed by the 3G system, the first mobile system handling broadband data [1].

The IP based services delivered over Internet was heading towards wireless which drives the mobile communication systems to deliver more and more IP based services. As a result, the traditional circuit-switched services will be ported over IP networks. With the massive increase in mobile data traffic, and as networks continue to converge towards all-IP infrastructures [2], many operators are considering LTE, or Long Term Evolution, a new network technology first defined in 3GPP release 8 [1] which promises to be an all-IP network infrastructure that will dramatically decrease the cost of delivering high-value multimedia services and applications.

Given the advantages, LTE also introduces some unique challenges for network operators that must be overcome in order to profitably deliver high quality services and applications. Coming from a 2G/3G environment, operators have come to understand what specific network parameters impact their network and services. In the all-IP LTE environment, network behavior requires new tools and techniques optimized for an IP environment. Interoperability is also a major issue since most operators deploying LTE will be doing so alongside their existing 2G/3G legacy networks.

With the advent of new wireless standards like LTE, mobile networks are evolving towards complex heterogeneous systems consisting of a multitude of different technologies. As a consequence, thorough testing of protocol implementations and complete mobile network elements is becoming more important than ever before. Telecom manufactures seek to improve test efficiency by means of test automation [3]. Related test tools must be able to address all necessary technologies as well as all phases of the test cycle to be cost-efficient.

1.2 Motivation behind the thesis work

The rapid growth of the number of subscribers forces the operators and vendors to adopt new technologies and standards to provide better services at a lower cost than the earlier systems [7]. The operators and vendors will undoubtedly drive the new technologies to stay ahead of competitions.

The LTE Mobility Management Entity (MME) in this thesis work adopts ATCA as its hardware platform [8]. ATCA (Advanced Telecom Computing Architecture) is a hardware platform with bladed architecture which is designed according to the requirements set by the next generation of communications equipment. Applying to the telecommunications industry, ATCA eases the serviceability of LTE/4G and other next generation wireless and wired infrastructure.

There are five key goals for ATCA [4]: to enable interoperability among manufacturers’ components, to reduce time to market, to reduce development costs, to reduce the length of development cycles and to create an open, standards-based scalable, flexible platform architecture.

The hardware of LTE MME consists of a group of interconnected computer units. Each computer unit has its own real-time task to do. There are several plug-in units inserted to the shelf of ATCA, each plug-in unit builds up one computer unit.

Because of the big amount of data traffic in LTE core networks, whenever we want to test the functionality or performance of a certain network element, like MME, which is the key control node of LTE network, we would like to do it automatically. This is the primary motivation behind the thesis work, which was done to evaluate the performance of MME ATCA platform of LTE core network by implementing automation testing cases which records and tracks the specific statistics to provide vendor an overview of the performance of MME platform.

1.3 Objectives and hypothesis

The performance of the MME platform is crucial, as it is the key control node that is responsible for the mobility management[5] of the LTE access network. We want to make sure that it functions smoothly and efficiently, that’s also the reason why we need to develop automation testing scenarios.

The aim of this thesis is to implement test suite which can monitor the s1, s6, s11 interfaces [6] of Mobility Management Entity (MME) of 4G LTE core network, tracking the traffic of different unit counters, tracking the CPU loads of interested computer units, and generating statistic graphs which show the trends of counters and CPU loads during the testing period. These test cases consist of two parts, which are remote protocol tester control and automated regression testing of MME. During the development of the test suite, we will utilize a device called Tektronix G35 protocol tester [9] to define the traffic profile and generate traffic to MME.

The test suite is supposed to have a user interface to allow engineer to define the period of how long the automation test is going to run. Normally, the automation testing suite runs overnight, so that engineers could see the final result in the next morning. Based on the testing result, engineers are able to evaluate the performance of the ATCA platform based MME, and take actions for certain failure accordingly. The stability is very important for the network element, like MME, and our target is to guarantee zero failure, since large financial losses can be faced if the network element is not working properly in the commercial use by operators.

1.4 Thesis structure

The first two chapters generally introduce the LTE system and its core network elements. Chapter 3 describes what tools, environment and methodology have been used in this thesis work, while chapter 4 tells what automated testing and regression testing are, and discuss the lifecycle of automated testing. The thesis work implementation is described in chapter 5. Chapter 6 indicates the result and conclusion of the thesis work. A further improvement is also proposed in chapter 6.

Chapter 2 LTE and SAE architecture

LTE is the 4th generation mobile communication system, which is packet switched system, unlike the 2G and 3G cellular systems that designed from circuit switched operations perspective. LTE is designed with only packet-switched network from end to end for high speed data services. The LTE system targets to a peak data rate of 100 Mbps in downlink and up to 50 Mbps in uplink.

This chapter firstly gives the general view of LTE system architecture, and then introduces the radio interface protocols of E-UTRAN. Finally, it describes the Mobility Management and Connection Management in LTE.

2.1 LTE network architecture

The entire LTE network is composed by the radio access network (Evolved UTRAN) and the core network (Evolved Packet Core), both of which have been defined as new components of the end-to-end network in release 8 of 3GPP specification [1]. As a 4th generation system, the LTE radio interface provides new packet data capability with an evolved core network which is specified and commonly known as System Architecture Evolution (SAE).

The 3GPP system architecture evolution (SAE) is concerned with the evolved core network architecture. The SAE system defines a scalable IP-based architecture of the Evolved Packet System (EPS) which consists of Radio Access Network also known as Evolved UTRAN and the Evolved Packet Core [10]. The Evolved Packet System is a packet based system, where the voice transport is based on Voice over IP (VoIP).

E-UTRAN is the radio access part of LTE architecture. It mainly consists of user equipment (UE) and evolved Node B (eNB)s. The eNB supports LTE radio interface and provides the packet-switched functionality of traditional radio network controller (RNC) of 3G. Thus, the RNC was removed from Evolved UTRAN, which means there are fewer network entities in the architecture.

The Uu is the LTE radio interface which is between user equipment and eNB. The eNBs are interconnected with each other with the X2 interface which is used during an inter-eNB handover. The eNBs are also connected to the Evolved Packet Core by means of S1 interface. To be more specific, the eNBs are connected to the Mobility Management Entity (MME) with S1-MME interface which carries control plane signaling information between eNB and MME. They are connected to the Serving Gateway (S-GW) with S1-U interface which carries the user plane data over a GTP tunnel.

Figure 1 shows the overview of LTE network architecture.

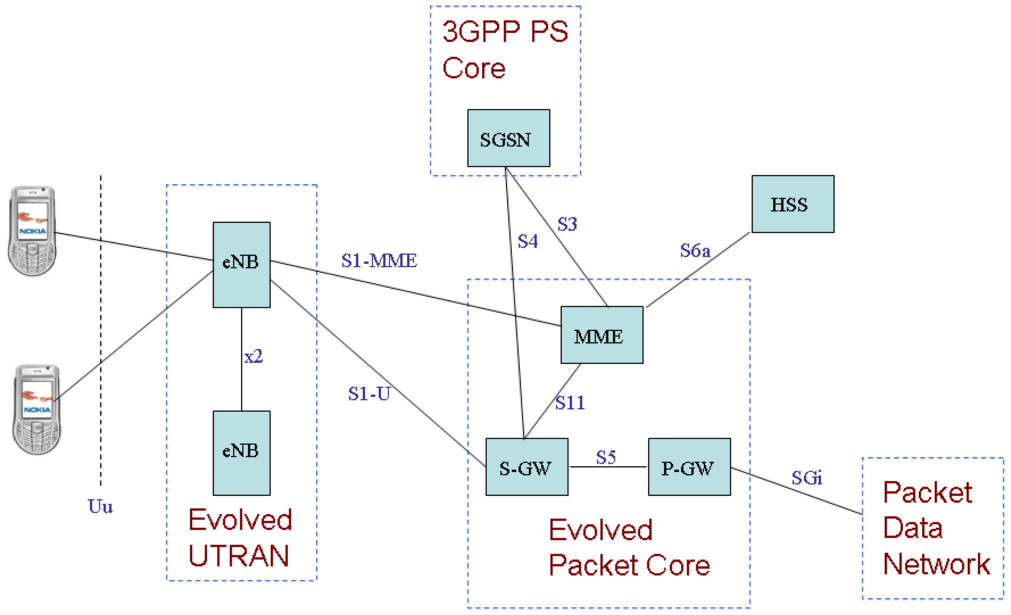


Figure 1. LTE network architecture [10]

The Evolved Packet Core part consists of Mobility Management Entity (MME), Serving Gateway(S-GW) and Packet Data Network Gateway (P-GW), they are introduced in 3GPP Release 8.

MME is the key node in LTE EPC. It is responsible for tracking and paging procedure, performs the signaling and control functions to manage the User Equipment (UE) access to network connections. MME basically has all control plane functions related to subscriber and session management [11].

The primary function of S-GW is to manage user-plane mobility, like the handover with neighboring eNBs. It also takes care of data transfer across user plane. P-GW is an anchor point for sessions towards the external PDN. It mainly supports policy enforcement features, packet filtering and charging [11].

The interfaces related with EPC are defined by 3GPP [11] as follows:

The S4 interface between S-GW and SGSN provides a GTP tunnel for the user plane during an inter-system handover.

The S3 interface carries signaling between the MME and the Serving GPRS Support Node (SGSN) located in 2G or 3G packet switched core network.

The S11 interface is used to carry signaling message between Serving Gateway and MME.

The S6a interface transfers subscription and authentication data between the Home Subscriber Server (HSS) and MME.

The interface between PDN Gateway and the packet data network (PDN) is the SGi. The PDN could be an external public or private IP network, or the operator's IP network, for example, the network which provides IP Multimedia Subsystem (IMS) service.

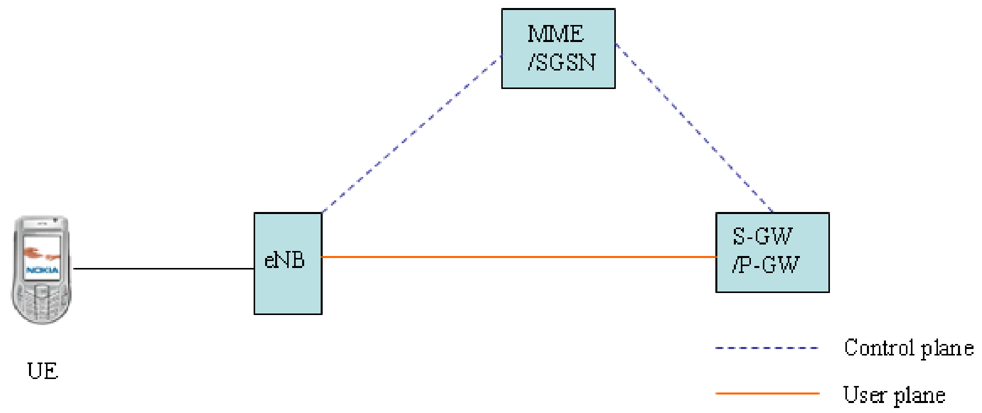


Figure 2. LTE Control plane traffic and User plane traffic

There is a user plane connectivity between the access network and the external packet data network, which is provided by S-GW and PDN Gateway (P-GW)

In Evolved Packet Core network, the MME element provides the basic control plane functionality. There is no user plane traffic go through MME.

2.2 Radio Interface protocols of E-UTRAN

The LTE radio interface is designed based on a layered protocol stack [13] which consists of user plane protocol stack and control plane protocol stack. Figure 3 shows the protocol stack.

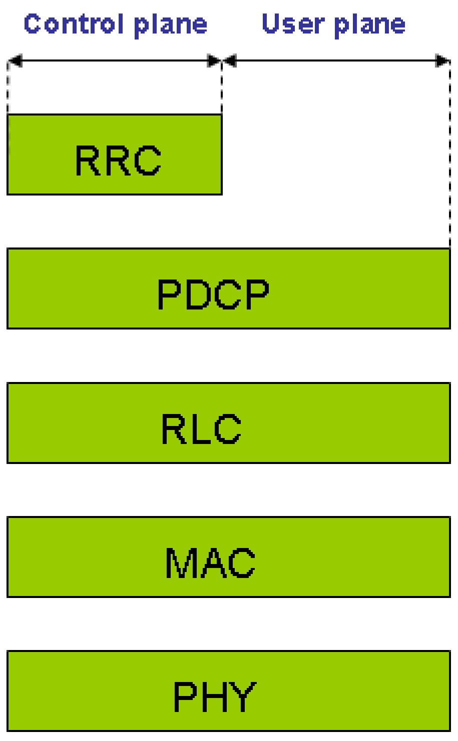


Figure 3. Radio interface protocol stack of LTE

The following layers compose the LTE radio interface protocol:

- **Radio Resource Control (RRC):** The RRC layer is part of LTE air interface control plane. It performs the control plane functionalities, for example, paging; establish, maintenance and release the RRC connection between UE and E-UTRAN; broadcast of system information related to the non-access stratum (NAS); broadcast of system information related to the access stratum (AS), and so on.

- **Packet Data Convergence Protocol (PDCP):** This layer exists in UE and eNB, it is part of the control plane and user plane of LTE air interface. The main functionality of this layer includes compression and decompression of IP packet header based on the Robust Header Compression (ROHC) protocol; ciphering and deciphering of data, and so on. For the control plane, the main services and functions of PDCP are ciphering and integrity protection and control plane data transferring.

- **Radio Link Control (RLC):** This layer is part of LTE air interface user and control planes. The main services and functionalities include protocol error detection and recovery; segmentation, reassembly and concatenation of data units; error correction through the Automatic Repeat Request(ARQ) protocol; transfer of upper layer (Protocol Data Unit)PDUs, and so on. It operates in three models:

  1. **Transparent Mode (TM):** The TM mode is used for specific purpose, for example the random access. The TM mode does not have RLC header addition, data segementation or concatenation.

  2. **Unacknowledged Mode (UM):** This mode does not require retransmission of the missing PDUs, but it allows packet loss detection, packet reordering and reassembly.

  3. **Acknowledged Mode (AM):** In this mode, in addition to the features supported by UM mode, the missing PDU is requested to retransmit.

- **Medium Access Control (MAC):** This layer is part of LTE air interface user plane and control plane. The main services and functionalities include Mapping between logical channel and transport channels; error correction through the Hybrid ARQ (H-ARQ) ; multiplexing / demultiplexing of MAC SDUs belonging to one or different logical channels; scheduling information reporting; priority handling between logical channels of one UE , priority handling between UEs by means of dynamic scheduling. This layer is also responsible for transport format selection of scheduled UEs, for example, selection of modulation format and code rate.

- **Physical Layer (PHY):** It is part of LTE air interface user plane and control plane. This layer performs the actual transmission and reception of data in forms of transport blocks. The control mechanisms in this layer include signaling of H-ARQ feedback, signaling of scheduled allocations and channel measurements.

Figure 4 below shows the user plane packet flow on different layers of radio interface protocol.

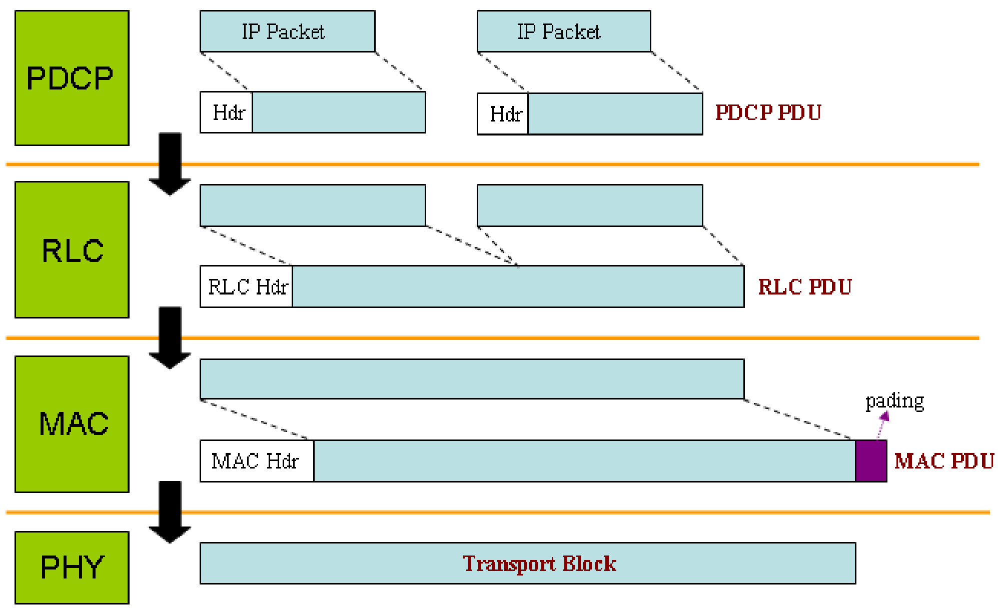


Figure 4. Packet flow in user plane [12]

From top to bottom, in the PDCP layer, the PDCP header is added to IP packet which forms a PDCP packet data unit (PDU). After that, the RLC header is added to the PDCP PDU forms the RLC PDU. It is noticeable that there is only one RLC PDU entity at the eNB and the UE per bearer [13] (The concept of ”bearer” is explained in section 2.3). Then, the RLC PDU is added with the MAC header and any extra padding as a result becomes the MAC PDU, which flows down to PHY layer becomes the PHY transport block.

2.3 EPS Bear

A bear is an IP packet flow with a defined QoS between the PDN GW and the UE. In the Evolved Packet System (EPS), the so called EPS bearers are employed for carrying the user data between the UE and the PDN Gateway or between the UE and the Serving Gateway.

The EPS bearer can consists of a radio bearer, an S1 bearer and an S5/S8 bearer. The transport of user data between the eNB and PDN Gateway takes place within a GPRS Tunneling Protocol (GTP) tunnel [6].

There is a certain Quality of Service (QoS) profile which is associated with each EPS bearer. Different packet flows with different QoS requirements will be associated with different EPS bearers, so that the network can prioritize packets accordingly.

A permanent EPS bearer is established for the lifetime of a Packet Data network (PDN) connection when a UE connects to a packet data network, so that an always-on IP connectivity with that PDN is provided. There are also additional dedicated EPS bearers which may or may not be allocated for user data transportation.

2.4 Mobility Management

There are two kinds of mobility management functions in LTE system. One is the mobility within the LTE system, which is called intra-LTE mobility. The other one is the mobility to other 3GPP systems or non-3GPP systems, it is called inter-RAT (Radio Access Technology) mobility [13]. When a UE moves between two eNBs which are connected to the same MME, the intra-LTE mobility takes place. When a UE moves from one eNB to another eNB which belongs to another Radio Access Network (RAN) attached to a different MME, the inter-RAT mobility takes place.

2.4.1 EMM and ECM states

Based on the information held by the Mobility Management Entity, there are two sets of states defined for each UE.

The two EPS Mobility Management (EMM) states:

- EMM-DEREGISTERED: The MME does not have the location information for the UE, the UE is not reachable.

- EMM-REGISTERED: The UE enters this state either due to the LTE attach procedure or due to a tracking area update (TAU) from a 2G or 3G network. The UE can be reached by pagying.

The above two EMM states describe whether or not the UE is registered in the MME and can be reached by paging.

There are also two EPS Connection Management (ECM) states, ECM-IDLE and ECM-CONNECTED. The two states describe the signaling connectivity between UE and Evolved Packet Core.

