OULUN AMMATTIKORKEAKOULU

Joni Huttula

Generator Tool for Operator Test Case

Generator Tool for Operator Test Case

Joni Huttula

Bachelor’s Thesis
Spring 2018
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology and Telecommunications, Wireless devices

Author(s): Joni Huttula
Title of thesis: Generator Tool for Operator Test Case
Supervisor(s): Esa Romppainen (MediaTek), Kristian Lappalainen (MediaTek), Teemu Korpela (Oamk)
Term and year of completion: Spring 2018
Number of pages: 54 + 8 appendices

The subscriber of the thesis is MediaTek Wireless Finland Ltd, and the objective was to develop a tool which helps the developer in his / her everyday development. The goal was to develop a tool which can read and detect SIP messages from a PCAP log file and finally generate a new test case file to a unit test environment. In this thesis test case contained scenarios such as UE registration, mobile originated and terminated call and UE de-registration of IP Multimedia Subsystem.

The tool was developed by using Python language. It was chosen because there was already an existing script written in Python and it was a good platform to start from. The existing script searched the needed parameters to Tshark and printed SIP messages to the text file with little modifications.

The tool achieved the goal set for it and it is a good platform for future improvements. The tool will be part of everyday developing in Oulu and other sites of MediaTek.

Keywords: IMS, SIP, test, programming

TABLE OF CONTENTS

APPENDICES

LIST OF ABBREVIATIONS

3GPP	The 3rd Generation Partnership Project
I-CSCF	Interrogating Call Session Control Function
IETF	The Internet Engineering Task Force
II-NNI	Inter-IMS Network to Network Interface
IK	Integrity key
IMPI	IP Multimedia Private Identity
IMPU	IP Multimedia Public Identity
IMS	IP Multimedia Subsystem
IP	Internet Protocol
IPSec	IP Security
JSON	JavaScript Object Notation
MO	Mobile Originating
MT	Mobile Terminating
P-CSCF	Proxy Call Session Control Function
RFC	Requests for Comments -document
SA	Security Associations
S-CSCF	Serving Call Session Control Function
SDP	Session Description Protocol
SIP	Session Initiation Protocol
TS	Technical Specification
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UE	User Equipment

1 INTRODUCTION

The objective of this thesis is to design a tool which will help developers to create an operator unit test case from a PCAP file. At the moment developers need to do a lot of manual work when creating a new unit test case resembling live network behaviour. To decrease manual work the tool needs to analyse the PCAP file where all the SIP messages are. The log file describes how user equipment (UE) has communicated with the particular network for example in IP Multimedia Subsystem (IMS) registration scenario. Using these test cases developers can be sure that new developed features or bug fixes do not break any already tested feature and the new code behaves as developers wanted. This could reduce costs of developing when travelling to other countries to verify products.

The developed tool is based on an already existing script which provides parameters for third party software Tshark to decrypt messages from the PCAP log. The script also does a little bit of parsing and gives a text file where they are near of “copy and paste” situation. However, this existing script still leaves a lot of manual work because messages are not in final format. The goal for this thesis is to develop a tool by using Tshark to read the PCAP log file, parse and identify a SIP messages and build a new test case file automatically.

When developing a new code for UE, which is connected via IP to a server, there are a lot of moving functions and all of them should be tested and verified before delivering the code to customer. Because of the width and depth of the testing field it was decided that for this thesis scope the tool needs to handle only a few scenarios and provide a good platform to expand the tool later with more testing scenarios. Scenarios under thesis scope are UE IMS registration, mobile originated and terminated IMS call and UE IMS deregistration.

The thesis was started by giving a short view on the IMS architecture of network, demands for a user equipment and protocols of the IMS. It is good to understand the working environment before taking a closer look at the tool and it is always easier to think and understand small details and their impacts when the great picture is understood. After the IMS and its demands, the thesis introduces the test case scenarios which are under the tool's scope. And finally, after introducing the scenarios of the test case, the thesis will focuses on the developed tool.

The subscriber of the thesis is MediaTek Wireless Finland Ltd. MediaTek is a fabless semiconductor company and it has 27 offices in 11 different countries. The company is a market leader in developing tightly-integrated, power-efficient system-on-chip for mobile devices, home entertainment, network and connectivity, automated driving, and IoT. (1, 2.)

2 IP MULTIMEDIA SUBSYSTEM

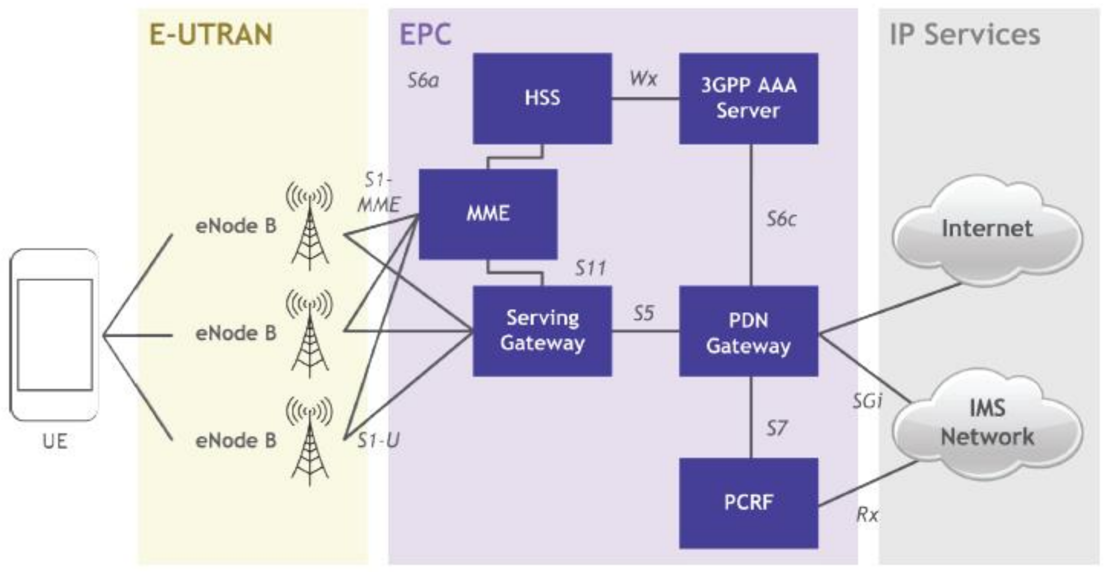


FIGURE 1. A simplified picture of IMS core which provides session and media control (3, pp. 9)