- ECM-IDLE: there is no signaling connection between UE and MME
- ECM-CONNECTED: there is a signaling connection between UE and MME. The connection consists of an RRC connection between UE and eNB, and an S1-MME connection between eNB and MME.

The main procedures for managing the end-to-end LTE system includes mobility management procedures in ECM-IDLE states, connection management procedures and mobility management procedures in ECM-CONNECTED state[14], which is also known as handovers. Figure 5 shows the end to end procedures of LTE system.

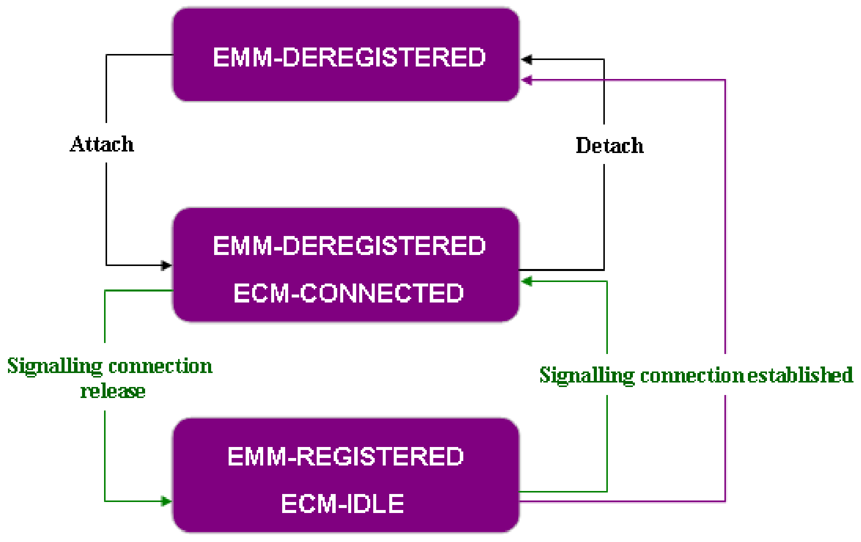


Figure 5. LTE end to end procedures

2.4.2 Mobility management in ECM-IDLE state

The mobility management functions are needed for keeping tracking of the current location of a UE. The basic procedures of mobility management in ECM-IDLE state include tracking area update and paging. Tracking area update is needed when the mobile terminal moves to a tracking area in which it is not registered.

The network coverage area is divided into cells; each cell belongs to a single tracking area (TA). The network needs to know where the location of a UE is if the network wants to communicate with the UE that is in ECM-IDLE state. For those cells that belong to the tracking area in which the UE in ECM-IDEL state has been registered can reach this UE. On the other hand, when a UE moves to a tracking area in which it is not registered, a tracking area update will be performed to ensure the UE can be reached.

A mobile terminal in ECM-IDLE state can only be reached through paging. The mobile terminal is paged in the tracking area in which it is registered.

Paging is like a initiate contact with UE. For example, when there is user data arrived in S-GW, the S-GW will send request to MME, then MME will page the UE to which the data in S-GW should be sent. Basically, the MME sends a paging message to every eNB in the UE registered tracking area.

In a paging process, the eNB broadcast the paging indication message in order to indicate the paging group. The indication message includes information about how the paging message can be read and what physical resources have been allocated for it.

The UEs are allocated to paging groups based on UE identifier, for example IMSI. The UE will read the full paging message only when it detected that its paging group is being paged. After that, when a UE noticed that it is being paged, it starts to transfer from ECM-IDLE to ECM-CONNECTED state, and establish an signaling connection to MME.

2.4.3 Tracking Area Update

When there is a need to update tracking area , for example, the UE detects that the current Tracking Area (TA) indicated on the broadcast channel is not on the list of TAs in which the UE has registered. Then, the UE and eNB firstly need to establish an RRC connection by using random access procedure.

After the RRC connection between UE and eNB has been established, the UE then sends a TAU request to the MME. The TAU request message includes the last visited TA which can be used to produce a new list of TAs and send to UE.

Then, the eNB forwards the TAU request to MME.

The MME now could perform an authentication based on the data acquired from HSS. Once the MME accepts the TAU request, it sends a TAU acceptance message to UE which includes a new list of tracking area in which the UE is now registered.

A so called MME relocation procedure is taking place during Tracking Area Update process. There are generally two MMEs be involved. That's the old MME handles the previous tracking area, while the new MME handles the new tracking area. In MME relocation, there is a signaling between two MMEs, between the new MME and HSS for adding new information, between old MME and HSS for removing old information.

2.5 Connection management

There are four connection management procedures which are random access, LTE attach, user data connection set up and the user data connection release.

2.5.1 Random access procedure

The UE performs random access procedure to initiate the communication with the network. In LTE system, there are two random access procedures have been defined.

The first type of random access procedure is the so called contention based random access, which is used when the UE starts the LTE attach procedure, or when the UE is in ECM-IDLE state and wishes to contact the network. The other type of random access procedure is non-contention based random access which is used when the UE is in ECM-CONNECTED state. For example, when the network requests the UE to perform a handover to another cell, the non-contention based random access is used [13].

The contention based random access procedure consists of four steps which are indicated in *Figure 6.*

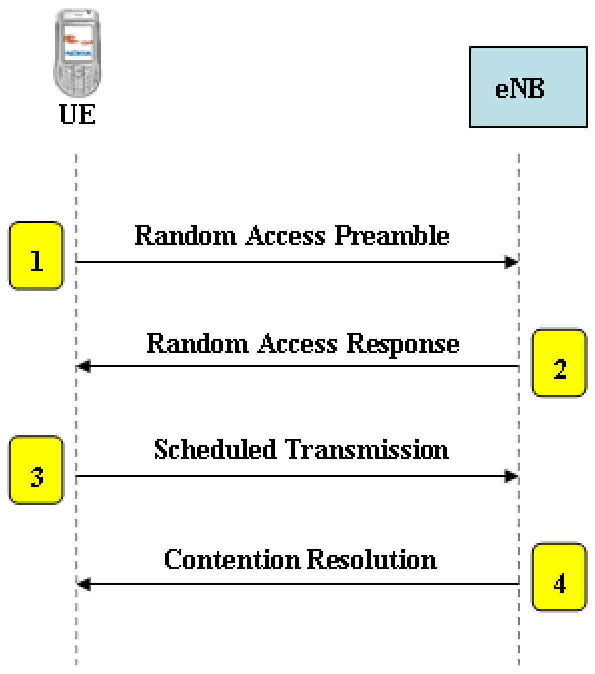


Figure 6. The contention based random access procedure

In step one, the UE randomly selects a random access preamble based on the information obtained from eNB. Open-loop power control is used to determine the initial transmit power level.

In step two, based on the received random access preamble, the eNB sends a response using the Physical Downlink Shared Channel (PDSCH) where multiple random access responses may be multiplexed into one shared data channel control message. Each response contains an uplink scheduling grant, timing advance information, and an assignment of a temporary Cell Radio Network Temporary Identifier (C-RNTI). The timing advance information is used by the mobile to time-synchronize its uplink transmission [15].

The UE is uplink synchronized after step two, it is able to transmit additional message on scheduled UL-SCH. It is possible that multiple UEs select the same preamble which causes a collision. To resolve the contention for access, the UE that detects a random access preamble transmits a message which contains a terminal identity. If the UE is connected to a cell, Cell Radio Network Temporary Identifier(C-RNTI) will be used, which is the unique UE ID at the cell level. In order to improve the transmission reliability, the H-ARQ protocol is supported in this step [15].

The forth step is called Contention Resolution, in which the eNB transmits the contention-resolution message on the DL-SCH, which contains the identity of the winning UE. The UE that observes a match between this identity and the identity transmitted in step three declares a success and completes its random access procedure. The UE with successful access will transmit an H-ARQ acknowledgement [15].

2.5.2 LTE attach procedure

When a UE just power on, it intend to register with the EPC network, which means it will move from the EMM-DEREGISTERED state to the MME-REGISTERED state. In this case, the LTE attach procedure is performed.

Figure 7 shows the LTE attach procedure. [16]Firstly, a Radio Resource Control (RRC) signaling connection between the UE and eNB is needed. In order to establish this connection, a random access procedure is necessary. Then, an Attach Request message is sent to the eNB by using the RRC signaling connection. The Attach Request message is then forwarded to the MME. An authentication is probably performed by MME at this stage. The MME then contacts the Home Subscriber Server (HSS) for the user’s subscription data, with this data, MME can create a context for the UE. During the LTE attach procedure, the default EPS bearer is established, so that the UE enters the ECM-CONNECTED state.

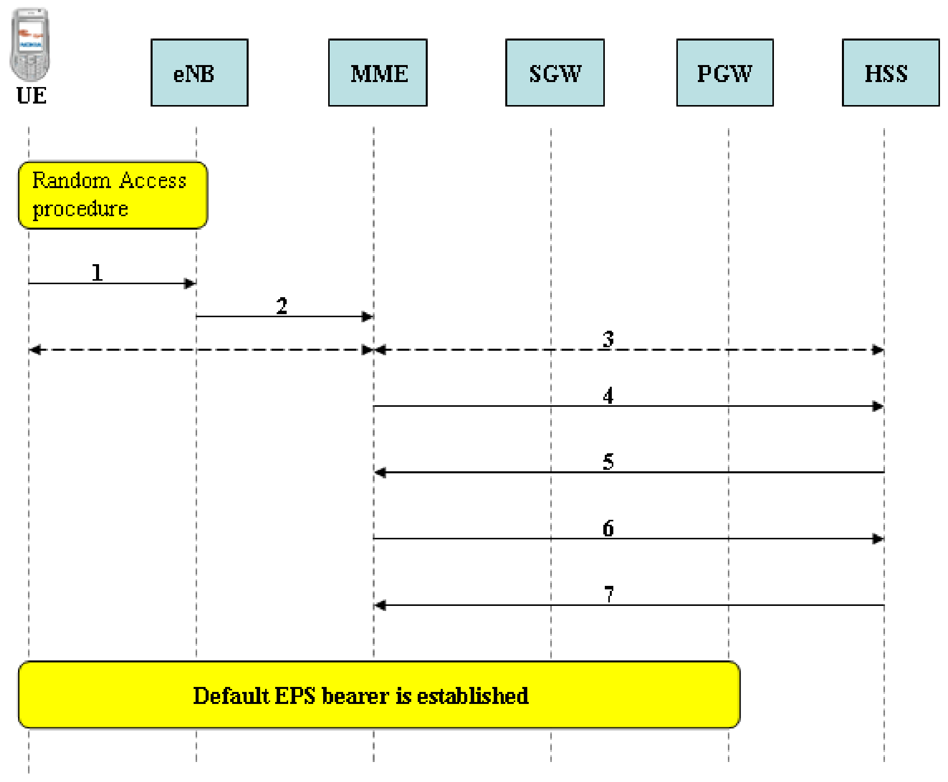


Figure 7. LTE attach procedure [16]. The processes from 1 to 7 are explained below.

1. The UE sends a Non Access Stratum (NAS) Attach Request message to eNB encapsulated in an RRC message.

2. The eNB forwards the NAS Attach Request message to a selected MME encapsulated in an S1AP Initial UE message. The selected MME serves the UE.

3. An authentication based on data from HSS is performed if the network does not know any context for the UE.

4. The MME informs the HSS that it is now serving the UE by sending a location update message.

5. The HSS sends the UE's subscription data to the MME.

6. The MME acknowledges the above HSS action.

7. The update location message which is sent by MME is received and acknowledged by HSS.

2.5.3 UE initiated communication

If UE has uplink data to be sent, a procedure is initiated by the UE. Generally, the following steps are performed during this procedure [16]:

Firstly, an RRC connection between the UE and eNB is established. Then, the UE sends a Service Request message to the MME, partly using the RRC connection. The MME may perform authentication if necessary. After that, the S1 bearers are established, after which the radio bearers are established. Finally, the user data is sent in the uplink.

2.5.4 Connection release

From UE's point of view release a connection means to transit from ECM-CONNECTED state to ECM-IDLE state. The following steps are performed:

1. The eNB send a UE Context Release Request to MME to release the signaling and user plane connection with UE.

2. The MME sends an S1AP Update Bearer Request message to S-GW informing the S-GW that the UE will be moved to ECM-IDLE state.

3. The S-GW releases the eNB related information, for example the Tunnel Endpoint Identifiers (TEID) of the downlink GTP tunnel endpoints in eNB. The S-GW acknowledges the release to MME. UE now can only be reached through paging. The downlink data for the UE arrives at S-GW is go to buffer now.

4. The MME sends an S1AP UE Context Release Command message to eNB.

5. Release the radio bearer over the air interface

6. The eNB release the S-GW related information, for example the Tunnel Endpoint Identifiers of the uplink GTP tunnel endpoints in S-GW. The eNB acknowledges the release to the MME by sending an S1AP UE Context Release Complete message.

Chapter 3 Test tools, environment and methodology

For telecommunication infrastructure manufacturer, in order to ensure positive return for the business investment on all mobile network components, the phases of development and deployment of the mobile network components require thorough testing at the functional and performance level.

As a tool for testing telecommunication network in this thesis work, a terminal is introduced which provides an interface with which developers can define a series of network actions and execute them sequentially so that the network traffic can be generated and the network status can be emulated. Thus, the network profile is defined through this terminal. In considered test cases, the terminal been used is Tektronix G35 protocol tester.

This chapter introduces the protocol testing tool Tektronix G35 and employed element platform ATCA. The Robot Framework and Agile methodology are also presented in this chapter.

3.1 Tektronix G35 protocol tester

There is a lot of data exchanged among the network elements, governed by certain rules which are called protocols. These protocols are defined in 3GPP technical specification. A protocol tester normally has abilities, for example, to generate traffic, display data in multiple views, detect errors automatically, determine the root cause of errors and generate timing diagrams, etc.

Protocol testing is an essential step towards commercialization of standards-based network element products. It helps to ensure that products from different manufacturers will operate together properly ("interoperate") and satisfy customer expectations. Besides, by utilizing a protocol tester during network element development, the time needed for the product entering market will be reduced when the productivity of testing is maximized.

The Tektronix G35 protocol tester [9] simulates protocols, generates traffic, provides interoperability and performance testing of LTE air-interface and fix line interfaces under various radio conditions through a fully integrated test system. It provides functional tests of specific protocol layers, network nodes or subsystems in order to verify correct implementation. It offers flexible protocol configuration and a wide range of programming and scripting tools that allow developer to define test cases.

The G35 provides specific capacity testing applications, for example, with Tektronix G35, you can verify the ability of specific network nodes (such as SGSN) to handle the signaling traffic related to a large number of subscribers. The G35 can also provide automated testing to efficiently verify a protocol layer’s or network node’s functionality using standardized test suites [9].

3.2 MME ATCA platform

“Our aim is to decouple hardware and software, making our network applications more hardware independent. Decoupling creates a more open environment that gives service providers greater flexibility to meet their hardware requirements and reduce their operational costs.” said Michael Clever, head of voice and IP transformation at Nokia Siemens Networks [18].

3.2.1 ATCA hardware architecture

The full name of ATCA is Advanced Telecom Computing Architecture which usually referred to as AdvancedTCA or ATCA. ATCA is a specification from PCI Industrial Computer Manufactures Group (PICMG)[19]. The specification is an open standard which is created with the work by the major global computer manufactures and telecommunication equipment manufactures. The availability, reliability and serviceability are three significant aspects for telecommunication hardware which must meet strict standards set by national and international industrial organizations. PICMG [20] defines ATCA as an new generation architecture for building high end equipment.

In ATCA shelf, there are 16 slots for computer units from which 2 slots are reserved for HUB blades with the purpose of internal switching and communication. The boards of ATCA are 322.25mm high and 280mm deep which are designed with careful considerations about cooling, front panel space, backplane size and rear panel I/O requirements [21]. Figure 8 shows an ATCA hardware.

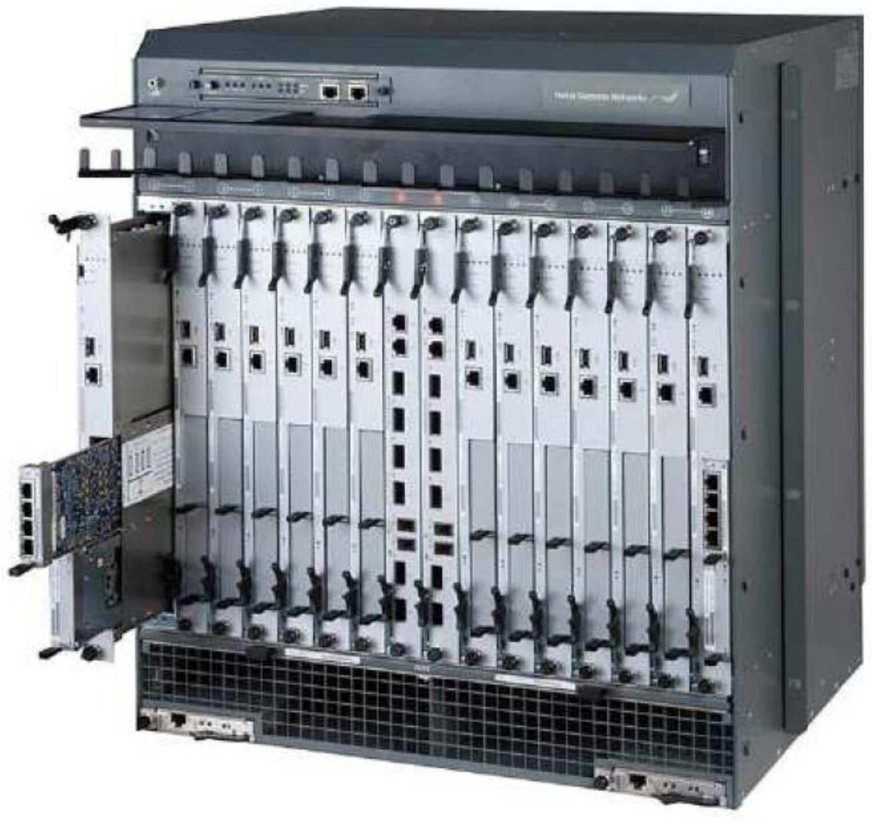


Figure 8. ATCA hardware [22]

ATCA delivers the benefits of modularity, scalability and flexibility. It defines open standards-based guidelines for the design and manufacture of next-generation carrier-grade telecom hardware. Practically, those equipments based on ATCA conform to standardized rack size and power supply guideline which makes it easier for operators to plan for deployment, regardless of the actual detailed makeup of the equipment. For example, Nokia Siemens Networks uses this hardware architecture in all its core network solutions.

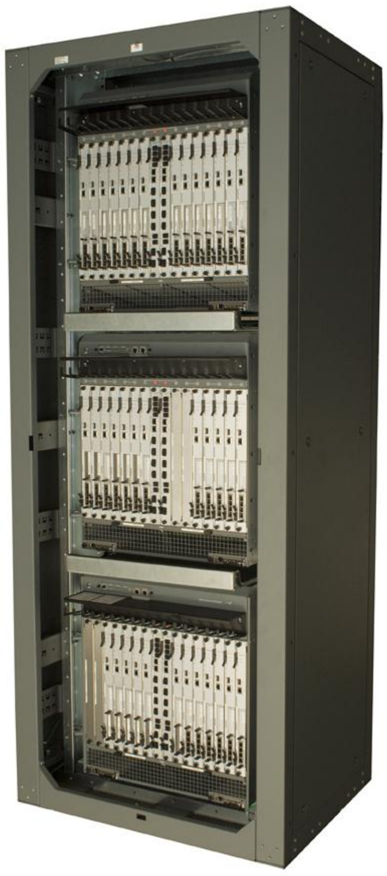


Figure 9. NSN ATCA rack [22]

By running core network software on ATCA hardware, network manufacturer delivers to service provider the ability to support a great number of subscribers with a smaller equipment footprint and significant energy saving. By using ATCA platform, service provider can also combine several core network applications into one hardware rack which helps to reduce the space required, the same hardware can also be used in the spare part of ATCA architecture. With this kind of hardware architecture, the management and maintenance of hardware are improved. Figure 9 shows an ATCA rack that contains three ATCA hardware units.