IP Multimedia Subsystem (IMS) is an architectural framework. It is originally designed to deliver Internet Protocol (IP) multimedia to mobile users to evolve Universal Mobile Telecommunications System (UMTS) networks but it has also become a part of many other networks. One of the key protocols is Session Initiation Protocol (SIP) but there are number of protocols which are used to enable multimedia based sessions. As figure 1 shows IMS is an external network and it has its own core network as figure 3 shows. The main elements of IMS core network are listed in figure 2 and they are used for running the basic operations of IMS. (4, 5, 6.)

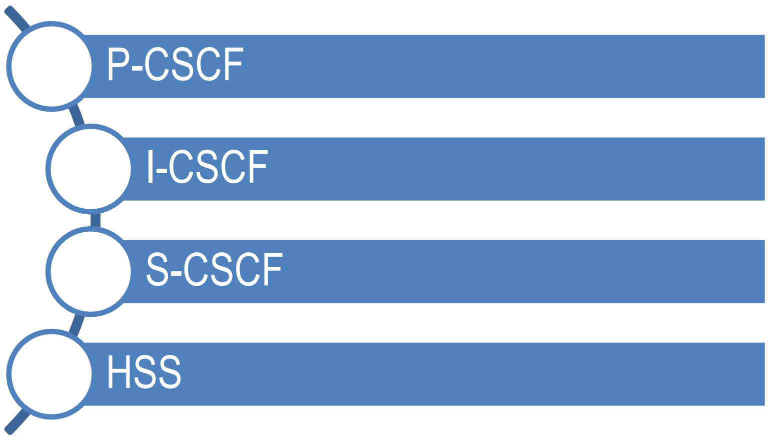


FIGURE 2. The main elements of IMS (5)

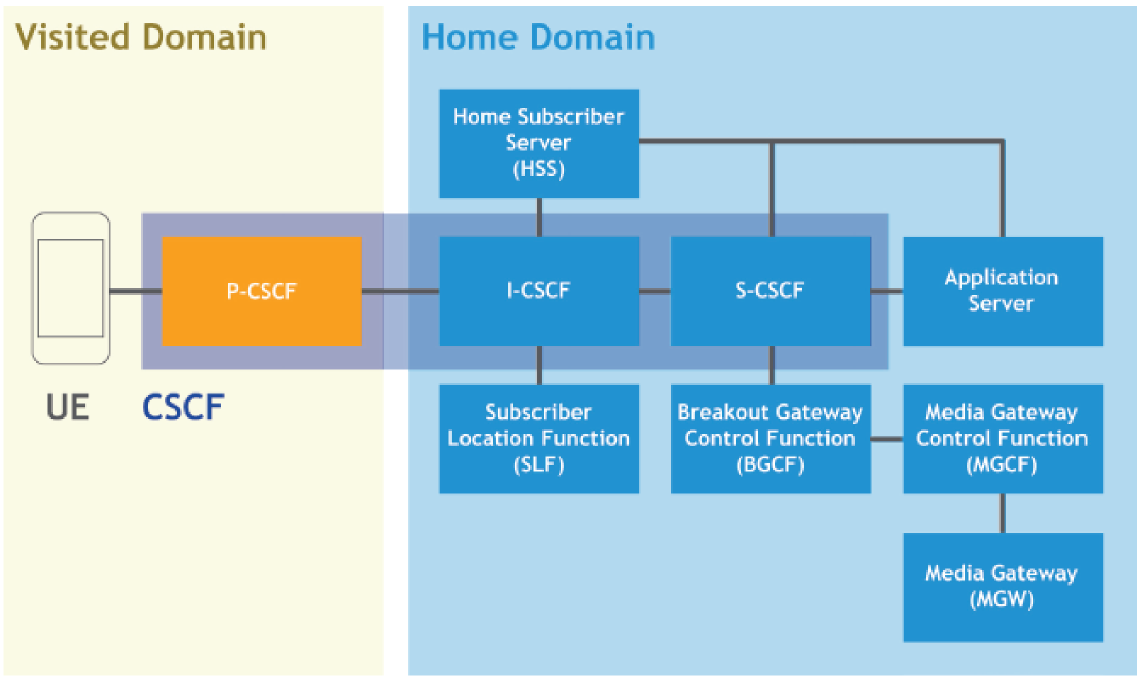

Visited Domain
Home Domain
Home Subscriber Server (HSS)
P-CSCF
I-CSCF
S-CSCF
Application Server
UE
CSCF
Subscriber Location Function (SLF)
Breakout Gateway Control Function (BGCF)
Media Gateway Control Function (MGCF)
Media Gateway (MGW)

FIGURE 3. Simplified functional view of the IMS architecture (3, pp. 9)

2.1 User Equipment

The User Equipment (UE) is part of the IMS architecture and it is used by a user. To be part of the IMS the UE has to have a Universal Integrated Circuit Card (UICC) and a SIP User Agent (UA). The UICC is a smart card which contains one or more applications. The Application can be a Subscriber Identity Module (SIM), a UMTS Subscriber Identity Module (USIM), a CDMA Subscriber Identity Module (CSIM), a Re-Useable Identification Module (R-UIM) or a IP Multimedia Services

Identity Module (ISIM). All of these modules contain user’s identity information which are used by different networks. By default, the IMS uses the ISIM but if the ISIM is not available, next choice will be the USIM or the CSIM. Simply, the ISIM contains a user’s IP Multimedia Private Identity (IMPI), an IP Multimedia Public Identity (IMPU) and a long-term secret. The IMPI is an identity which includes domain information of home operator. The IMPU is an identity which contains user’s public identity such as SIP URI. The long-term secret is a key and it is used for authenticating and calculating a cipher key in the SIP registration process. (3, pp. 6-7; 7, pp. 62.)

The SIP UA is a logical terminal of the SIP network and it is used for managing SIP session from the terminal end. There are two kinds of the SIP UAs: A User Agent Client (UAC) and a User Agent Server (UAS). The UAC is responsible for sending SIP messages and the UAS is responsible for receiving SIP messages but also for sending responses. With these two, the user can for example make a dial or answer to a call invite. (3, pp. 7.)

2.2 Access

There are a couple of requirements for the IMS. To get access to the IMS user must have an IP connectivity. The second requirement is an access independence which means that the user can use any network which provides the IP connectivity to get access to the IMS. In the 3GPP Release 5 there was only a GPRS access but nowadays there are many other options to use services of the IMS, for example an E-UTRAN, a UTRAN, a GERAN, a fixed line, an I-WLAN, a DOCSIS®, a WiMAX™, a cdma2000® and a DVB-RCS2. (7, pp. 11-13; 8, pp. 14.)

2.3 Core Network

The IMS entities can be divided for six main categories: a session management and a routing family (a CSCFs), databases (a HSS, an SLF), interworking elements (a BGCF, an MGCF, an IMS-MGW and an SGW), services (an application server, an MRFC, an MRFP), support entities (a THIG, an SEG and a PDF) and a charging. However, the key functions for the IMS are the CSCFs, the interworking elements and the HSS. The CSCFs contains three other functions: a Proxy-CSCF (P-CSCF), an Interrogating-CSCF (I-CSCF) and a Serving-CSCF (S-CSCF). Together the CSCF family provides the registration of the endpoints. It also provides a routing for the SIP messages. (6; 7 pp. 18; 9, pp. 26.)

The P-CSCF is the first touch for the UE when trying to get access to the IMS and all the SIP messages from or to the UE will go via the P-CSCF. It behaves as a proxy but in addition it also may behave as a user agent. This all means that the P-CSCF verifies every request and forwards them. The P-CSCF also processes and forwards responses from the network. The I-CSCF is a contact point within an operator’s network and it chooses the S-CSCF for user by taking contact to the HSS. It also for example forwards the SIP requests and responses to the S-CSCF. The S-CSCF is the brain of the IMS. It is focused on performing session control and registration services for the UE. (7, pp. 19-22.)

2.4 Protocols of the IMS

In paragraph 4.4 in TS 23.228 by 3GPP it is defined that an IMS uses a Session Initiation Protocol (SIP) for a single session control between a UE and a CSCF. The SIP is an application-layer control protocol which can establish, modify and terminate the multimedia sessions. The SIP can also invite participants to an already existing session. (10, pp. 45; 11, pp. 9-10.)

The SIP does not provide a fully complete communication system. It is more like a component which can be used with other IETF protocols to build the complete multimedia system. These protocols could be for example the Real-Time Transport Protocol (RTP), the Real-Time Streaming Protocol (RTSP), the Media Gateway Control Protocol (MEGACO) and the Session Description Protocol (SDP). (11, pp. 9-10.)

2.4.1 Session Initiation Protocol

The SIP is a text-based protocol and can be either request from a user to a server or response from a server to a user. The SIP message contains a start-line or a request-line, one or more header fields, an empty line and an optional message-body. The start-line, the message-header line and the empty line must include a carriage-return line-feed sequence (CRLF) at the end of the line. The empty line after the message-header line indicates the end of the header fields. (11, pp. 26-27.)

The SIP request message starts with the request-line. The request-line contains a method name, a Request-URI and a protocol version. These are separated by a single space character. The request method name can be for example REGISTER, INVITE, ACK, or CANCEL. More SIP request methods can be found from figure 4. The request-URI can be SIP, SIPS URI or general URI, and it indicates the identity of the user or the server which sent the message. It is possible that a SIP element can also be “tel” URI instead of “sip” or “sips”. The SIP-Version indicates the version of the used SIP. (11, pp. 27-28.)

INVITE	Mandatory	• Used to request connection to the requests receiver.
ACK	Mandatory	• Used to confirm that final response is received.
BYE	Mandatory	• Used to terminate specific session.
CANCEL	Mandatory	• Used to cancel previous request which is sent by the user.
OPTIONS	Mandatory	• Used to query capabilities of the UA or the proxy.
PRACK	Mandatory	• Used to confirm provisional reponse.
UPDATE	Mandatory	• Used to update the session information when session is established but before final response of initial INVITE request has generated.
INFO	Optional	• Used to provide a used mechanism for an application level transport.
MESSAGE	Optional	• Used to deliver instant messages. Content of the message will be in the body of the MESSAGE method.
REFER	Optional	• Used to suggest receiver to take contact to a third party.
NOTIFY	C1	• Used to inform the user about changes of subscription which is created by the SUBSCRIBE method.
PUBLISH	C1	• Used to publish event state.
SUBSCRIBE	C1	• Used to request current state and state updates from a remote node.
REGISTER	C2	• Used to register contact information of user to host.