3.2.2 Key functional units in ATCA MME

The studied core network element Mobility Management Entity (MME) is based on ATCA hardware. There are five key functional units in MME which are Control Plane Processing Unit (CPPU), Signaling and Mobility Management Unit (SMMU), IP Director Unit (IPDU), Marker and Charging Unit (MCHU), Operation and Maintenance Unit (OMU).

CPPU is responsible for transaction based mobility management. SMMU is responsible for storing information of visiting subscribers into the visiting subscriber database, it provides S6a interface to Home Subscriber Server (HSS). IPDU is used to balance the loads and connectivity. MCHU offers statistics functions. OMU handles operation and maintenance functions[22].

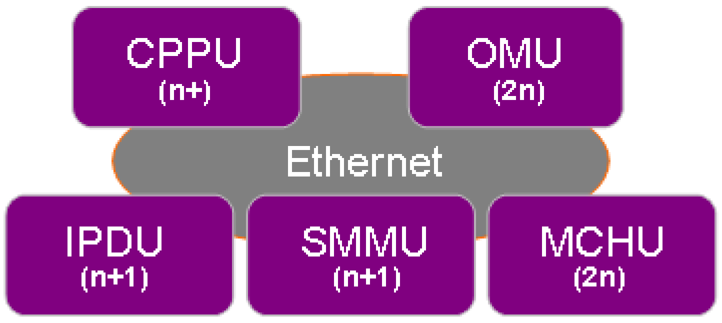


Figure 10. Five key units of studied MME

Actually, each key unit is just an category of units, that's each key unit category has a group of units. The units of each key unit are paired with working status units and spare status units. Figure 10 shows the five key unit categories (groups) of studied MME element. The string inside parenthesis indicates the number of pairs of working and spare units for each unit category. As figure 10 shows, CPPU contains "*n*+" units which means there should be at least one unit of CPPU that is in working status for CPPU unit group. The rest units of CPPU can be either in working or spare status. The IPDU and SMMU are both indicated with "*n*+*1*" , which means there has to be at least one working unit and one spare unit for them. The "*2n*" of OMU and

MCHU indicates that there have to be n working units and n spare units for OMU and MCHU.

Actually, the SMMU was later removed from MME, and replaced by Mobility Management and Database Unit (MMDU), which also handles subscribers database, besides, it also controls the EPS mobility management and EPS system management level functions for LTE subscribers.

The ATCA based MME has the capability to support 10M subscribers and 30M sessions (bearers) in order to accommodate the increasing number of subscribers and session in the network.

3.3 Robot framework

Robot framework is a generic test automation framework. It is open source software whose license is under Apache 2.0.

This framework is implemented in Python, and can be extended with Python, Java or other languages supported via XML-RPC based remote interface. It utilizes the keyword driven testing approach to create test cases. User can create new keywords from existing ones. The high level architecture of Robot Framework based test is illustrated in *Figure 11*.

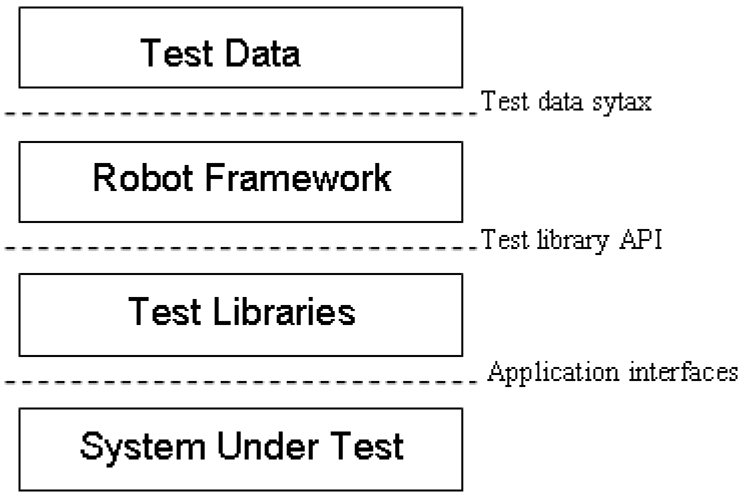


Figure 11. High level architecture of Robot Framework based test

Robot framework enables text based tests in a text editor. The test cases have a column-oriented syntax. Robot framework utilizes test libraries with test library integration, through which the robot framework interacts with the system under test.

To create test case in a uniform way, robot framework enables tabular [23] syntax for creating it. User uses keywords to construct the test cases. There are robot framework build-in keywords, user can also import keywords from test libraries or create own keywords [24]. *Table 1* shows a test case example of Robot Framework using tabular syntax.

Table 1. Test case example of Robot Framework

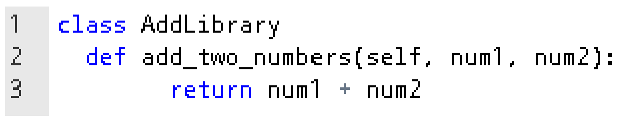
Test Case	**Action**	**Argument**	**Argument**
Valid login	Input Name	First argument	Second argument
Setting variables	Do something	First argement	Second argument
	${value}=	Get some value	

The above table shows the basic syntax of a test case. There are step definitions, each row is one step. They are executed row by row during the execution of the test case. The first column of the table indicates the test case name, while the second column contains either keyword or variables (like ” *${value}=*” in above table) which holds the return value of one keyword. The columns after keyword name contain possible arguments to the keyword. With this keyword-driven style, the test cases do not need to know what is happening underneath besides keywords. You can consider each keyword as one function call which accepts arguments.

Each variable in robot framework has a scope. It can be defined as global variable, which is recommended to use all capital letters [24], for example *${PATH}*. It can also be defined as a test suite specific or test case specific variable that is only available in certain test suites or test cases. Variables can be defined directly in test definitions or can be input from command line or can be from external resource file.

So, different test scenarios can be described into test cases by using keywords. All test cases are encapsulated in one test suite. Each test scenario could actually be split into more detailed sub-scenarios, so that more descriptive keywords can be used to construct the test case. At the beginning of writing test cases, the build-in keywords can be used, but as the test cases become more and more specific, the build-in keywords are quite limited that there is a need to create own key words. The self-created keywords are called *user keywords* to differentiate them from lowest level *library keywords*. In robot framework, creating user keyword is easy as long as you know Python or other programming languages [24]. If using Python, each keyword is actually one Python function call when running the test case.

How to create a user keyword of robot framework? Firstly, we need to know what functions we need for the test. Then, we can implement each function in a programming language (e.g. Python). After that, we should add the programmed class in ”*Library*” of settings. Finally, we can call the function implemented in the external class as a keyword in robot framework syntax. For example, figure 12 shows a simple code snippet of Python which creates a function named *add_two_numbers*. It simply takes two numeric arguments, and returns the summation of the two arguments.



```
class AddLibrary
  def add_two_numbers(self, num1, num2):
        return num1 + num2
```

Figure 12. A Python function used as keyword

Table 2 shows a test case which utilizes the above Python code. In the ”*Settings*” part of Robot Framework, we add the python class as a library to ”*Library*” section of the test case. Then, in test case implementation, we call the python function by replace underscores of the python function name with space, and capitalize the first letter of each word. The return value is assigned to an *${result}* variable.

Table 2. A test case with user keyword

Settings	**value**
Library	AddLibrary

Test case	**Action**	**Arguments**
Example test case	${result}= Add Two Numbers Should Be Equal	

In the above example, we created an ”*Add Two Numbers*” user keyword of robot framework and used it in one test case. It is quite easy and intuitive.

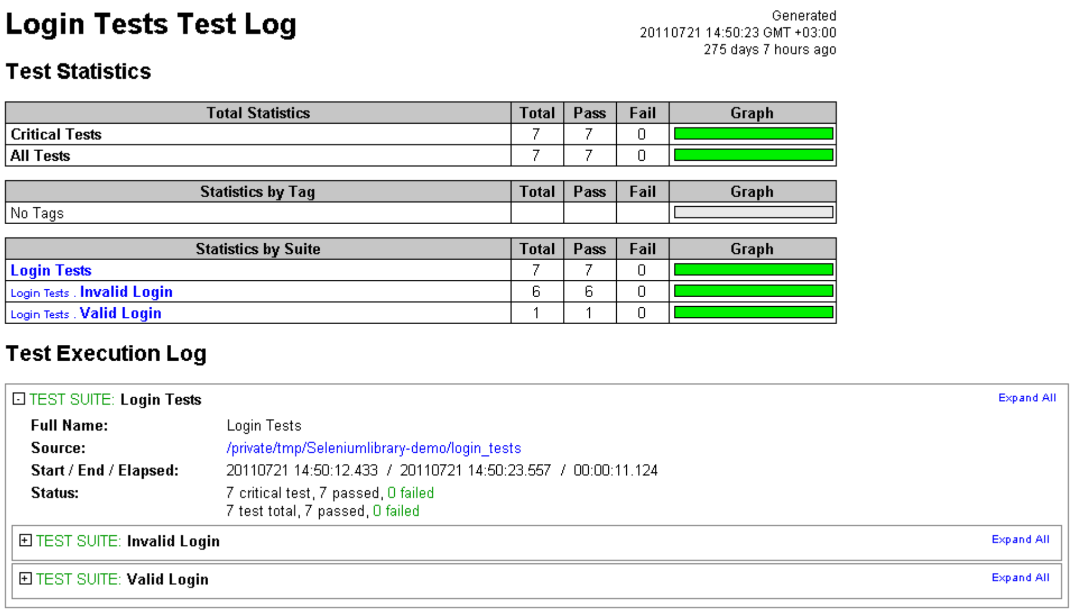
In general, with robot framework, we have high level tests that specify the general work flows, while underneath, we have keywords that deal with the details of how the implantations of functions are called. All these make robot framework a handy tool.

In each test suite, there are *Setup* and *Teardown* parts. The *Setup* part defines the initial phase of the test suite which allows actions for initialization and preparation of test cases. For example, in *Setup*, we can check if the database connection is established, if the database server is started before real tests run. The *Teardown* part is the final phase of a test suite, in which we can define some final actions for the test suite. Table 3 shows an example of setup and teardown settings. The availability of *Setup* and *Teardown* phases make it easy to keep the real tests focused. Test cases only focus on real tests definitions, leave other things be handled by the context. The individual test cases can also have their own setup and/or teardown.

Table 3. Setup and Teardown settings

Setting	Value	Value	Value
Test Setup	Check db connection	db name	
Test Teardown	Close db connection	db name	

After test cases execution finished, there are output files be created which contain test results in XML format, based on which a log file and a report file are generated. The detailed information of executed test cases can be seen from log file. The log file is an important resource for tester to investigate, especially when problems rise up during test cases run. *Figure 13* shows the test log file.



Login Tests Test Log

Generated
20110721 14:50:23 GMT +03:00
275 days 7 hours ago

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	7	7	0	🟩
All Tests	7	7	0	🟩

Statistics by Tag	Total	Pass	Fail	Graph
No Tags				

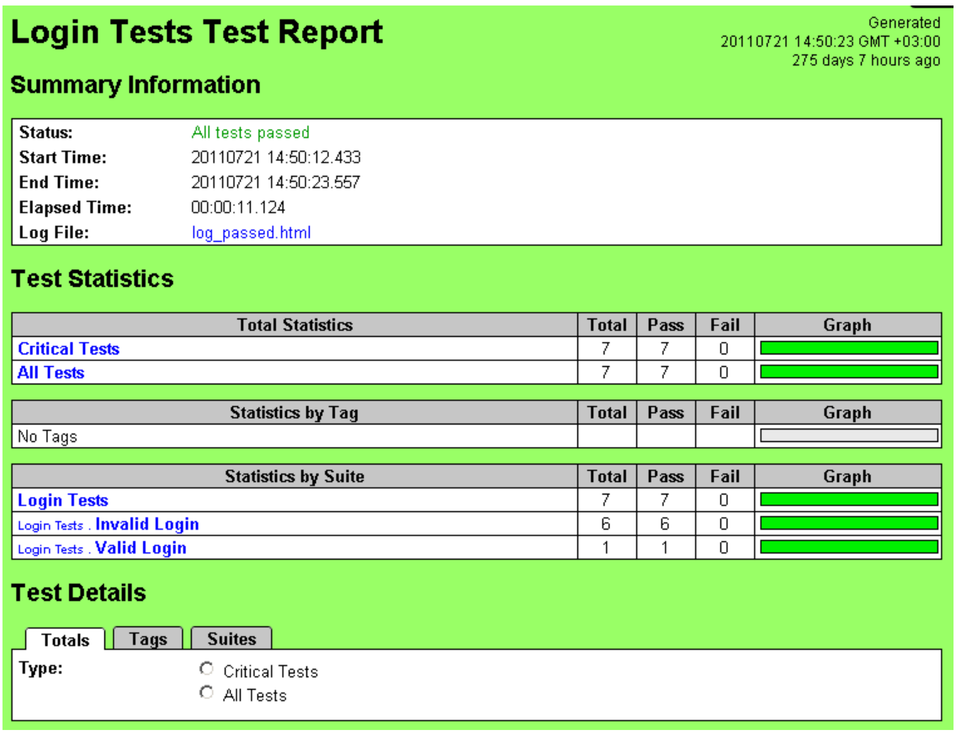
Statistics by Suite	Total	Pass	Fail	Graph
Login Tests	7	7	0	🟩
Login Tests . Invalid Login	6	6	0	🟩
Login Tests . Valid Login	1	1	0	🟩

Test Execution Log

```
⊟ TEST SUITE: Login Tests                                                    Expand All
    Full Name:            Login Tests
    Source:               /private/tmp/Seleniumlibrary-demo/login_tests
    Start / End / Elapsed: 20110721 14:50:12.433 / 20110721 14:50:23.557 / 00:00:11.124
    Status:               7 critical test, 7 passed, 0 failed
                          7 test total, 7 passed, 0 failed
  ⊞ TEST SUITE: Invalid Login                                                Expand All
  ⊞ TEST SUITE: Valid Login                                                  Expand All
```

Figure 13. Test log file

Report file is a high-level overview of test results in HTML format. It contains links to log file to provide easy access for detailed information. With report file, it is easy to see both the overall test execution status and the detailed step information. If all critical tests pass, the background of the report is green (like *figure 14* shows), otherwise it is red (like *figure 15* shows).



Login Tests Test Report

Generated
20110721 14:50:23 GMT +03:00
275 days 7 hours ago

Summary Information

Status:	All tests passed
Start Time:	20110721 14:50:12.433
End Time:	20110721 14:50:23.557
Elapsed Time:	00:00:11.124
Log File:	log_passed.html

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	7	7	0	🟩
All Tests	7	7	0	🟩

Statistics by Tag	Total	Pass	Fail	Graph
No Tags				□

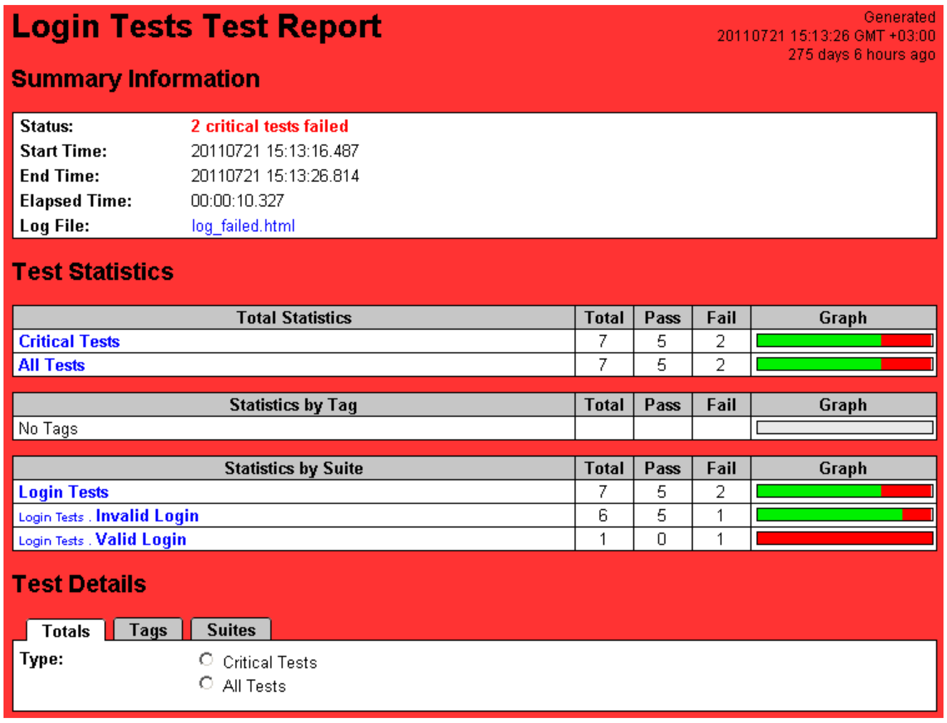
Statistics by Suite	Total	Pass	Fail	Graph
Login Tests	7	7	0	🟩
Login Tests . Invalid Login	6	6	0	🟩
Login Tests . Valid Login	1	1	0	🟩

Test Details

Totals | Tags | Suites

Type: ○ Critical Tests ○ All Tests

Figure 14. Test report file (All pass)



Login Tests Test Report

Generated
20110721 15:13:26 GMT +03:00
275 days 6 hours ago

Summary Information

Status:	2 critical tests failed
Start Time:	20110721 15:13:16.487
End Time:	20110721 15:13:26.814
Elapsed Time:	00:00:10.327
Log File:	log_failed.html

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	7	5	2	🟩🟥
All Tests	7	5	2	🟩🟥

Statistics by Tag	Total	Pass	Fail	Graph
No Tags				□

Statistics by Suite	Total	Pass	Fail	Graph
Login Tests	7	5	2	🟩🟥
Login Tests . Invalid Login	6	5	1	🟩🟥
Login Tests . Valid Login	1	0	1	🟥

Test Details

Totals | Tags | Suites

Type: ○ Critical Tests ○ All Tests

Figure 15. Test report file (with failure tests)

3.4 Agile methodology

Nowadays, we often hear the word ”Agile” in software development. What is ”Agile” in this circumstance? The answer can be seen from the manifesto for Agile software development [25]:

“We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.”

In software development, perhaps the biggest problem is changing requirement. A project that is readily accepting changes can cost less than that without accepting changes. Nowadays, the Agile development model becomes popular in software development because of its efficiency and flexibility for accepting changes. The main principle of Agile methodology is developing working software while keeping customer satisfaction in mind. In Agile way, the working software can be delivered frequently with shorter timescale. The customer is able to make changes on the requirement during each review of the delivered software.

Generally, *Agile* process is an iterative and incremental model [26]. The whole development process is divided into development cycles, where in each cycle there is the intention of doing only a small portion of the whole software product. The tasks in each cycle are supposed to be completed on time to be ready for the next iteration round. In case if the tasks are not finished at the end of the current cycle, they will be moved to the next iteration round. In this way, the team always try to take a portion of work which they estimated that they are able to finish during one development cycle.

There are a number of agile methods, of which the most well-known one is probably the ”Scrum”. The Scrum model is more concentrated in project management than software development practices. Its process is aimed at flexibility, adaptability and productivity [27]. The assumption in Scrum is that during the software development, the requirements, time frame, resource and technologies will most probably be changing during the process. It makes the Scrum an efficient project management tool in situation where it is difficult to plan ahead. In Scrum, the work is organized into short development cycle called Sprint, which normally could be one week or two weeks in one Sprint. During the development, the features of the software are chosen and prioritized together with customer. A Daily Scrum is conducted in order to get immediate feedbacks from customer and the development team. There are also planning meeting in the beginning of each Sprint and a Review meeting at the end of each Sprint [27].

Chapter 4 Automated software testing

Briefly, software testing could be understood as a procedure of finding faults and defects in a software application so that the problems could be fixed by software developers. With this procedure, the quality of the software application is guaranteed. This chapter firstly talks about what automated testing is, after which an introduction of the life-cycle of automated test and the concept of automated regression testing are presented.

4.1 Definition of automated testing

The new challenge of today's software development is that the time schedule is shrinking, software development needs to be performed within a short period and within minimum resources, that's to do more with less. Software testing is a required step guarantees the software application's quality. Organizations would like to do the test adequately, meanwhile, as quickly and thoroughly as possible. To achieve this goal, organizations nowadays choose to use automated testing.

In manual testing, a person who plays a tester role initiates a test, interacts with it. The tester needs to interpret, analyze and document the test results. A brief description of automated testing is that a tester-free procedure runs a suit of test cases, automatically generates the test results, makes analysis and reports the pass and fail of each test case for the test run.

In order to closely reflect and meet user's needs and preferences, software development is conducted in the way of continuous changing through each software build [28]. Software testing is performed in an iterative manner. Each new build of application is accompanied by a considerable number of new tests as well as reworks to the existing test scripts. In this environment, automated software testing becomes an important control mechanism for the software application which is under the continual changes during development. Automated testing ensures the stability and accuracy of the software through each build.

The main question in modern software development is not whether to use automated testing or manual testing. Instead, one should consider of the level of automation to use and which level of tests is feasible to automate [31].

The initial step in test automation is to understand what software application is going to be tested, identify the features we would like to test, and identify the environment the software is running upon. Besides, we should understand the features of testware and tools for test automation. It is not necessary to test everything of the software, it is important to identify which component of the software should be included in automation test, and which should be excluded. The most important features that to be tested should be defined [32].

Test environment and interfaces must be considered. Under what environment the tests are supposed to run? What interfaces of utilities would be used in tests? What is the immediate technical environment? The answers to these questions are important for test automation since they provide the facilities and constrains on practical approach. If there are requirements to support the test automation in different platforms and environments, we should take into account that tools and utilities could be available across different platforms but have incompatible interfaces.

When running a test, there are input data and output result. The input data could be keystrokes or a simple user input. It is also possible that the input data of an automation test is one keystroke which triggers a function to load external data from a remote database. The output result could be counters, statistics or graphs, which must draw a proper conclusion for the tests. So, the definition of input data and expected output results is also an important issue.

The existence and availability of testware elements and tools defines the scope of planned automation tasks [32]. The testware elements here mean the documentation, program and associated procedures for test activities. The tools means the operating system utilities, test selection and control programs, etc. It is common to have a requirement to include existing testware in automation test [32], which should of course be carefully considered since if the existing testware can cover all the targeting level of automation test for an application, there would be no need to develop our own automation test.

In manual test, tester will check the result data and program states, make investigations on them, and judge whether the test is passed or failed. In automated testing, we need to implement this procedure programmatically. That’s to verify the results, to identify the important indications of errors and systematically checking them after each test. More preferably, the automated test could generate a final report to show the overall view of test results.

4.2 Automated test life cycle

Automated testing provides the benefits of improving product quality and minimizing project schedule and effort. In order to achieve these benefits, test engineer should be involved in analysis and design activities, so that test engineers get a clear view of customer needs. The test team can then develop an appropriate test environment and test design. Besides, by involving test engineers in analysis and design activities, errors can be detected early without migrating the errors from requirement specifications to design.

The Automated Test Life-cycle Methodology (ATLM) [28] is introduced in automated testing. It represents a structured approach for the implementation and performance of automated testing. This approach mirrors the benefits of modern rapid application development.

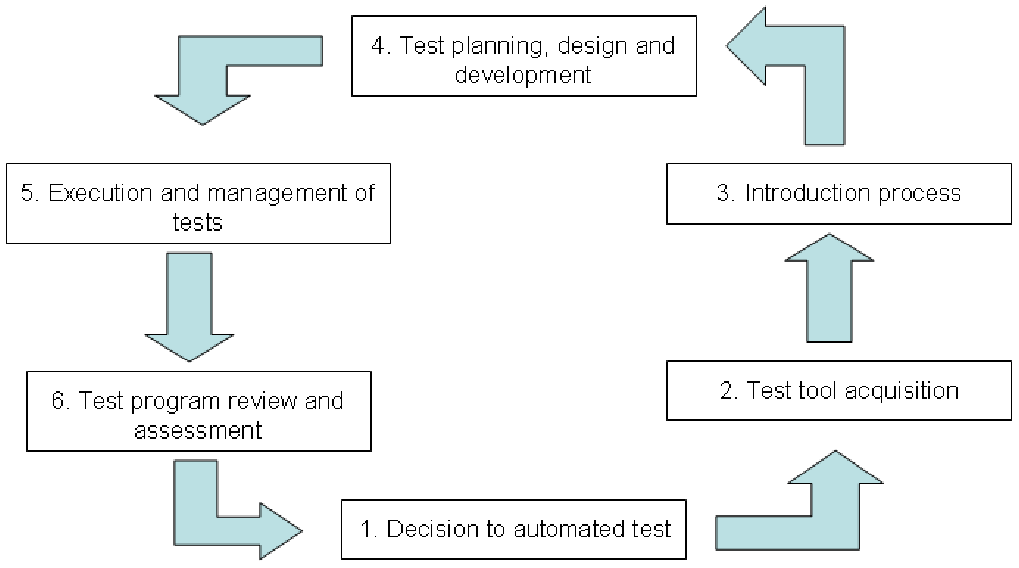

4. Test planning, design and development
5. Execution and management of tests
3. Introduction process
6. Test program review and assessment
2. Test tool acquisition
1. Decision to automated test

Figure 16. Automated Test Life-cycle Methodology (ATLM) [28]

By applying ATLM, test engineers can be involved early in the system life cycle from business analysis to design and development, which enables the test team to conduct a thorough review of requirement specification, gain a complete understanding of business needs and requirement so that a more appropriate test environment can be designed, and a more rigorous test design can be generated.

The first phase of ATLM is the decision to automated test. So, what does the term ”Decision” here actually mean? Basically, it means the test team should understand the expectations of automated testing. Moreover, the test team can outline the potential benefits of automated testing. At this phase, an approach for developing a test tool proposal is outlined, which will be helpful in acquiring management support.

The next phase is to select a proper automated test tool. The selected tool should fit the requirements of the organization and allows the organization to make the most use of the tool. There is normally a process takes place in test team for test tool evaluation and selection. In this thesis work, the test tool has already been defined by NSN.

The third phase is to introduce the automated test to a new project to which the test team is targeting. A test process analysis is supposed to take place in order to ensure that an overall test process and strategy are in place. The test team defines the test process metrics so as to allow process improvement. The definitions of test goal, objectives and strategies must be done in this phase.

In test planning phase, the test team defines the hardware, software and network requirements which support the test environment. Besides, the test team identifies the test procedure, test data requirement, performance measurement requirement and a defect tracking procedure. A preliminary test schedule should also be defined. Setting up the test environment is part of testing planning, which includes the installations of hardware, software and network resources.

Test design addresses the definition of test scenarios, the number of tests to be performed, and how those tests should be approached. The development of the designed tests consumes most of the time. The test suit and its test cases are implemented. Each test case should be reusable, repeatable, and maintainable [29].

After each execution of test scripts, an refinement could be made according to the test procedure execution schedule. An evaluation activity of test execution outcomes should be conducted too [28]. If there are system problems, a report should be generated which describes the where goes wrong. Finally, the test team should perform regression tests, and track the problems.

Test review and assessment should be performed throughout the testing life cycle, thereby allowing continuous improvement. In order to improve the test process, metrics need to be evaluated and a final review and assessment need to be conducted.

4.3 Automated regression testing

The automation of test activities provides its greatest value in instances where test scripts are repeated or where test script subroutines are created and then invoked repeatedly by a number of test scripts. During the development and integration of product, such testing, where reusable scripts may be run a great number of times, offers a significant payback. As an important control mechanism, automated testing ensures the accuracy and stability of the target system or product (e.g. software) [28].

Regression testing at the system test level seeks to verify that the functions of a modified system or software product perform the same as specified, the operation of the system or product has not encountered any unintended change [29]. By allowing automated testing for regression testing, the regression testing can be executed in a more efficient manner.

In practice, regression testing means either that, the functionalities which are working before any changes have been made to the software, are still working as expected, or means that the test cases which used to pass before making changes are still passed.

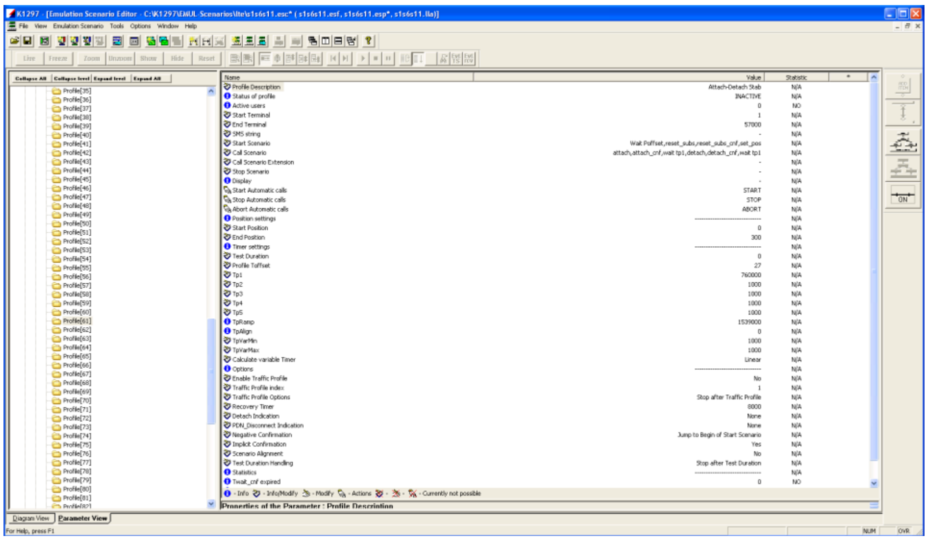
In this thesis work, we can consider our test suite as a client, and the platform we are testing as a server. Regards to a client-server or Web environment, the tests can include many types of assessments, for example, functional requirement testing, server performance testing, user interface testing, unit testing, integration testing, system load performance testing. This thesis work is more focusing on functional requirement testing, server performance testing and load performance testing, which can be performed in a more efficient, repeatable manner.

Chapter 5 Automated Tektronix control and MME regression testing

How can we test one element of the core network? The answer could be various depending on from which perspective you consider it, however, one thing should definitely be included in the answer, that's a method for emulating the different status of the network, so that the performance of the network can be evaluated. To facilitate this kind of emulation, telecommunication network manufactures normally introduce a terminal which provides either an interface for developers to program with or a user interface with which developers can configure the network profile. This thesis work uses Tektronix G35 tester as the terminal.

5.1 Traffic profile in Tektronix G35

The general idea is that G35 tester generates network traffic, which will be monitored in the Mobility Management Entity (MME) of core network. The performances of MME's s1, s6 and s11 interfaces are evaluated finally. Thus, the first thing to do is to setup G35 to generate network traffic. G35 tester provides an *Emulation Scenario Editor* user interface to facilitate the configuration of network traffic profile.



K1297 - [Emulation Scenario Editor - C:\K1297\EMUL-Scenarios\lte\s1s6s11.esc* (s1s6s11.esf, s1s6s11.esp*, s1s6s11.lla)]

Name	Value	Statistic
Profile Description	Attach-Detach Stab	N/A
Status of profile	INACTIVE	N/A
Active users	0	NO
Start Terminal	1	N/A
End Terminal	57000	N/A
SMS string	-	N/A
Start Scenario	Wait Poffset,reset_subs,reset_subs_cnf,set_pos	N/A
Call Scenario	attach,attach_cnf,wait tp1,detach,detach_cnf,wait tp1	N/A
Call Scenario Extension	-	N/A
Stop Scenario	-	N/A
Display	-	N/A
Start Automatic calls	START	N/A
Stop Automatic calls	STOP	N/A
Abort Automatic calls	ABORT	N/A
Position settings	------------------------------	N/A
Start Position	0	N/A
End Position	300	N/A
Timer settings	------------------------------	N/A
Test Duration	0	N/A
Profile Toffset	27	N/A
Tp1	760000	N/A
Tp2	1000	N/A
Tp3	1000	N/A
Tp4	1000	N/A
Tp5	1000	N/A
TpRamp	1539000	N/A
TpAlign	0	N/A
TpVarMin	1000	N/A
TpVarMax	1000	N/A
Calculate variable Timer	Linear	N/A
Options	------------------------------	N/A
Enable Traffic Profile	No	N/A
Traffic Profile index	1	N/A
Traffic Profile Options	Stop after Traffic Profile	N/A
Recovery Timer	8000	N/A
Detach Indication	None	N/A
PDN_Disconnect Indication	None	N/A
Negative Confirmation	Jump to Begin of Start Scenario	N/A
Implicit Confirmation	Yes	N/A
Scenario Alignment	No	N/A
Test Duration Handling	Stop after Test Duration	N/A
Statistics	------------------------------	N/A
Twait_cnf expired	0	NO

ⓘ - Info - Info/Modify - Modify - Actions - - - Currently not possible

Properties of the Parameter : Profile Description

Profile[35] Profile[36] Profile[37] Profile[38] Profile[39] Profile[40] Profile[41] Profile[42] Profile[43] Profile[44] Profile[45] Profile[46] Profile[47] Profile[48] Profile[49] Profile[50] Profile[51] Profile[52] Profile[53] Profile[54] Profile[55] Profile[56] Profile[57] Profile[58] Profile[59] Profile[60] Profile[61] Profile[62] Profile[63] Profile[64] Profile[65] Profile[66] Profile[67] Profile[68] Profile[69] Profile[70] Profile[71] Profile[72] Profile[73] Profile[74] Profile[75] Profile[76] Profile[77] Profile[78] Profile[79] Profile[80] Profile[81]

Figure 17. G35 Emulation Scenario Editor

In G35 tester, engineer can define multiple traffic profiles. Each traffic profile is located under its own directory which is labeled with an id that is shown in a square bracket of the directory name. For example, a directory with the name *Profile[38]* indicates that it is the network traffic profile with id 38. *Figure 18* shows the directories of network traffic profiles in G35 tester.

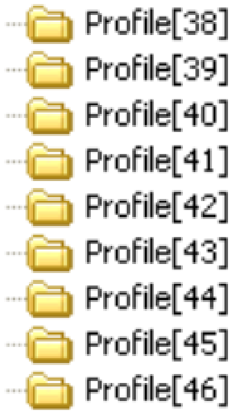


Figure 18. Directories of network traffic profile in G35 tester

In each profile, there is a bunch of parameters which can be edited. We can edit the traffic profile by setting proper values to parameters as shown in *Figure 17*.

There are three important parameters in the traffic profile, which are ”***Start scenario***” , ”***Call scenario***” and ”***Stop scenario***”. They form a sequence of communications in the automatic mode of network traffic procedure. We normally define the ”***Start scenario***” to do some preparation tasks for the network traffic generation that is going to happen in ”***Call scenario***”. In ”***Call scenario***” we define the real actions to generate the traffic, for example, attaching subscribers to the network. The ”***Stop scenario***” is usually used to define some final phase actions on the network, for example detaching subscribers from the network. A concrete view of automatic network traffic procedure in G35 tester is showing in *Figure 19.*

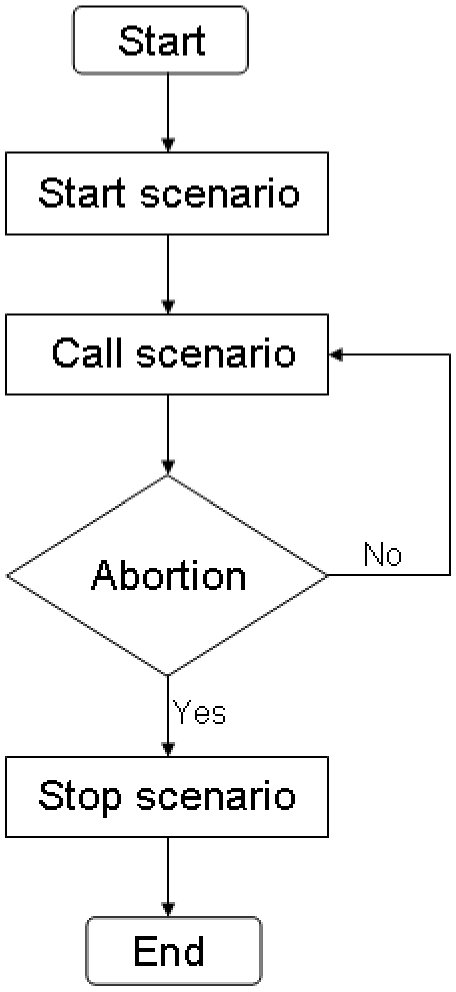
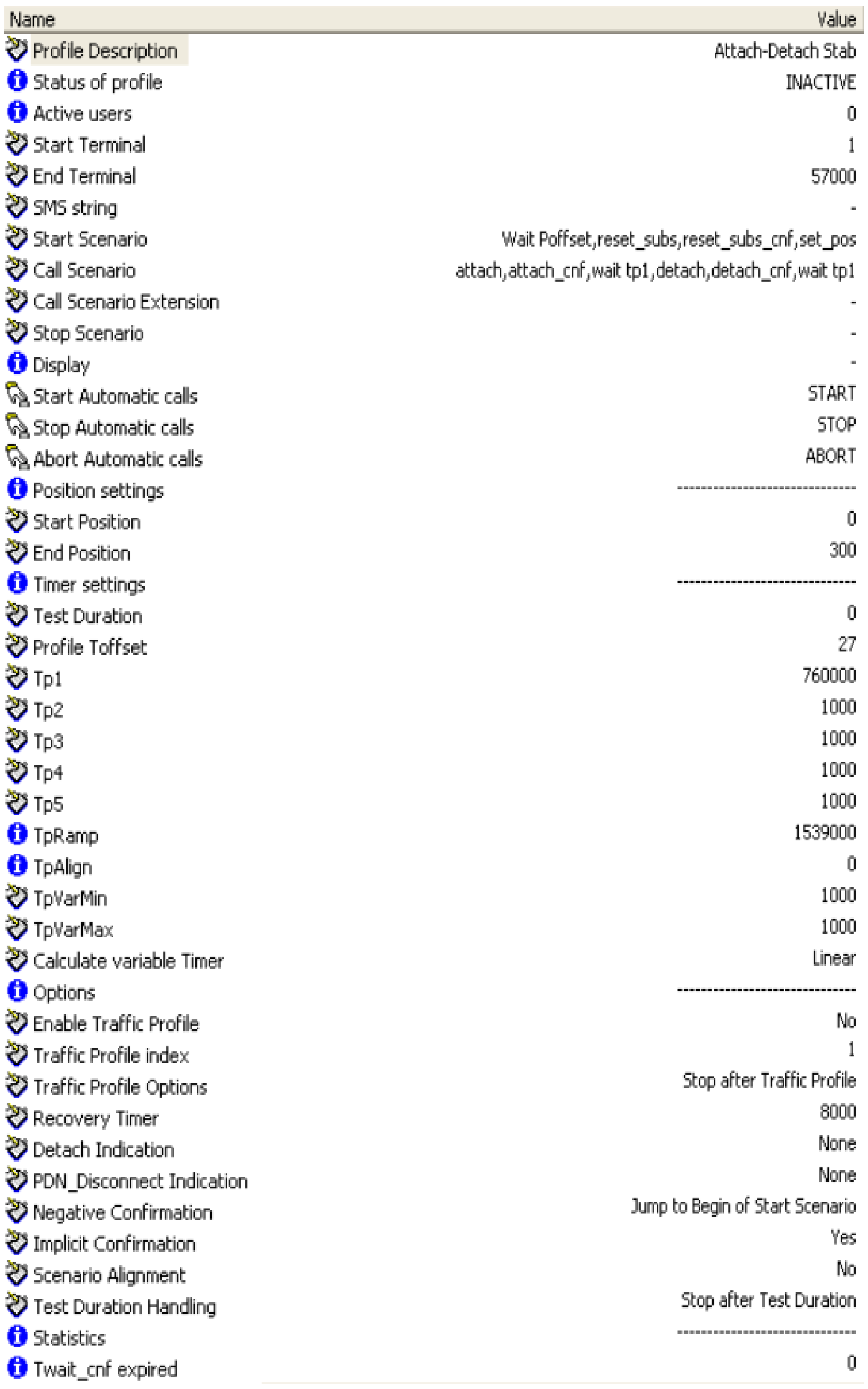


Figure 19. Sequence diagram of automatic traffic procedure

The above sequence diagram shows that the ”***Start scenario***” and ”***Stop scenario***” are supposed to be run only once during the whole procedure, while the ”***Call scenario***” can be run circularly. We can explicitly stop the ”***Call scenario***”.