FIGURE 4. IMS supported SIP methods (11, pp. 16-89; 12, pp. 25; 13, pp. 6; 14, pp. 4; 15, pp. 7-9; 16, pp. 2; 17, pp. 4; 18, pp. 3; 19, pp. 2)

In figure 4 a mandatory tag means that network should at least forward messages over an Inter-IMS Network to Network Interface (II-NNI) if operator's policy applies to them. Nevertheless, network or connected UE does not have to support them. An optional tag means that the message may or may not be supported at II-NNI. A C1 tag means that in a roaming case the message is mandatory but in the other cases it is optional. A C2 tag means that in a roaming case the message is mandatory and in the other cases it is not applicable and it should be discarded by an IBCF in receiving side. (12, pp. 29).

The SIP response message begins with the Status-Line. The Status-Line contains a SIP-Version, a Status-Code and a Reason-Phrase. The Status-Code is a three-number code which indicates how the receiver has understood the requests. The Reason-Phrase is a short textual description of the Status-Code. (11, pp. 28.)

SIP version SIP/2.0 allows six different categories as a response. The first number of the three-digit response code tells which category the code belongs to. 1xx indicates that request has been received and it is in process. 2xx indicates that request has been received, understood and accepted. 3xx indicates that the request has been forwarded for further actions. 4xx indicates that there was a client error and 5xx indicates that it was a server error. 6xx is sign for a global failure. (11, pp. 26-29.)

All the responses which terminate SIP transaction are called as final responses. 2xx, 3xx, 4xx, 5xx and 6xx belong to the final response segment. Another possible response is a provisional response and it indicates the progress of the request. 1xx responses belong to the provision response segment. (11, pp. 21-23.)

All the SIP messages may contain a message body but it is not mandatory. The format of the SIP message body must be given in a Content-Type header if the SIP message has a body. If the message is compressed, the body type must be indicated in a Content-Encoding header field. There are multiple possible protocols for the bodies. One is Session Description Protocol (SDP) and it should be supported in the bodies of INVITE, ACK, PRACK, UPDATE, 200 OK (in the case of INVITE, PRACK and UPDATE requests) and 18x (in the case of INVITE request) messages. (11, pp. 33; 20, pp. 19-23.)

2.4.2 Session Description Protocol

The SDP is used to describe a multimedia session such as a voice-over-IP call (voIP) and a video streaming. However, it is not a transport protocol and it is not for a session content or media encoding negotiation. Purpose of the SDP is only to describe the media format for wanted session. (21, pp. 3.)

There are a couple of requirements and recommendations for the content of description. The SDP description should contain at least a session name and a purpose, a session activity time, the way of data is compressed, and other information which are used to get an access to the media. The description may also contain more information such as the used bandwidth and the contact information of the responsible person of the session. The SDP session description is built by lines of a type and a value as in figure 5. The type is a single case-significant character. The format of the value depends on the type but it may be for example a single number or a string of characters. (21, pp. 5-8.)

```
<type>=<value>
```

FIGURE 5. The form of the SDP session description (21, pp. 8)

The SDP session description is made-up with session-level sections and media-level sections. The description must start with the session-level “v=” and then continue with the media-level section “m=”. The full set of the supported section-level and media-level types are introduced in figure 6. Optional items are marked with “*”. The items must follow exactly the same order than they are shown in figure 6. (21, pp. 8.)

```
Session description
v= (protocol version)
o= (originator and session
identifier)
s= (session name)
i=* (session information)
u=* (URI of description)
e=* (email address)
p=* (phone number)
c=* (connection information -- not
required if included in all media)
b=* (zero or more bandwidth
information lines)
One or more time descriptions ("t="
and "r=" lines; see below)
z=* (time zone adjustments)
k=* (encryption key)
a=* (zero or more session attribute
lines)
Zero or more media descriptions

Time description
t= (time the session is active)
r=* (zero or more repeat times)

Media description, if present
m= (media name and transport
address)
i=* (media title)
c=* (connection information --
optional if included at session
level)
b=* (zero or more bandwidth
information lines)
k=* (encryption key)
a=* (zero or more media attribute
lines)
```

FIGURE 6. Format of SDP. The optional items are marked with "" (21, pp. 8-9)*

2.4.3 Security of SIP Signalling

A reference point is an interface and as TS 23.002 specifies, a Gm reference point is used for the SIP signalling between a UE and an IP Multimedia Core Network (IM CN) subsystem. Security for the reference point is enabled by an IPsec Security Associations (SAs). The SA is established by a negotiation between the UE and the P-CSCF. The negotiation starts with unprotected initial REGISTER request and its 401 unauthorized response. Both of the messages contain parameters which are used for the negotiations. A result of the negotiation is a pair of the IPsec SAs between the UE and the P-CSCF. The SA is a simplex connection that offers security services to the SIP signalling. Protocols for the security services are an Authentication Header (AH) and an Encapsulating Security Payload (ESP). For an authentication and key agreements, the IMS use an IMS Authentication and Key Agreement (IMS AKA) protocol. The IMS uses the ESP instead of the AH for the confidentiality and integrity protection. (7, pp. 143-144; 22, pp. 94; 23, pp. 25-26.)

To obtain the pair of SAs the UE and the P-CSCF need shared keys. The P-CSCF will get these keys from the S-CSCF in the 401 Unauthorized response. These keys are called as an Integrity Key (IK) and a Cipher Key (CK). The P-CSCF needs to remove these keys from the response and forward the message to the UE without the keys. Since the response is received without the shared keys, the UE needs to calculate the IK from the 401 Unauthorized response. With the IK, the P-CSCF and the UE can establish the SAs between four ports. One SA is between the client port of UE and the server port of P-CSCF. The other one is between the server port of UE and the client port of P-CSCF. (7, pp. 144-146.)

Parameters for the negotiation are an encryption algorithm, an integrity algorithm and a security parameter index (SPI). There are also several SA parameters which are not under negotiation. These parameters are a life type, an SA duration, a mode and the key lengths of the IK and the CK. (23, pp. 27-28.)

As TS 33.203, TS 33.210 and RFC 7321 defines, there are several encryption algorithms that must be supported: a NULL, an AES-GCM and an AES-CBC. An AES-CTR and a TripleDES-CBC may be supported. The supported integrity algorithms (called as authentication algorithms in RFC 7321) are a HMAC-SHA1-96, an AES-GMAC and a NULL. An AES-XCBC-MAC-96 should be supported. The SPI value must be a 10-digit number between 0 and 4294967295. Both the UE and the P-CSCF select two different SPIs. One for an inbound SA and one for an outbound SA. (23, pp. 27; 24, pp. 4-5; 25, pp. 12.)

3 SCENARIOS OF TEST CASE

The main point of the operator test case is to test that a new code does not break any SIP features which have been verified in the field tests. Shortly, the test cases describe the signalling between the UE and the network. The test case is partitioned to the scenarios. Each scenario has its own test sequence and one sequence is followed by another. There are number of possible scenarios but figure 7 shows the scope of the test scenarios which have been chosen for the thesis scope. (26.)

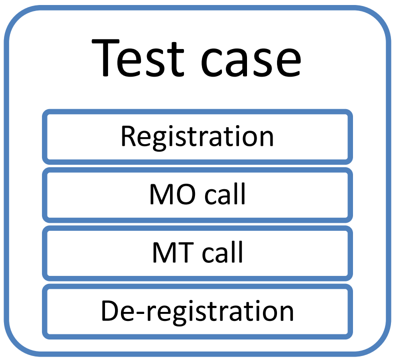


FIGURE 7. List of the test scenarios for thesis (26)

3.1 Registration

After the UE has obtained it's the IP connectivity it can start the registration process. This process will start by generating and sending the REGISTER request to the P-CSCF. The REGISTER request should include information about a Public User Identity, a Private User Identity, a home network domain name, a UE IP address, an Instance Identifier and a GRUU Support Indication. The whole process flow is shown in figure 8. (10, pp. 75; 11, pp. 56.)

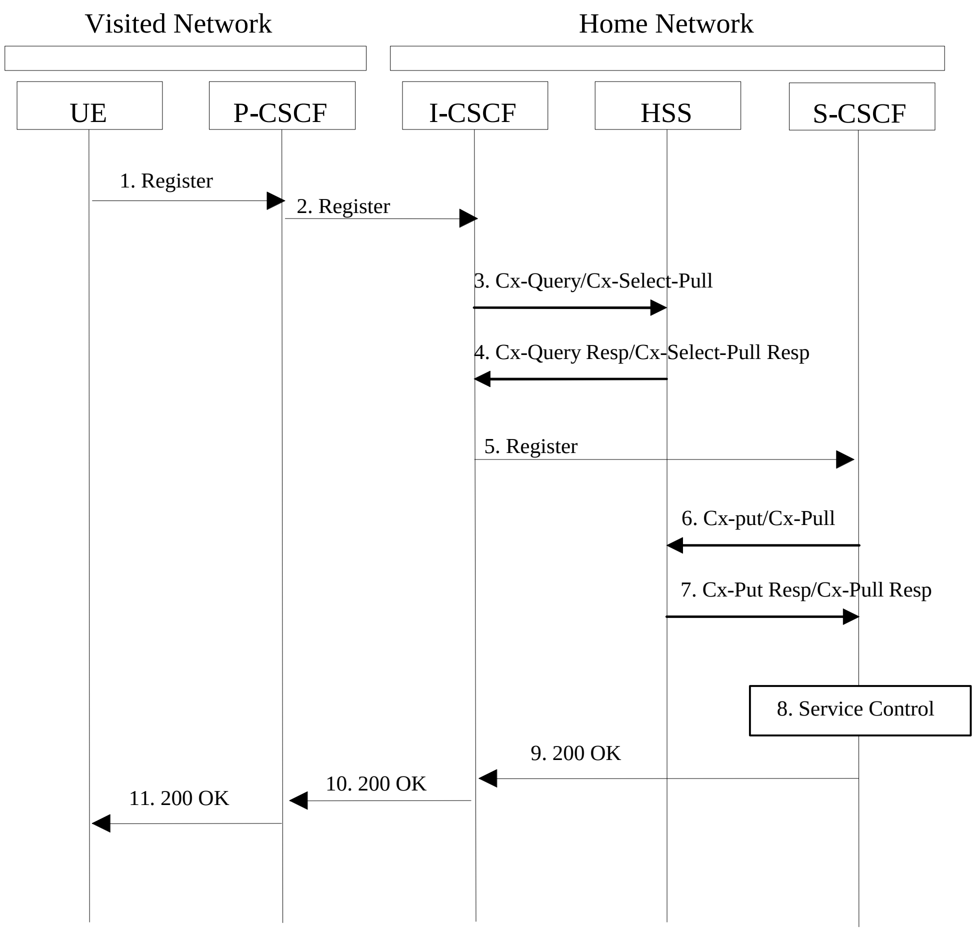
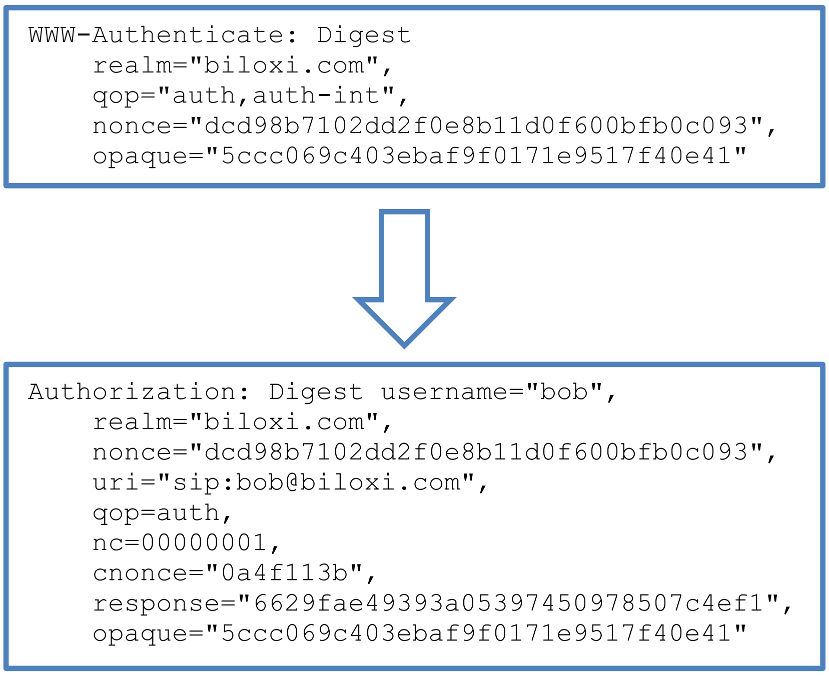