In G35, the three scenario parameters and all the other parameters are editable in a emulator editor window. *Figure 20* shows what the possible parameters are in a G35 traffic profile in the editor window.



Name	Value
Profile Description	Attach-Detach Stab
Status of profile	INACTIVE
Active users	0
Start Terminal	1
End Terminal	57000
SMS string	-
Start Scenario	Wait Poffset,reset_subs,reset_subs_cnf,set_pos
Call Scenario	attach,attach_cnf,wait tp1,detach,detach_cnf,wait tp1
Call Scenario Extension	-
Stop Scenario	-
Display	-
Start Automatic calls	START
Stop Automatic calls	STOP
Abort Automatic calls	ABORT
Position settings	------------------------------
Start Position	0
End Position	300
Timer settings	------------------------------
Test Duration	0
Profile Toffset	27
Tp1	760000
Tp2	1000
Tp3	1000
Tp4	1000
Tp5	1000
TpRamp	1539000
TpAlign	0
TpVarMin	1000
TpVarMax	1000
Calculate variable Timer	Linear
Options	------------------------------
Enable Traffic Profile	No
Traffic Profile index	1
Traffic Profile Options	Stop after Traffic Profile
Recovery Timer	8000
Detach Indication	None
PDN_Disconnect Indication	None
Negative Confirmation	Jump to Begin of Start Scenario
Implicit Confirmation	Yes
Scenario Alignment	No
Test Duration Handling	Stop after Test Duration
Statistics	------------------------------
Twait_cnf expired	0

Figure 20. Traffic profile parameters

Since there are many parameters and sub-parameters in G35 traffic profile, I will use bold italic font to indicate the main parameters that is showing in *Figure 20*, and use italic font to indicate sub-parameters and its values. Those sub-parameters are defined inside each keyword which will be explained.

As we see in *Figure 20*, the three important parameters ”***Start scenario***”, ”***Call scenario***” and ”***Stop scenario***” each contains a value that consist of a string of action keywords. For example, ”*Wait Poffset, reset_subs, reset_subs_cnf, set_pos*”. These keywords define a sequence of actions on the network that yields a specific traffic sequence.

The value of ”***Start Scenario***” parameter is a string of action keywords which are supposed to be executed only once in the starting phase of the traffic profile. The traffic profile specific ”*Poffset*” usage is also an important action which guarantee a constant and stable message rate and CPU usage[33]. For example, in *Figure 20,* we have an action keyword ”*Wait Poffset*” in ”***Start Scenario***”. We also call the string of action keywords as traffic sequence.

The ”***Profile Toffset***” parameter in the traffic profile defines the network traffic speed implicitly. The ”*Wait Poffset*” action in parameter ”***Start scenario***” defines the waiting time interval between subscribers. In other words, ”*Wait Poffset*” means that before we start this traffic profile, we will wait the time that has been defined for the ”*Profile Toffset*” parameter. To be clearer, let’s take an example. We set ”*Profile Toffset*” parameter with value 10ms, then we will get the traffic speed by dividing one second by ”*Profile Toffset*”, which is 1000ms / 10ms = 100attach/sec . Thus, the traffic speed is 100 subscribers attaching to the network per second. If we have three similar profile running simultaneously, the traffic speed will be 300attach/sec. Well, in the automated regression testing system, the highest traffic speed that has been used in our team is 1000 attach/second.

Like what I have already mentioned above, in G35 tester, traffic sequence parameters are defined by using the keywords, those keywords actually come from so called LTE CALLGEN Scripts. Generally, each keyword represents a network action, for example, actions of subscribers’ attachment and detachment are described with keywords ”*ATTACH*” and ”*DETACH*” respectively. *Table 4* shows some keywords used for traffic sequence definition.

Table 4. Example keywords of LTE CALLGEN Scripts

Action	Keywords	Parameter 1	Parameter 2
Attach	**ATTACH**	Position(0..299)	
Detach	**DETACH**	Detach type (EPS, POWER_OFF)	
Set position	**SET_POS**	Position(0..299)	Cell Idx(0..15)
Reset subscriber	**RESET_SUBS**		
Start timer	**WAIT**	Timer id or time in ms	

As can be seen in *Table 4*, there are sub-parameters which can be defined for each keyword. These parameters have certain range of values defined.

Since G35 tester is used to emulate communication systems. For the emulation case of 4G LTE system, there are of course definitions of eNB. The maximum number of eNBs for the Tektronix G35 tester is 5000. Each eNB is identified by its id. The parameter ” *Position(0..299)*” in above table means that we will use eNBs with id from 0 to 299. So, eNBs with id 0 to 299 are reserved for this network traffic profile only. Other available eNBs can be used in other profiles, for example, if we have a second network traffic profile, we could use eNBs with id from 300 to 599. This eNBs configuration can be going on until eNB with id 4999, with which we have reached the maximum allowed amount of eNBs in G35 tester.

The ”*SET_POS*” keyword represents the action of ”Set position” for eNBs. It is used as the last action in ”**Start scenario**” in our case. It sets the traffic profile to start the scenario by using the first eNB (that’s the eNB with id 0) that is defined for this traffic profile. More detailed explanation is that, our traffic profile which has the eNBs with id from 0 to 299 will start the traffic by using eNB with id 0 firstly for the first subscriber, then, uses eNB with id 1 for the second subscriber, and so on with a round robin fashion.

As you may have noticed already, in traffic profile, we can define subscribers in network traffic in G35. For example, in ”***Call scenario***”, we can define a string of actions like ”*attach, attach_cnf, wait tp1*”. The last action of this string ”*wait tp1*” is actually a timer. In order to allow a large number of subscribers to be used in G35 tester without overloading the CPU, a timer must be placed. The timer can be used between any actions that are defined in action strings.

There are several timers in Tektronix G35, they are ”*Tp1,Ttp2,Ttp3,Ttp4,Tp5*”. Like *Figure 20* shows, we defined the value of timers as following in our case:

Table 5. Timers

Timer	**Value(ms)**
Tp1	760000
Tp2	1000
Tp3	1000
Tp4	1000
Tp5	1000

G35 tester user can use the defined timer to set delays for the traffic profile in any place that is needed.

Normally, we might want to attach all subscribers in the ”***Start scenario***” while doing routing area updates in ”***Call scenario***” for example, and detach them all in ”***Stop scenario***”. But in our particular traffic profile, we do not define subscribers attachment in ”***Start scenario***”, instead, we define it in ”***Call scenario***”. Besides, we do not have ”***Stop scenario***” action in the profile. The reason is that, we would like to periodically attach subscribers into the network instead of attaching them once only, and we would like to see the performance of MME when the traffic is attached and detached .

Our ”***Start scenario***” and ”***Call scenario***” action strings are defined as below in Table 6.

Table 6. Start scenario, call scenario and stop scenario of network traffic profile

Start scenario	Wait Poffset,reset_subs,reset_subs_cnf,set_pos
Call scenario	attach,attach_cnf,wait tp1,detach, detach_cnf,wait tp2
Stop scenario	-

In ”***Start scenario***” of *Table 6*, the ”*reset_subs*” parameter means to reset terminals. While, the ”*set_pos*” parameter means to set positions of eNBs which are going to be used and identified by id. The ”**_cnf*” parameters represent confirmation messages, since G35 tester will wait for the confirmation.

With all the explanations above, we can easily translate the action strings in *Table 6*. We can say that the ”***Start scenario***” in *Table 6* firstly wait for an amount of time which is defined by ”*Profile Toffset*” (that’s 27 milliseconds). After that, it resets the terminals, and waits for a confirmation message for it (that’s a reset successful message). Finally, the ”***Start scenario***” set the position of eNBs that will be used.

The action string of ”***Call scenario***” indicates that, the tester firstly attach subscribers with a speed of ”*1sec/Poffset*” attaches per second, then, it waits for the confirmation message of successful attachment. After all subscribers have been attached, the tester waits for 760 seconds (because here ”*Tp1*” is set to 760000 milliseconds), after which G35 tester then starts to detach the subscribers until it gets the detach successful confirmation message. Finally, the ”***Call scenario***” waits for 1 second, as ”*Tp2*” is set to 1000 milliseconds.

We have no action for ”***Stop scenario***” as indicated in *Table 6*.

Some other noticeable parameters in traffic profile are:

- ”***End terminal***” which represents the number of subscribers in the traffic. For example, in *Figure 20*, we set it with value 57000, that’s in total 57000 subscribers.
- ”***Start position***” which indicates the id of starting eNB when traffic starts. In *Figure 20*, we set it to 0. That’s start with eNB 0.
- ”***End position***” which indicates the id of ending eNB. In *Figure 20,* we set it to 300. As we use the id range of eNBs from 0 to 299, there are totally 300 eNBs, the ending eNB with id 300 is actually not in use during the traffic time.
- ”***Traffic profile options***”, this parameter defines what is going to do after all the scenarios (”***Start scenario***”, ”***Call scenario***”, ”***Stop scenario***” and so on) have gone through. In our case, we just stop everything after scenarios have gone through. You can find the value ”*Stop after Traffic Profile*” for this parameter in *Figure 20* .
- ”***Start Automatic calls***”, this is actually a button which triggers the traffic to start running.
- ”***Stop Automatic calls***”, this is also a button like element which can stop everything immediately during the running of network traffic.

With the network traffic profile tool in G35 tester, we can define a sequence of network actions with an amount of subscribers to achieve one or multiple network traffic scenario(s). G35 tester strongly facilitates us to generate network traffic for the purpose of testing network elements.

5.2 Enable Tektronix G35 remote control

The G35 tester provides a Distributed Programming Environment (DPE), which allows for a remote-controlled implementation and evaluation of measurements. The DPE is installed as part of Tektronix G35 Remote Operation Package (ROP) which consists of two parts: 1. A CORBA interface for self-created remote control programs. 2. A web-based querying possibility. We use the first solution to implement a remote control of G35 tester.

The G35 tester supports test automation by providing a CORBA based interface, which enables the tester to remotely control a customer specific external test management system. In this way, user does not need to develop test scenarios using the actual test hardware on Tektronix G35, instead, the test scenarios can be developed in a remote client and running remotely through networks.

CORBA is the acronym for Common Object Request Broker Architecture. The Object Management Group (OMG) defines CORBA [34]. OMG comprises almost all the major vendors and developers of distributed object technology. CORBA is a vendor independent architecture. Computer applications can use it to work together over networks. A CORBA based program from any vendor, on almost any computer, operating system, programming language and network can interoperate with a CORBA based program from the same or another vendor, on almost any other computer, operating system, programming language and network [34].

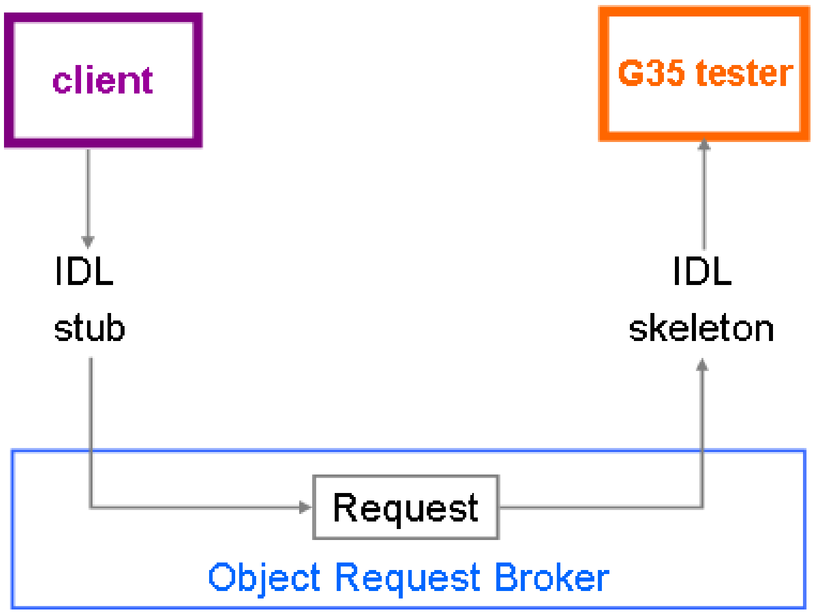


Figure 21. Remote control of G35 through CORBA

The basic CORBA paradigm is to request for services of a distributed object. There is one interface that is defined in OMG's Interface Definition Language (IDL), which we called IDL interface. The remote distributed object is identified by object reference which is typed by IDL interface. Client holds an object reference to the distributed object, and send request through the so called Object Request Broker

(ORB). The ORB delivers client request to the distributed object and returns the results which are generated by the services of the distributed object to the client. The ORB is a distributed service which implements the request to the distributed object. It also locates the remotely distributed object on the network, delivers client request to the object, waits for the results, and transfer the results back to the client [35].

With CORBA interface, the remote client can access Tektronix G35 through the advertised interface. Client invokes the operations that are exposed by G35 tester through its IDL interface.

The CORBA software comes with an IDL compiler, which compile the IDL into client stub and object (G35 tester) skeleton. That's producing IDL definitions for the usage with IDL stub on client and the usage with IDL skeleton on G35 tester. IDL compiler can translate the IDL files to Java, Python or C++. Stub and skeleton serve as proxies for client and G35 tester respectively. IDL defines interfaces very strictly that there is no problem even if the client stub and G35 tester skeleton are compiled into different programming languages, or running on different ORBs from different vendors.

On G35 tester, there are three IDL files which are *control.idl*, *protocoltest.idl* and *session idl*. In order to initialize the client side local ORB and establish the client stub successfully, we have to copy these three IDL files from G35 tester to client like *figure 22* shows.

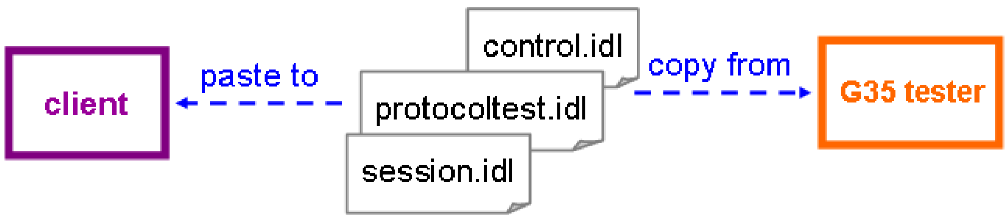


Figure 22. Three IDL files from G35 tester to client

After client got the three IDL files from G35 tester, it uses a tool called *omniidl* to compile the IDL files to python files (*control_idl.py*, *protocoltest_idl.py* and *session_idl.py*). The client program developed for remotely controlling the G35 tester uses Python as the programming language. The client python program imports the above mentioned three python files so that the client local ORB object can be initialized. The client local ORB is used to instantiate proxy objects which are defined in IDL interface. The services defined in proxy objects are invoked in client python program, the local ORB then generates and transmit the case specific request to the remote ORB on G35 tester.

5.3 Initial phase of test suite

The initial phase test case builds the working environment of Tektronix G35 tester, configures the SSH feature on ATCA server, set the test suite variables and organize the directories structure of tests results. These initial steps prepare all the things that are needed for main test cases. Basically, by running this initial test case, we want to make sure all these preparations are successful, if anything is failed during the test case running, the whole test suite will be stopped at that point.

There are several computer units in ATCA, we are interested in their CPU loads during test running. In order to track the CPU load of each computer unit in ATCA simultaneously, there is one python script which implements the multithreading feature. This Python script is located on a remote Linux server. In order to communicate with ATCA server, we established an SSH connection between the Linux server and ATCA. SSH is a protocol that provides secure network communications with authentication, encryption and data integrity capabilities. SSH offers a secure command-shell, secure file transfer and remote access to applications via a secure tunnel [36].

There are two SSH connections involved in order to get the CPU load of each computer unit in ATCA. The first SSH connection is between the Linux server and ATCA, through which, the python script on Linux server can send MML commands to ATCA server to track the CPU loads. The second SSH connection is between the client PC where test case is running and the Linux server, so that test case can trigger the python script remotely on Linux server. The ATCA server’s SSH feature must be activated, and by default, it is in the inactivate mode, so, the test case must activate the SSH feature in ATCA as an initialization step.

There are test suite variables defined in this phase. Variables like results directory paths, counter IDs are defined as test suite variables, which means they are visible by test cases inside the test suite.

5.4 Statistics collection phase of test suite

The essential part of the test suite is the counter statistics collection for both G35 tester and ATCA server.

We use Robot framework to develop the test suite. The first part of the test suit is the test cases of Tektronix G35 control, which includes loading of G35, configuration of network traffic (traffic profile), starting and stopping of traffic and collection of counter statistics.

Counter statistics collection is periodically conducted. Both the values of counters on ATCA and G35 tester are collected. The results are organized and write to files in order to show periodical statistics which then be drawn as time-value graphs automatically.

5.4.1 Tektronix G35 counters collection

Our test case remotely controls Tektronix G35 tester so that the developer does not need to work besides the G35 tester physically when creating the test case. It eases the work physically, because developer is able to focus on working with his/her workstation, while the G35 tester can be located besides the ATCA MME, so that the other tasks besides MME won't be bothered.

Section 5.2 has introduced how to enable remote control for G35 tester by utilizing the CORBA interface defined in Tektronix G35 Remote Operation Package (ROP). After everything has been setup correctly, a custom Python file named ”*TesterController.py*” is created on client like *Figure 23* shows. This custom Python file imports the python files compiled from IDL files from G35 tester. Developers can freely create functions in this file to control the G35 tester. This Python file is located under *Remote_G35_SE/* directory on client.

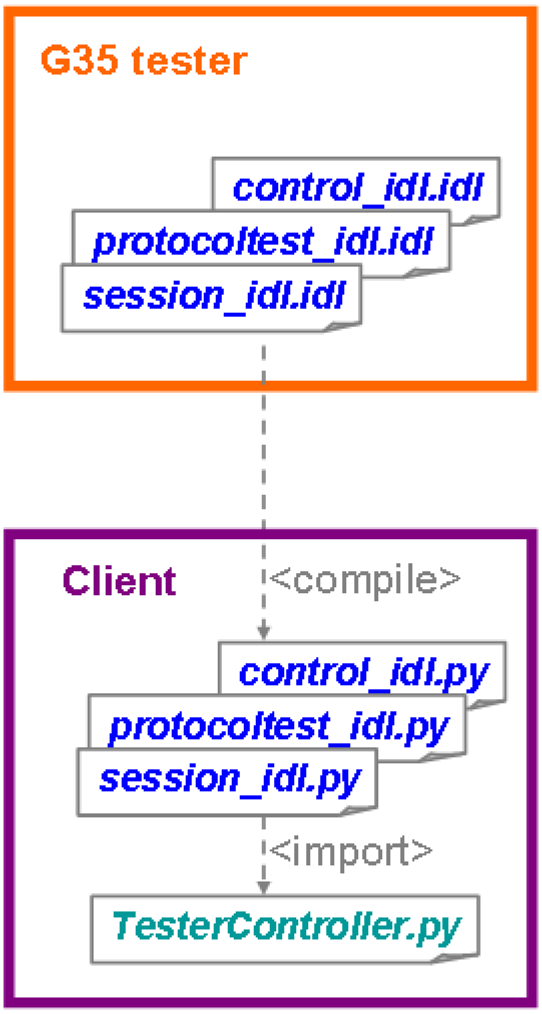


Figure 23. Python file for Tektronix G35 remote control

Our test case for G35 remote control utilizes the functions in *TesterController.py* to establish a remote connection to G35 tester and get the counters which we are interested in.

There is a customized keyword in our test case. This keyword invokes a function named *create_tektronix_counter_parameter_list* that is defined in Python file *TesterController.py*. The corresponding keyword in robot framework is then named "*Create Tektronix Counter Parameter List*" which invokes the Python function. This Python function creates a list that contains all the counter parameters defined in G35 tester. In order to get the counter list of G35 tester, the Python function needs to know where the counters' value could be found on G35 tester. In our case, the counter parameters' path on G35 tester is "*ENB_UE_NAS*

Statistics.Counter.Procedure". So, the Python function is able to get all counters' value by accessing this path.

5.4.2 ATCA CPPU & SMMU counters collection

The ATCA based MME contains the following operation units:

OMU ... OPERATION AND MAINTENANCE UNIT

CPPU .. CONTROL PLANE PROCESSING UNIT

MCHU .. MARKER AND CHARGING UNIT

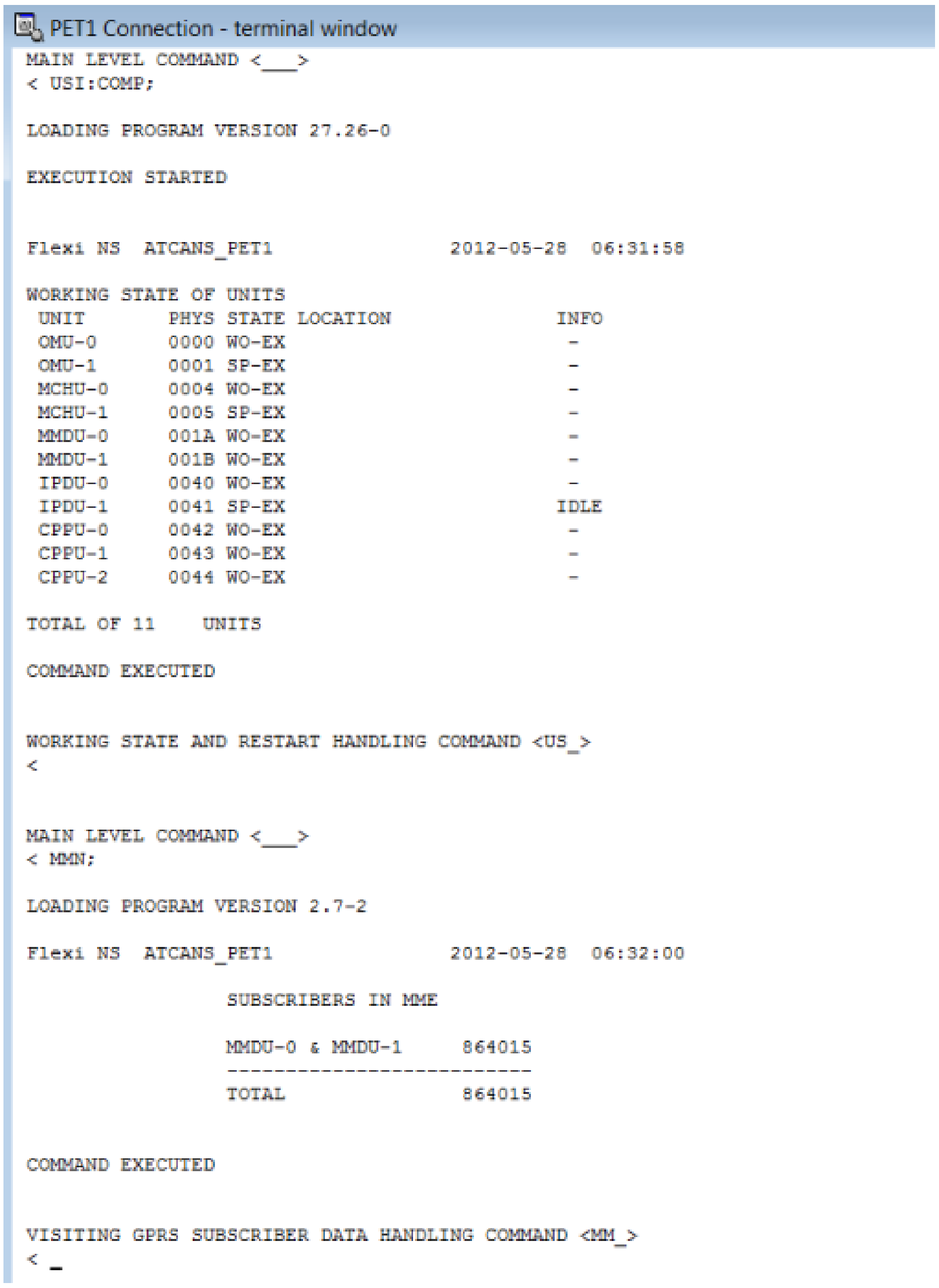
SMMU .. SIGNALING AND MOBILITY MANAGEMENT UNIT

The SMMU has been replaced by MMDU later on. MMDU stands for Mobility Management and Database Unit, which stores visiting subscriber information into the visiting subscriber database.

In this thesis work, we are mostly interested in CPPU and SMMU counters on ATCA MME.

We use Man Machine Language (MML) command [37] to communicate with ATCA MME. MML is an interface for managing a network device from a console. We have a terminal window through which the engineer can send MML command to access data and to operate units on MME. *Figure 24* shows the ATCA terminal window.

In order to get the MME counters, our test cases firstly establish a telnet connection to ATCA server, and then it starts a remote session to the operation unit of ATCA MME which we are interested in. By giving the MML command "*ZDDS*" on ATCA terminal, a remote session is started, the only thing left is to tell the terminal to which operation unit the remote session would like to connect. For example, we are interested in CPPU, the test case executes command " *ZDDS:CPPU*" on terminal to establish a remote session to CPPU unit. After the remote session has been established, the terminal is in CPPU's Linux operating system, we can execute Linux command on that unit.



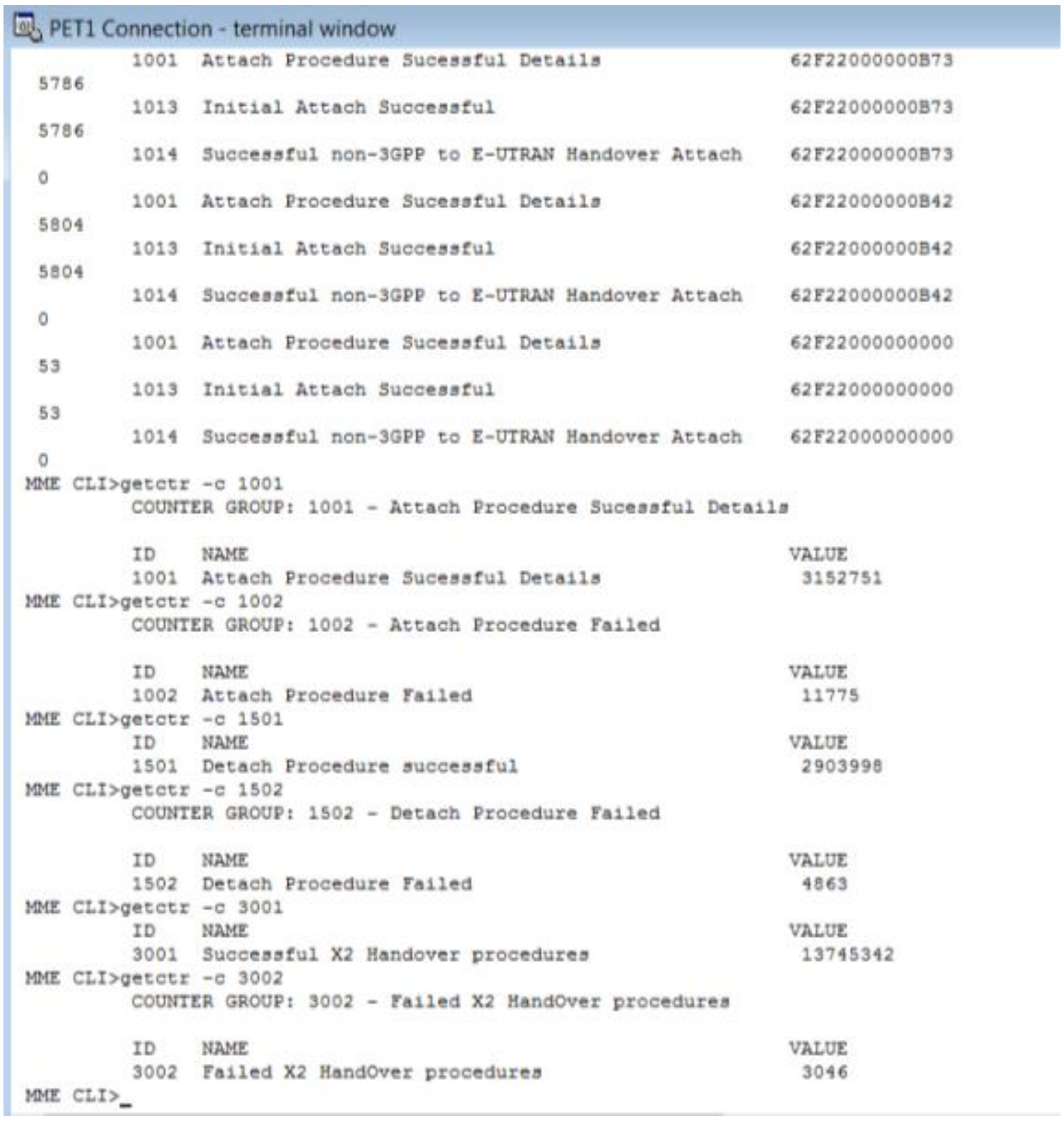
```
PET1 Connection - terminal window
MAIN LEVEL COMMAND <___>
< USI:COMP;

LOADING PROGRAM VERSION 27.26-0

EXECUTION STARTED


Flexi NS  ATCANS_PET1               2012-05-28  06:31:58

WORKING STATE OF UNITS
 UNIT       PHYS STATE LOCATION             INFO
 OMU-0      0000 WO-EX                       -
 OMU-1      0001 SP-EX                       -
 MCHU-0     0004 WO-EX                       -
 MCHU-1     0005 SP-EX                       -
 MMDU-0     001A WO-EX                       -
 MMDU-1     001B WO-EX                       -
 IPDU-0     0040 WO-EX                       -
 IPDU-1     0041 SP-EX                      IDLE
 CPPU-0     0042 WO-EX                       -
 CPPU-1     0043 WO-EX                       -
 CPPU-2     0044 WO-EX                       -

TOTAL OF 11    UNITS

COMMAND EXECUTED


WORKING STATE AND RESTART HANDLING COMMAND <US_>
<


MAIN LEVEL COMMAND <___>
< MMN;

LOADING PROGRAM VERSION 2.7-2

Flexi NS  ATCANS_PET1               2012-05-28  06:32:00

                 SUBSCRIBERS IN MME

                 MMDU-0 & MMDU-1     864015
                 --------------------------
                 TOTAL               864015


COMMAND EXECUTED


VISITING GPRS SUBSCRIBER DATA HANDLING COMMAND <MM_>
< _
```

Figure 24. ATCA terminal

There is one jar file located under path opt/java/ of CPPU’s operation system. We run the following commands to run this jar program:

java -jar mmeCLIHandler-1.0-SNAPSHOT.jar

After run the jar file, the terminal is switched to the level of **MME CLI**, in where we execute the command ”*getctr -r 7250-7283*” to get the CPPU counters. The “*7250-7283*” is the range of CPPU counter ID. *Figure 25* shows the results of CPPU counters we got from ATCA terminal. Our test case gets the counters' data showing on the terminal, a further manipulation on these data will be conducted later on.



PET1 Connection - terminal window

```
         1001  Attach Procedure Sucessful Details                62F22000000B73
 5786
         1013  Initial Attach Successful                         62F22000000B73
 5786
         1014  Successful non-3GPP to E-UTRAN Handover Attach    62F22000000B73
 0
         1001  Attach Procedure Sucessful Details                62F22000000B42
 5804
         1013  Initial Attach Successful                         62F22000000B42
 5804
         1014  Successful non-3GPP to E-UTRAN Handover Attach    62F22000000B42
 0
         1001  Attach Procedure Sucessful Details                62F22000000000
 53
         1013  Initial Attach Successful                         62F22000000000
 53
         1014  Successful non-3GPP to E-UTRAN Handover Attach    62F22000000000
 0
MME CLI>getctr -c 1001
         COUNTER GROUP: 1001 - Attach Procedure Sucessful Details

         ID    NAME                                              VALUE
         1001  Attach Procedure Sucessful Details                 3152751
MME CLI>getctr -c 1002
         COUNTER GROUP: 1002 - Attach Procedure Failed

         ID    NAME                                              VALUE
         1002  Attach Procedure Failed                            11775
MME CLI>getctr -c 1501
         ID    NAME                                              VALUE
         1501  Detach Procedure successful                        2903998
MME CLI>getctr -c 1502
         COUNTER GROUP: 1502 - Detach Procedure Failed

         ID    NAME                                              VALUE
         1502  Detach Procedure Failed                            4863
MME CLI>getctr -c 3001
         ID    NAME                                              VALUE
         3001  Successful X2 Handover procedures                  13745342
MME CLI>getctr -c 3002
         COUNTER GROUP: 3002 - Failed X2 HandOver procedures

         ID    NAME                                              VALUE
         3002  Failed X2 HandOver procedures                      3046
MME CLI>_
```

Figure 25. CPPU internal counters

Finally, the test case kills the remote session established with CPPU unit and close the telnet connection to ATCA MME.

Basically, we use the similar way to get SMMU counters. By giving the command "*ZDDS*", the terminal is in the level of service terminal, then, execute the commands "*'ZLP:1,MMT;', '1'*" and "*'L'*" to get the SMMU counters.

5.4.3 ATCA GPRS Measurement Handling

In ATCA MME, there are five types of GPRS measurement, which are listed below:

MMMT ..... MOBILITY MANAGEMENT TRANSACTION MEASUREMENT

SMMT ..... SESSION MANAGEMENT TRANSACTION MEASUREMENT

USLM ..... USER SMMU LEVEL MEASUREMENT

UMLM ..... USER MME LEVEL MEASUREMENT

SMLM ..... SECURITY MME LEVEL MEASUREMENT

On ATCA terminal, we execute the MML command "*ZTP*" to enter into GPRS measurement handling part. There are also counters for each measurement, our test case collect those counters too.

The measurements are time based. ATCA terminal provides measurement time configuration. To get the GPRS measurement counters, the first thing need to do is to set the measurement time characteristic for each measurement type. Thus, for each measurement type, we need to define the "*Measurement Day*", which is supposed to be picked from weekdays; the "*Measurement Interval*" for the day, which includes the measurement "*start time*" and the measurement "*stop time*"; also the "*Output Interval*" which defines how often to write the counter results to XML files.

For example, if we execute the MML command:

ZTPM: MMMT,ALL,00-00-24-00,24:;

it is for *MMMT* measurement, the measurement days are from Monday to Sunday, which is indicated by "*ALL*"; on each day, the measurement start time is 00:00, the measurement stop time is 24:00, that's "*00-00-24-00*", which means the measurement lasts 24 hours on each day; the output interval is 24 hours, indicated by the last figure "*24*", which means we only get the measurement output at the end of the day. So, for each day, we only have one output which will be written to XML files.

On ATCA server, the counter data of GPRS measurements are both written on XML files and buffered on the memory. As explained above, we have set an "*Output Interval*" (e.g. "*2*") for the measurement, which constrained that the measurement data is only able to be written on XML files after the output interval has reached (e.g.

24 hours), which means during the output interval, there is no record written on XML files. To get rid of this limitation, during the measurement, our test case is designed to get the measurement results directly from the memory, which gives the instantaneous result at any particular time without any constraint. Though, it sounds like the "*Output Interval*" setting is a useless action, it is an obligatory parameter to be set anyhow.

By executing the MML command '*ZTPP*' on ATCA terminal, the terminal is led to the level of '*DISPLAY MME MEASUREMENT COUNTERS*' which shows all the counters result from memory. The complete MML command to get courters result from memory is

ZTPP:[measureType],MEM:;

where "*[measureType]*" is supposed to be the name of measurement type. For example, to get the counters data of MMMT measurement, the complete MML command is *ZTPP:MMMT,MEM:;* . Our test case goes through all the measurement types to get the counters data of all five measurements.

After all the GPRS measurements are completed, the measurements need to be stopped. We execute the MML command *ZTPE: [measureType],;* to stop the measurements, where "*[measureType]*" should be replaced by the actual name of the measurement type as well. If the measurement is not stopped properly, there will be an error message of '*GPRS is unlocked and busy*' when you are trying to run the MML command to display the measurement counters again.

5.4.4 TOPTEN tracking

The term "TOPTEN tracking" means to track the CPU loads of all units on ATCA. The reason why we check the CPU loads is that we need to make sure the CPU load of each unit is reasonable for the automation test. We need to check that the load does not increase more than it should be. For example, there could be some leakage in CPU load which would cause the CPU load to increase indefinitely until the unit runs out of free CPU capacity. Besides, we would like to make sure that the load is within acceptable levels. For another example, the CPU load is supposed to not exceed more than 70% of its total capacity in order to make sure that the system is still functional well. So, if the CPU load becomes near to 100%, then we know that we probably will run into problems, since the unit is already overloaded.