FIGURE 8. Registration flow when user is not already registered (10, pp. 75)

When the P-CSCF receives the REGISTER request from the UE, it searches an entry point to the home network. The entry point could be for example an I-CSCF. When the P-CSCF has localized the entry point of the home network, it modifies the request a little bit and forwards it. The request should contain information as a P-CSCF address/name, a Public User Identity, a Private User Identity, a P-CSCF network identifier and a UE's IP address. The I-CSCF receives the forwarded request and sends a Cx-Query/Cx-Select-Pull information flow to the HSS. The HSS checks whether the UE has already registered or not. If everything is alright the HSS sends a Cx-Query Resp/Cx-Select-Pull Resp back to the I-CSCF which should contain the name of the S-CSCF. When the S-CSCF receives the request, it sends a Cx-Put/Cd-Pull to the HSS. The HSS returns one or more names/addresses which are used to indicate the UE. If the registration is successful the S-CSCF returns a 200 OK response to the I-CSCF. The response of the REGISTER request contains the home network contact information and the GRUU set. The I-CSCF forwards response to the P-CSCF and so on until the UE receives it. (10, pp. 75-76.)

Before the UE receives the 200 OK response it may receive a 401 Unauthorized challenge from the P-CSCF. This may occur by missing credentials from an Authorization header field. For the 401 Unauthorized response the P-CSCF needs to add a WWW-Authenticated header to indicate an authentication scheme or schemes. Figure 9 shows in the upper box an example of the WWW-Authenticated header of a 401 Unauthorized response. With the WWW-Authenticated header field the UE can locate the missing credentials and rebuild a new REGISTER request with filled Authenticated header. The lower box in figure 9 shows an example of an Authorization header of a REGISTER request. (11, pp. 195-196.)



```
WWW-Authenticate: Digest
    realm="biloxi.com",
    qop="auth,auth-int",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
    opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

```
Authorization: Digest username="bob",
    realm="biloxi.com",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
    uri="sip:bob@biloxi.com",
    qop=auth,
    nc=00000001,
    cnonce="0a4f113b",
    response="6629fae49393a05397450978507c4ef1",
    opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

FIGURE 9. An example of 401 response's WWW-Authenticate header and REGISTER request's Authorization header (11, pp. 196)

As it is described in paragraph 5.1.1.3 in TS 24.229 the UE shall send a SUBSCRIBE request for registering a new contact address after getting a 200 OK response for a REGISTER request. For that the UE shall use a default public user identity. The SUBSCRIBE request is used to request a current state and state updates from the network. After the 200 OK response for the SUBSCRIBE request has opened the dialog, the network will send a NOTIFY request. The NOTIFY request is sent to the subscriber to inform the changes of subscription state. Figure 10 shows an example how the whole registration scenario in IMS has been done. (15, pp. 7, 9; 27, pp. 101.)

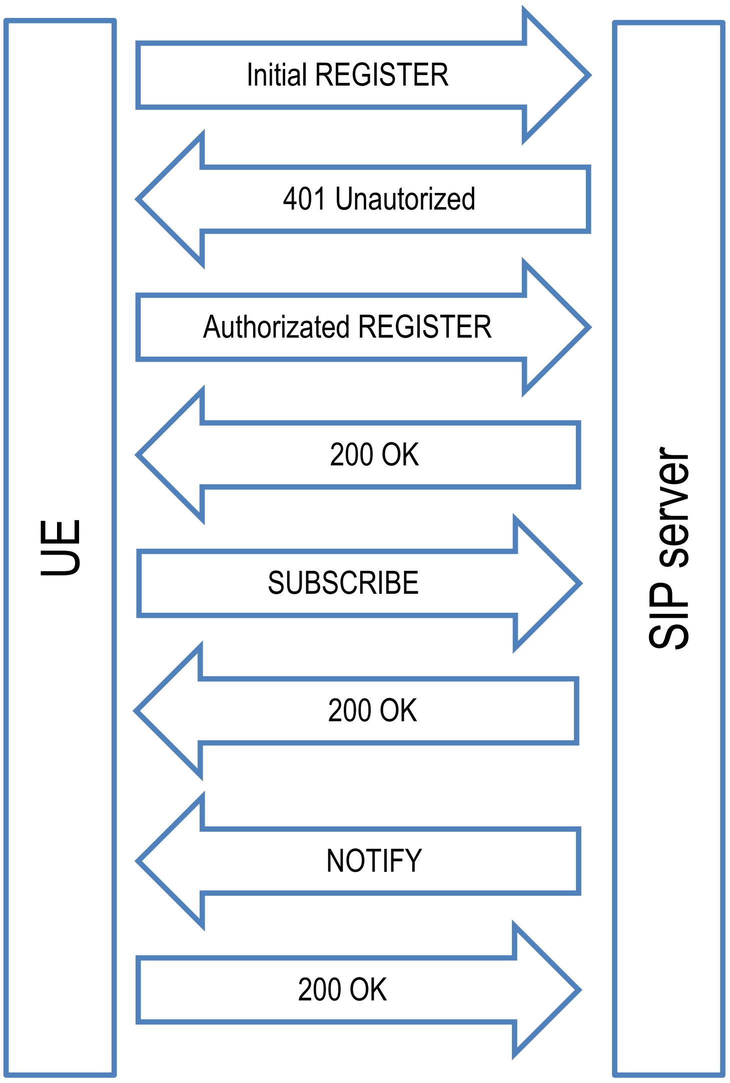


FIGURE 10. An example of IMS registration with the 401 responses (15, pp. 5; 28, pp. 5)