The test case functions of TOPTEN tracking are mainly implemented in a Python script which is located on a remote Linux server. Robot framework test case invokes the functions as keyword. The units to be tracked on ATCA server are OMU-0, OMU-1, MCHU-0, MCHU-1, SMMU-0 and CPPU. All the CPU loads of those units are tracked simultaneously by implementing the multithreading feature in python. There are two SSH connections established, one of them is between the client computer in where the test case is running and the remote Linux server, the other one is between the Linux server and the ATCA server.

So, the general process is that, the test case firstly establish a SSH connection to the Linux server, then, it remotely executes a Linux command through SSH connection to run the python script (i.e. *unitsLoadsTrack.py*) on the Linux server. After that, the running python script on the Linux server establishes a SSH connection to ATCA server and sends MML commands through SSH connection to ATCA to get the units CPU loads. Finally, the python script gets the ATCA returned units CPU loads, and returns the results to the test case. The process is illustrated in *Figure 26*.

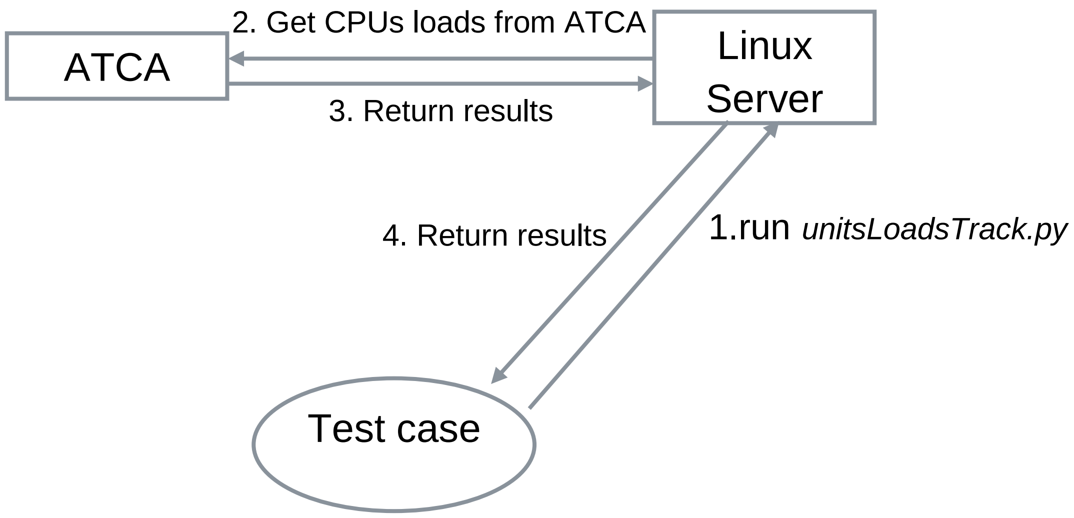


Figure 26. The process of TOPTEN unit load tracking

5.5 Statistics recording phase of test suite & Graph generation

After the test cases get all the groups of counters data, they write the data into files. The statistics recording phase includes three steps, which are data extraction, item formatting and value recording.

Basically, the counter data got from ATCA are in plain text, from which the test case needs to extract the data and information which we are interested in. The test case firstly converts the plain text into a list of string, where each item of the list is generated from one row of the plain text, that's one record. The test case then iterates through the list and gets each counter name and counter value which we are interested in from each row of the plain text.

For each row of the plain text, after the counter name and counter value have been extracted, the test case then organizes those items into the format of "*Start Time Period, Counter Number, Counter Name, Counter Value*" which is considered as one record item. After that, each record item will be written to a **.dat* file. The following table shows the sample records in a **.dat* file:

Table 7. Sample records of counter's data

#PERIOD_START_TIME	**#COUNTER_NUMBER(s)**	**#COUNTER_NAME(s)**	**#COUNTER_VALUE(s)**
25-5-2010_11:9:37	20059	Tau---Paging-COLLISION	0000000000
25-5-2010_11:10:6	20059	Tau---Paging-COLLISION	0000000000
25-5-2010_11:10:31	20059	Tau---Paging-COLLISION	0000000000
25-5-2010_11:10:55	20059	Tau---Paging-COLLISION	0000000000

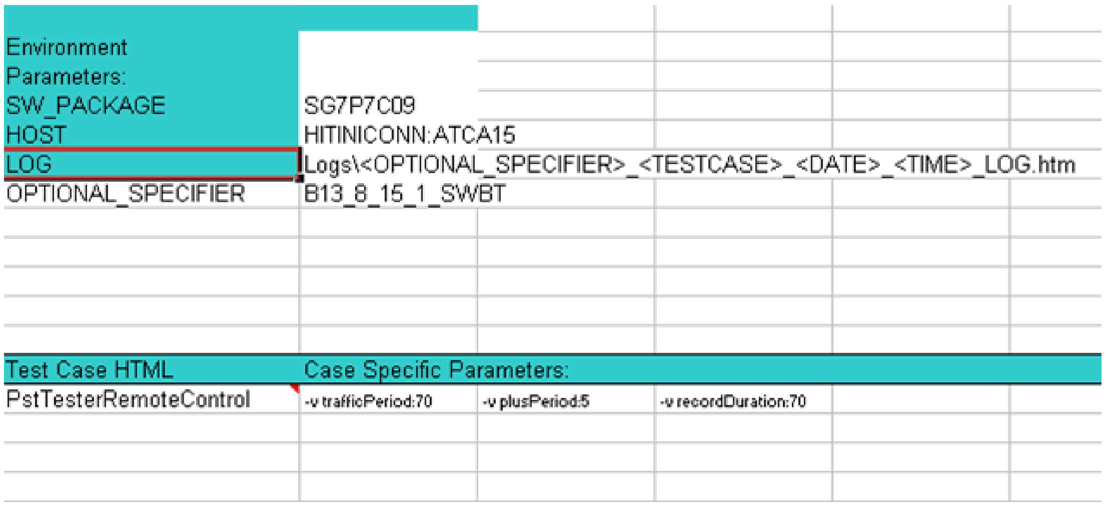
As the above table shows, our test case records the timestamp, counter ID, counter name and counter value into a DAT file. After the test case got these records, it is ready to generate statistic graphs of each counter.

We use a command-line driven graphing utility called Gnuplot [38] as the tool to generate statistic graph from DAT file. With the generated DAT file, our test case continues generating **_setting.gnu* files which is used by Gnuplot tool to generate graphs. Generally, the **_setting.gnu* files are configurations which define the data used for graphs, and how the data should be shown on the generated PNG graphs. The test case executes command "*gnuplot*" to generate PNG graphs.

5.6 Test case design in time domain

One requirement for this test automation is that a test engineer is able to start the test cases before he/she leaves the office, and can get the test results on the next working day. In other words, the test cases are supposed to be running for more than 10 hours. Of course the actual running time depends on when the test engineer starts and stops the tests. During the tests run, most of the time is actually consumed by periodical counters data collection from G35 and ATCA.

There is a user interface which allows test engineer to define explicitly the time domain parameters for the test suite, which is shown in figure 27.



Environment Parameters:				
SW_PACKAGE	SG7P7C09			
HOST	HITINICONN:ATCA15			
LOG	Logs\<OPTIONAL_SPECIFIER>_<TESTCASE>_<DATE>_<TIME>_LOG.htm			
OPTIONAL_SPECIFIER	B13_8_15_1_SWBT			
Test Case HTML	Case Specific Parameters:			
PstTesterRemoteControl	-v trafficPeriod:70	-v plusPeriod:5	-v recordDuration:70	

Figure 27. Test case time domain parameters

As you see in *figure 27*, there are three important time domain parameters, which are *trafficPeriod*, *plusPeriod*, and *recordDuration*. These parameters actually implicitly define the running period of our tests, as explained below.

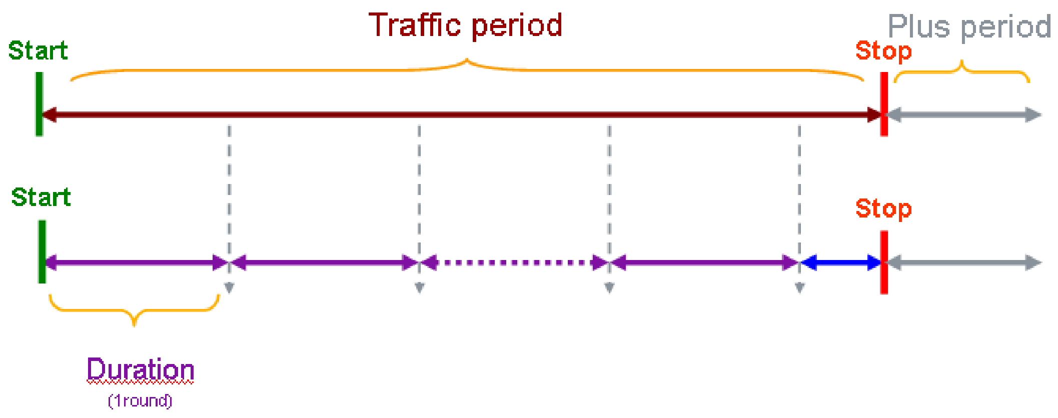


Figure 28. Traffic Period, Plus Period and Duration

Figure 28 above shows the definitions of the three parameters in time domain. The green color "*Start*" and red color "*Stop*" in *figure 28* indicate the traffic starting and stopping time. The parameter "*trafficPeriod*" in *figure 27* refers to the term *Traffic Period* of our tests in *Figure 28*. It indicates the total period spent on periodical counters collection after the network traffic started. *Plus Period* ("*plusPeriod*" in *Figure 27*) means the period spent on counters collection after we stopped the network traffic. The "*Duration*" in *figure 28* is identical to parameter "*recordDuration*" in *figure 27* which tells the time amount used for each round of counters collection.

The complete general view of our tests in time domain is showing in *Figure 29*.

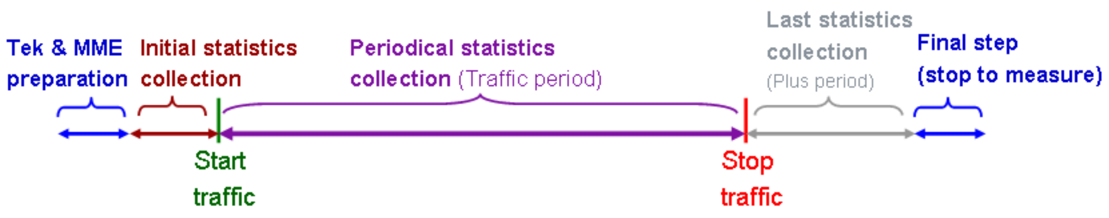

Tek & MME preparation
Initial statistics collection
Periodical statistics collection (Traffic period)
Last statistics collection (Plus period)
Final step (stop to measure)
Start traffic
Stop traffic

Figure 29. Tests in time domain

Firstly, the test case makes some preparations for Tektronix G35 tester and ATCA server. This mainly includes preparations for G35 counters and timer definitions for counters collection from MME (ATCA) memory (see section 5.4.3).

After that, before the traffic is started, there is an initial counters collection step. Generally, this step gets the data of the initial status of counters on G35 and ATCA.

Then, the test case starts the traffic from Tektronix G35 remotely. At this point, the real traffic started, counters start to record new values.

After the traffic started to flow, the test cases start the main task, that's the periodical counters data collection. By utilizing the user (test engineer) inputted parameter '*trafficPeriod*' and 'recordD*uration*' (this 'recordD*uration*' indicates how often to collect counters), the test suite is able to calculate how many rounds of collections there will be. If user defined traffic period is not exactly equal to an integer multiplied by user defined collection duration, the number of collection rounds will be extended properly. After the traffic stopped, the test cases wait for a '*plus period*' which is also defined manually, and then starts the last counters collection. The final step is to stop the counters measurement on ATCA.

Chapter 6 Results & Conclusions

The test suite is divided into several test cases. We consider each test case as one phase of this automation test. There are also time domain measurements for each test case. After the completion of test suite run, a bunch of results are generated. In this chapter, a summary of the average time spend on each test case during a test run is concluded; the results are explained. Besides, a further improvement is also proposed in this chapter.

6.1 Test phases & time domain overview

In general, there are seven test cases in our test suit, which implies seven phases for a complete test run. They are the "*Tektronix G35 and ATCA MME preparation*" phase, the "*Initial statistics collection*" phase, the "*Traffic starting*" phase, the "*Periodical statistics collection of traffic period*" phase, the "*Traffic stopping*" phase, the "*Waiting & statistics collection of plus period*" phase and the "*Final step*" phase. *Figure 30* shows the sequences of the seven phases.

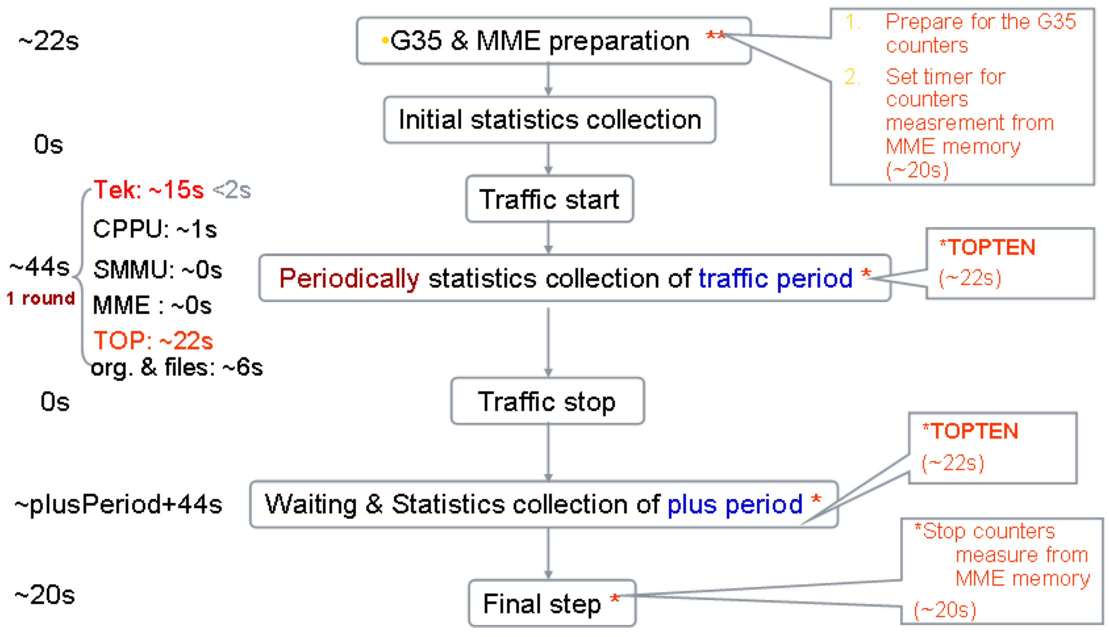

~22s
0s
~44s
1 round
0s
~plusPeriod+44s
~20s
Tek: ~15s <2s
CPPU: ~1s
SMMU: ~0s
MME : ~0s
TOP: ~22s
org. & files: ~6s
G35 & MME preparation **
Initial statistics collection
Traffic start
Periodically statistics collection of traffic period *
Traffic stop
Waiting & Statistics collection of plus period *
Final step *
1. Prepare for the G35 counters
2. Set timer for counters measrement from MME memory (~20s)
*TOPTEN (~22s)
*TOPTEN (~22s)
*Stop counters measure from MME memory (~20s)

Figure 30. Seven phases (test cases) of our test suite

There is a conclusion on the average time spent for each phase during the test run. As above figure shows, the first phase "*Tektronix G35 and ATCA MME preparation*" takes about 22 seconds in average, where most of the time is spent on setting timers for counters measurement from MME memory.

The step to prepare for G35 counters is faster, it takes about 2 seconds. At this point, the preparation for counters' collection is done; however, there is no network traffic yet, so the initial statistics collection is very fast. Our test case directly catches the values of counters in this phase, which is followed by a traffic starting action.

The network traffic is started in the third phase, after which there is traffic flowing on the core network. The most time consuming phase is the "*Periodical statistics collection of traffic period*". The total time spend on this phase is depending on the user's definition of "*Traffic period*" and "*Plus period*", which implies the number of rounds the test would take for the periodical statistics collection. On average, one round collection for all counters takes about 44 seconds, which includes around 14 seconds spending on G35 counters collection, 1 second spending on CPPU counters collection, 22 seconds for TOPTEN data collection and 6 seconds for organizing the counters' data and generating the statistic files. The steps for collecting SMMU counters and MME measurements are very fast, the time spends on them is less than 1 second.

After the periodical statistics collection, our test case stops the traffic, and enters to the "*Waiting & statistics collection of plus period*" phase. Here, we are interested in the changes of counter values after we stopped the traffic. So, the test case waits for a "*Plus period*" defined by user. Then, it triggers a last time collection for counters. The last collection takes about 44 seconds.

The final phase of our test suit is named "*Final step*", which stops the measurement of counters from MME memory. This part takes 20 seconds in average.

Overall, the most time consuming steps among all the phases are the setting of timers for counters measurement from MME memory, TOPTEN tracking and the stopping of counters measurement from MME memory.

6.2 Results directory structure

All the test results are automatically stored under a directory named "*Statistics*" on the workstation. Since the test suite collects counter statistics in several rounds during the execution, we use timestamp to indicate each round of the counters collection. The timestamp indicates each collection's start date and time, in the format of "*date@time*". For example, one result directory under "*Statistics*" folder is named "*2010-06-24@15_47_34*" which indicates that the results under this folder are generated from the collection that is started at time 15:47:34 on the date 24.06.2010. The number of timestamp folders is equal to the number of collections that are conducted during the test run.

Under each "*date@time*" directory, there are sub-directories for different collection groups of counters. The sub-directories of result folder are named by counter group names, for example: '*MME \ MMMT*', '*SMMU*'. Inside each counter group folder, there are DAT ("**.dat*") files which contain the records of counters' name and value. Besides, there are GNU ("**_Settings.gnu*") files which include configurations for graph generation. The most important files are those PNG "**.png*") files which are the counters' statistic graphs that are generated from DAT files. *Figure 31* shows the directory structure of test results.

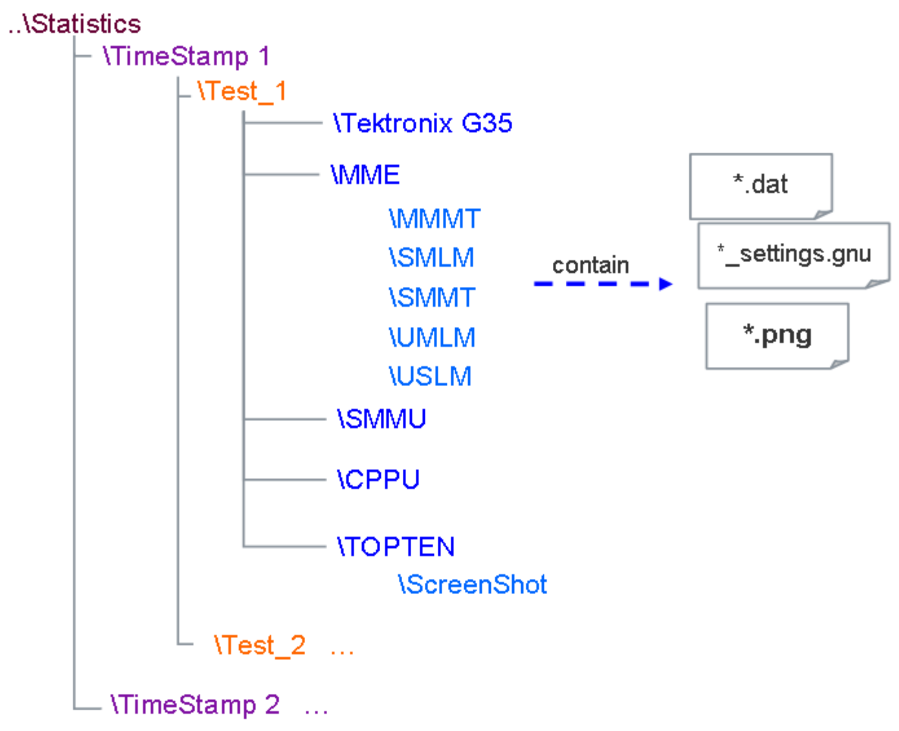


Figure 31. Directory structure of test results

6.3 Result statistics

Generally, there are three technical frameworks in the whole test system. There is a counter-framework in Tektrnoix G35, there is an ATCA MME counter-framework, and there is our test framework. Our test framework collects available counters data from both Tektronix G35 and ATCA MME counter-frameworks.

Tektrnoix G35 generates the network traffic and updates its counters in every second. The ATCA MME counters on the other hand are updated periodically. In ATCA MME, there is a waiting period which defines how often MME updates its counters. This period can be configured before starting the measurement. For example, the period in MME could be set to thirty minutes, or one hour, or even one day. MME counters are updated in every period.

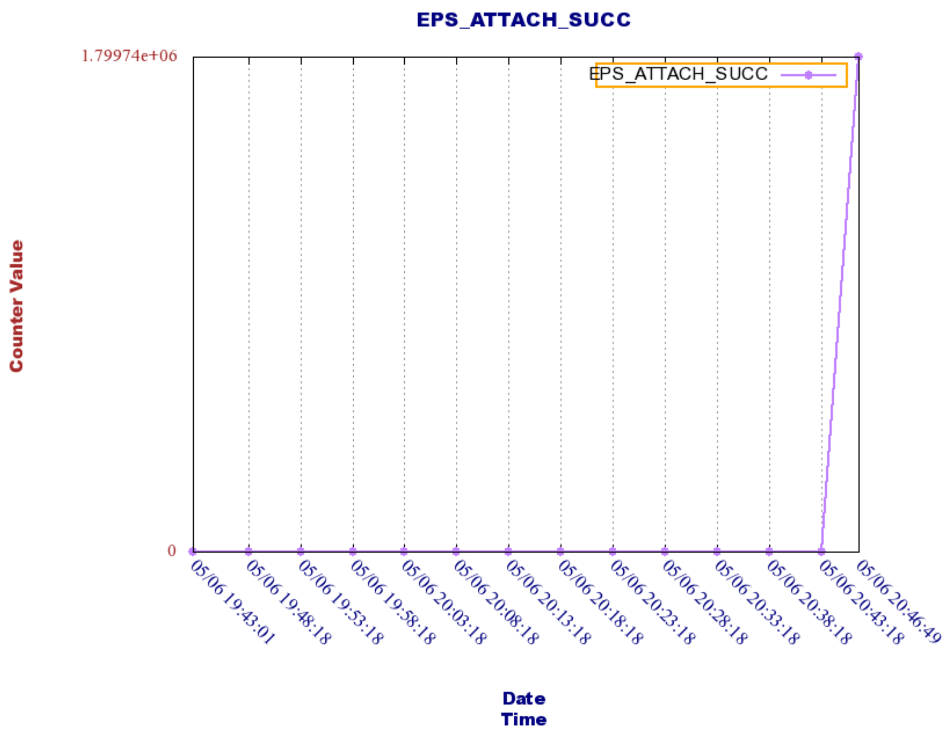


Figure 32. Statistics of ATCA MME counters

Let’s take an example of our test scenario. We set the period of MME to one hour, which means MME updates its counters once in every hour. We define the duration of test run to a bit more than one hour, that’s to define the ”*Traffic period*” of our test to one hour, and define the ”*Plus period*” to about three minutes. So, if our test framework collects data in every five minutes, the counters value will be zero in the first hour. Once one hour is reached, MME updates its counters, after which our test framework collects data again. There will be at least one sample which is valid practically. *Figure 32* and *figure 33* show the generated statistics graph of our example scenario.

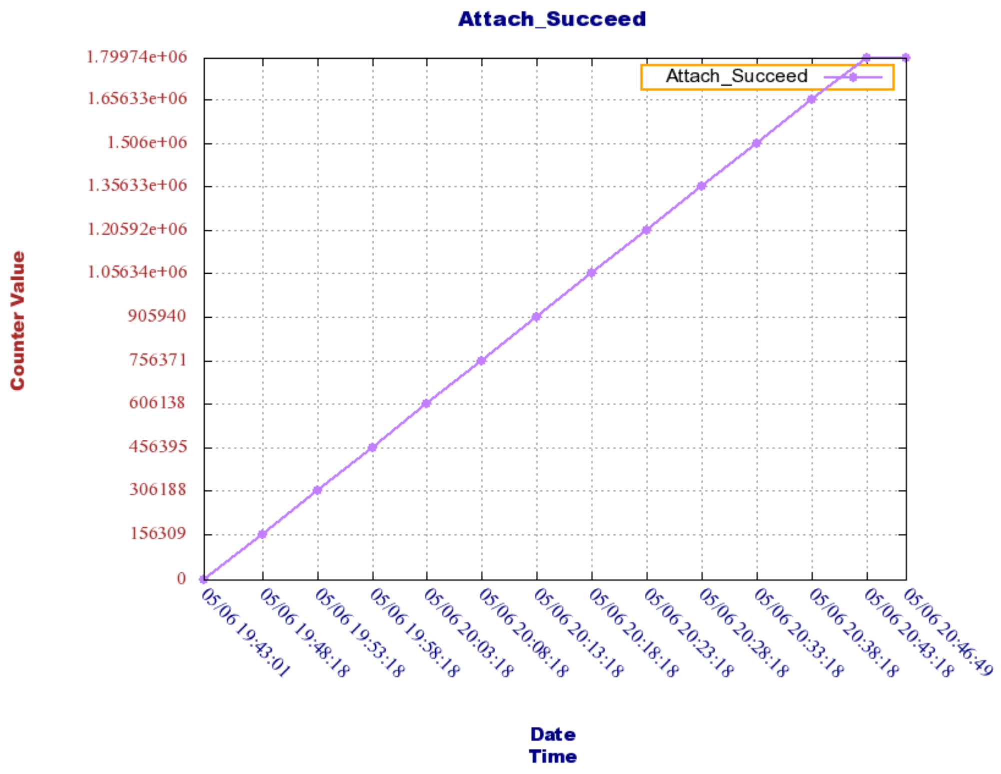


Figure 33. Statistics of Tektronix G35 counters

Figure 32 shows the result statistics of ATCA MME counters. Since MME updates its counters in every one hour, and our test framework collects data in every five minutes, so the counters are zero in the first hour. After then, MME updates its counters, and our test framework got one sample during the “*Plus period*”. The counters value in the sample is 1.79974*10^6.

Figure 33 shows the statistics of Tektronix G35 counters of our example. Since Tektronix G35 updates its counter in every second, so, we got a steady increase of counter values, but except the last collection period. Because that is the "*Plus period*", the network traffic has been stopped at the beginning of this period.

Let's also take a look at the statistics of MME TOPTEN tracking which are the MME CPU loads in our example. *Figure 34* shows the CPU loads of CPPU.

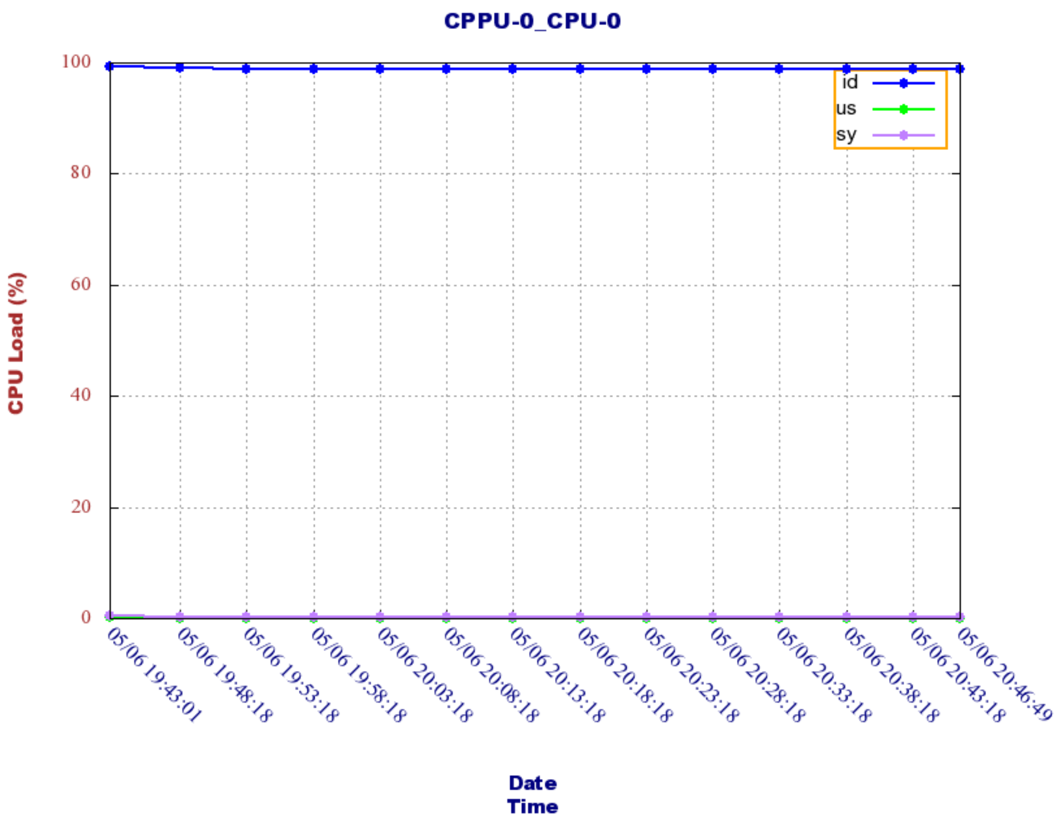


Figure 34. CPU loads of CPPU

The purpose to have also a statistics for the CPU loads is that we need to check that the CPU loads did not increase more than it should be. In *figure 34*, there are three colored lines. There is one blue line which indicates the idle usage, thus, how much is not in use. There is also one green line which indicates the user CPU usage. The purple line indicates the system CPU usage.

The relationship among the three usage lines is that when user usage and system usage are increasing, the idle usage will be decreasing. That's the total CPU load is increasing.

Generally, *Figure 34* shows that our test did not load the unit very much, since the user usage and system usage are only couple of percent, which is very low.

6.4 Further Improvements

During this thesis work, the Agile method [39] has been applied to the development process, because we want to make efficient development and set the goals clearly on weekly basis. During the test suite development, there are of course bugs coming out. Developers have to spend time to look for the bugs and fix them. This is understandable but it could be the bottleneck for the development process.

In the world of software development, produce reliable builds is an important part. One of the best practices in software development is to apply a fully automated build and test process that can notify the development team multiple times a day. Continuous Integration (CI) [40] is the term used for this practice, and it is actually a core Agile method practice. The fundamental benefit of using CI is that it removes the time spend on bug hunting during the development work. Generally, there are test suites written for the software during development, with CI, we are able to ensuring that the project compiles and pass its test suite.

Jenkins is an open source Continuous Integration tool written in Java [41]. It is very easy to use and can be used for projects in a wide variety of languages and technologies. As our test suite is developed based on Robot framework, there is also Robot framework plugin of Jenkins [42] available. This plugin currently provides features like showing test suite and case details with trend graphs, generate a summary of latest Robot run on build page and so on.

References

[1] Abdullah, M.F.L., Yonis A.Z. *Performance of LTE release 8 and release10 in wireless communications*. Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on 26-28 June 2012

[2] Sawahashi, M. *Broadband radio access: LTE and LTE-advanced*. 2009 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS 2009) December 7-9, 2009

[3] Kreuer, D. *Applying test automation to type acceptance testing of telecom networks: a case study with customer participation*. Ericsson Eurolab Deutschland GmbH, 52134 Herzogenrath, GermanyOct 1999

[4] *Diversified Technology, AdvancedTCA*. e-document, from: http://www.dtims.com/products/atca/whatisatca.php

[5] Liu, R. *An investigation on LTE Mobility Management*, IEEE TRANSACTIONS ON MOBILE COMPUTING, National Chiao Tung University, Hsinchu, Taiwan. 2011.

[6] Magnus Olsson, *SAE and the Evolved Packet Core: Driving the Mobile Broadband Revolution*, Academic press 2009, ASIN B0084F0RTM , Chapter 10, 227-237, chapter 6, 109-116

[7] Dahlman E., Parkvall S., Sköld J., Beming P. *3G Evolution : HSPA and LTE for Mobile Broadband*, Academic Press, 2007, ISBN 9780123725332 , Chapter 2 P17-21, Chapter 13 P314-316

[8] Nokia Siemens Networks: *open core system technical white paper*, e-document, from https://www.online.nokia.com/ [Retrieved 03.022012]

[9] Tektronix: WiMAX testing on the K1297-G35 platform, e-document, from

http://www.tek.com/datasheet/wimax-testing-k1297-g35-platform [Retrieved on 10.022012]

[10] 3GPP LTE overview, 3rd Generation Partnership Project webpage, from http://www.3gpp.org/Technologies/Keywords-Acronyms/LTE [Viewed on 10.02.2012]

[11] *General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network(E-UTRAN) access*, 3GPP TS 23.401 v10.8.0 (2012-6)

[12] freescale semiconductor: *Long Term Evolution Protocol Overview*, e-document, from http://www.freescale.com/files/wireless_comm/doc/white_paper/LTEPTCLOVWWP.pdf

[13] Arunabha G., Jun Z., Jeffrey G. A., Rias M..*Fundamental of LTE*. Pearson Education, Inc. 2010 , ISBN 0137033117, chapter 6, chapter 9, P348-350 chapter 10, P371-377.

[14] Xueli,An; Pianese, F.; Widjaja, I.; Acer, U.G.. *DMME: Virtualizing LTE Mobility Management*, Local Computer Networks(LCN), IEEE 36th conference on 4-7 Oct. 2011

[15] Vukovic, I; Filipovich, I. *Throughput analysis of TDD LTE Random Access Channel*, Personal Indoor and Mobile Radio Communications (PIMRC), 2011 IEEE 22nd International Symposium on 11-14 Sept. 2011

[16] radisys white paper: *Protocol Signaling Procedures in LTE*, e-document, from http://go.radisys.com/rs/radisys/images/paper-lte-protocol-signaling.pdf

[17] Artiza Networks: *LTE resources, RRC Connection Establishment/Release Procedure*, e-document, from http://www.artizanetworks.com/lte_tut_cpl_pro.html

[18] Nokia Siemens Networks website: http://www.nokiasiemensnetworks.com/ [viewed on 13.03.2012]

[19] Rajan, D. *Building IP networks using Advanced Telecom Computing Architecture*. Nortel, Research Triangle Park, North Carolina, USA. 19-25 June 2007

[20] *PICMG: AdvancedTCA*, website: http://www.picmg.org/v2internal/resourcepage2.cfm?id=2 [viewed on 15.03.2012]

[21] *PICMG 3 (AdvancedTCA) questions & answers*, e-document, from http://www.picmg.org/pdf/AdvancedTCAQA.pdf

[22] *Nokia Siemens Networks Flex Network Server*, NSN customer document: https://www.online.nokia.com [Viewed on 14.12.2011]

[23] Korpela J., *Tab Separated Values (TSV): a format for tabular data exchange*, e-document, from: http://www.cs.tut.fi/~jkorpela/TSV.html, [viewed on 20.03.2012]

[24] *Robot Framework User Guide version 2.1.3*, e-document, from http://robotframework.googlecode.com/svn/tags/robotframework-2.1.3/doc/userguide/RobotFrameworkUserGuide.html [viewed on 20.03.2012]

[25] *Manifesto for Agile Software Development*, e-document, from: http://agilemanifesto.org/, [retrieved on 22.03.3012]

[26] Lisa Crispin, Janet Gregory. *Agile Testing: A Practical Guide for Testers and Agile Teams*, Addison-Wesley Professional, 2008, ISBN 978-0-321-53446-0

[27] Minna Räisänen, *Case of three Scrum teams: Agile software development methods at Nokia - The people perspective*, Master's thesis, 2012, Aalto University School of Economics

[28] Elfriede Dustin, Jeff Rashka, John Paul, *Automated software testing* , Addison-Wesley, 2008, ISBN 0-201-43287-0, P3-19

[29] Collins, E.F., de Lucena, V.F., *Software Test Automation practices in agile development environment: An industry experience report*, Automation of Software Test (AST), 2012 7th International Workshop on 2-3 June 2012.

[30] Wissink, T., Amaro C., *Successful Test Automation for Software Maintenance*, Software Maintenance , 2006. ICSM '06. 22nd IEEE International Conference on 24-27 Sept. 2006

[31] Patton Ron, *Software Testing, 2nd edition*, Sams, 2005, ISBN 978-0672327988, page 231-240

[32] Douglas Hoffman, *Test Automation Architectures: Planning for Test Automation*, Software Quality Methods, LLC. 24646 Heather Heights Place Saratoga, California 95070-9710

[33] *UMTS SGSN Capacity Testing Including 2G/3G Handovers Using the K1297-G20*, e-document from: http://direct.www.tek.com/Measurement/App_Notes/2F_17575/eng/2FW_17575_0.pdf [Retrieved on 03.05.2012]

[34] *CORBA basics*, e-document from: http://www.omg.org/gettingstarted/corbafaq.htm [Viewed on 10.05.2012]

[35] Alex Chaffee, Bruce Martin, *Introduction to CORBA*, e-document from: http://java.sun.com/developer/onlineTraining/Downloads/CorbaIntro.PDF [Retrieved on 10.05.2012]

[36] SSH Overview, e-document from: http://www.vandyke.com/solutions/ssh_overview/ssh_overview.pdf [Retrieved on 10.05.2012]

[37] MML command overview, e-document from: http://www.cisco.com/en/US/docs/voice_ip_comm/pgw/9/command/reference/mmlovw.pdf [Retrieved on 20.05.2012]

[38] gunplot, gnuplot website: http://www.gnuplot.info/ [Viewed 20.05.2012]

[39] Cockburn, A.: *Agile Software Development*. Addison-Wesley, 2001.

[40] Paul M. Duvall, Steve Matyas, Andrew Glover. *Continous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007. ISBN-10 0321336380

[41] John Ferguson Smart. *Jenkins: The Definitive Guide, Continuous integration for the masses*. O'Reilly Media, 2011. ISBN-10 1449305350

[42] Janne Piironen : *Robot Framework Plugin of Jenkins, 2012* , website: https://wiki.jenkins-ci.org/display/JENKINS/Robot+Framework+Plugin [Viewed on 20.07.2012]