3.2 De-registration

A de-registration process is a reverse of the registration process. To perform de-registration the UE needs to generate a new REGISTER request but now it adds a value of zero to the expiration header. The expiration header and the value of zero is also needed for de-subscribing the subscription of the register state. Flow for de-registering is almost identical with the registration but now the S-CSCF clears the user's information. When de-register is done the S-CSCF sends a 200 OK response to the UE via the I-CSCF and the P-CSCF to indicate the user that the de-registering is successful. Figure 11 shows an example of the de-registration process. (10, pp. 79-80; 15, pp. 7-8.)

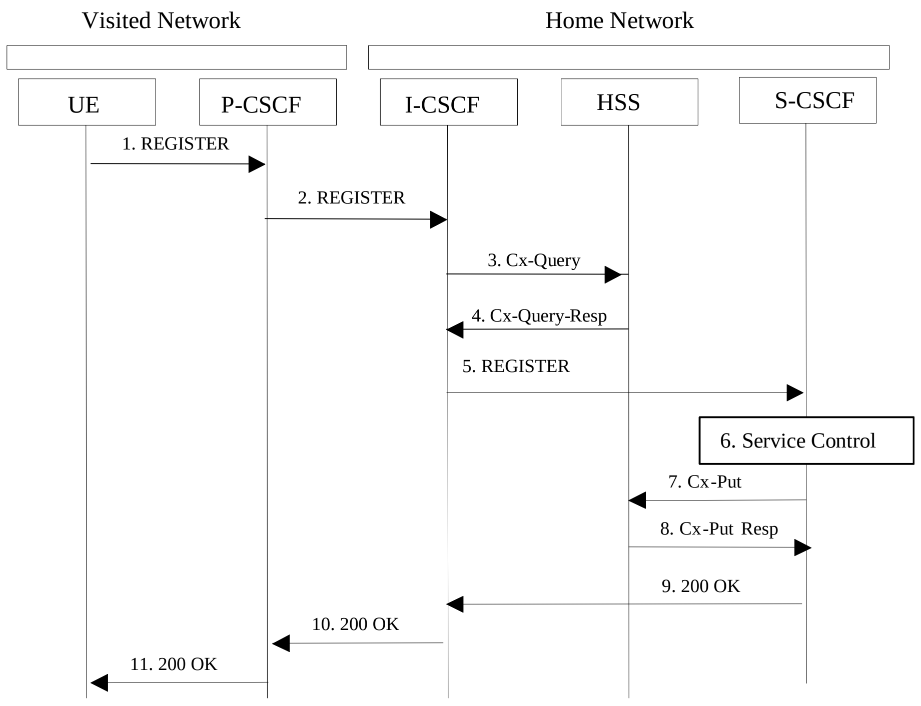


FIGURE 11. Successful de-registration (10, pp. 79)

It is also possible that the network wants to de-register the user. Reason for de-registration can be for example a timeout of registration. The REGISTER request needs to contain an expiration header which tells how long the user wants to be in the registered state. When the time is up, the S-CSCF performs the de-registration procedure by clearing information from the HSS and the user just vanishes from the network. It is also possible that the home network administrative function wants to de-register users from the network. Decisions of de-registering can be made by the HSS or a service platform of the S-CSCF. Figure 12 shows how the service platform performs de-registering. In these cases, the S-CSCF sends information of de-registration to the P-CSCF and the P-CSCF forwards information to the user. The information should tell the reason of de-registration if it is available. When the UE receives information about de-registering, it sends a 200 OK response back to the P-CSCF if possible. Because of multiple reasons it is possible that the UE cannot answer correctly to the de-registering request, the P-CSCF should still perform de-registering without an exception. (10, pp. 81-83.)

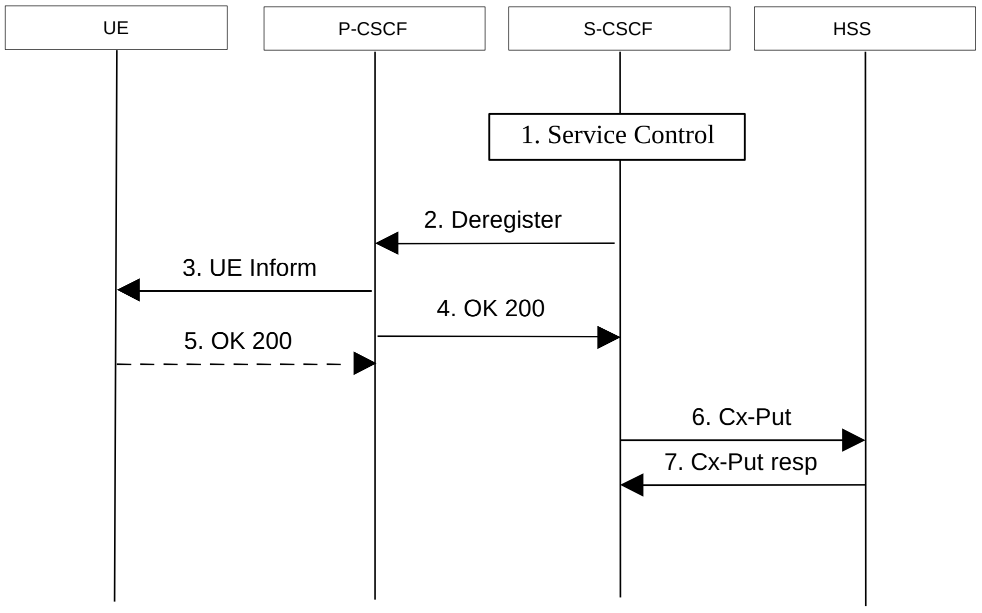


FIGURE 12. A network initiated de-registration by a service platform (10, pp. 83)

3.3 Mobile Originating and Terminating Sessions

As figure 13 shows, for establishing an audio or a voice multimedia session the UE needs to generate an INVITE request. The INVITE request with a final response will establish a session. If there was any error, server will send a 3xx, a 4xx, a 5xx or a 6xx response. Before the final response server can also send a 1xx as a provisional response which is intended to tell a progress of request. It is also described in RFC 3261 that the UE must send an ACK request when it receives the final response. (11, pp. 77-78.)

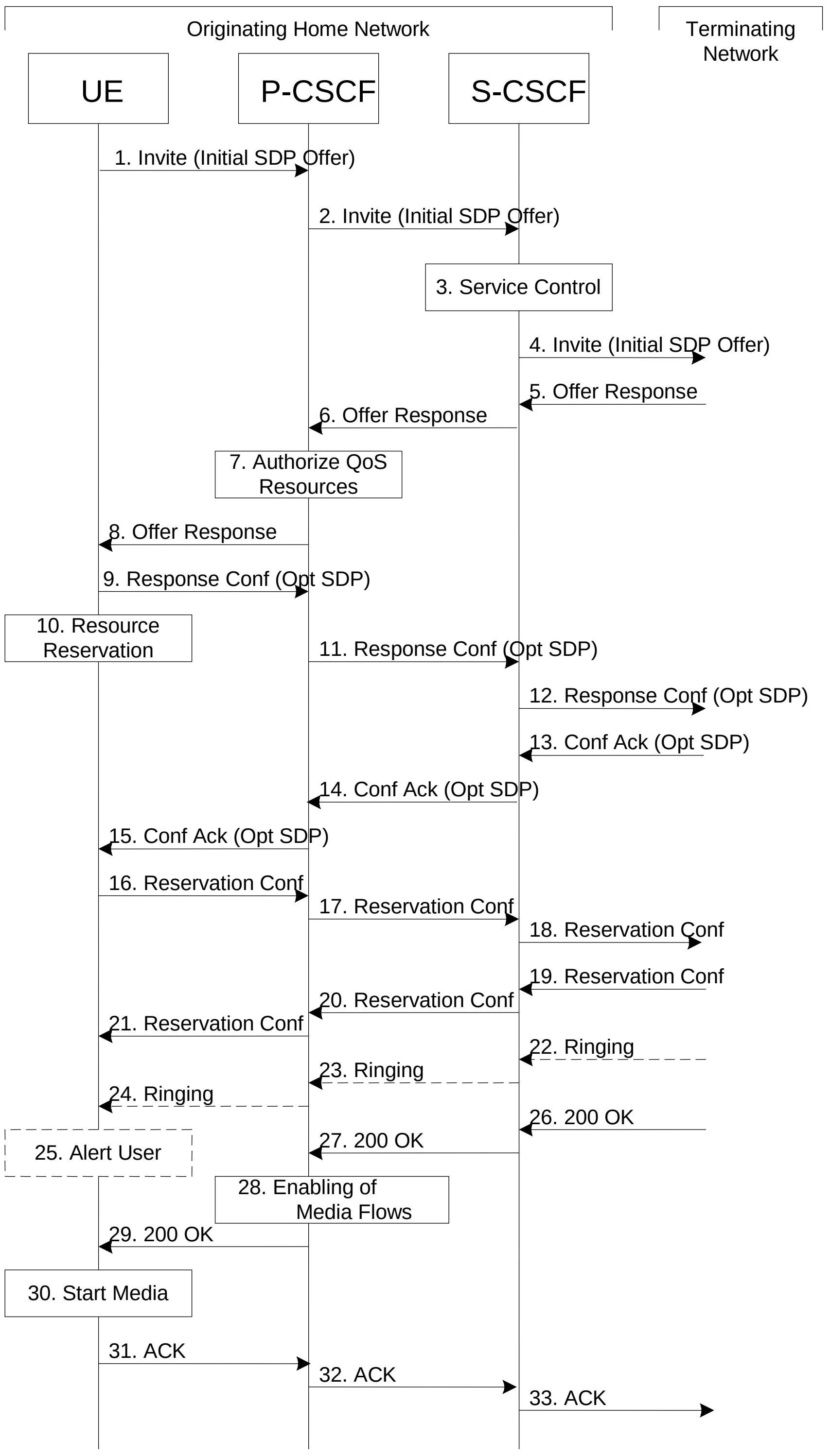


FIGURE 13. A mobile origination procedure in a home network (10, pp. 117)

A mobile originating session process starts when the UE sends an INVITE request which contains a body. The body may be proposal of a multiple type of media for a multimedia session. The P-CSCF receives this request and forwards it to the S-CSCF. The S-CSCF forwards the requests to the wanted terminating network. The terminating network responds by sending back information of a media stream capability of the destination. The S-CSCF forwards response to the UE via the P-CSCF. (10, pp. 117.)

When the UE receives the response, it decides the offered set of media streams for the session and sends a confirmation to the P-CSCF. This confirmation may include an SDP body and it can be the same than the received one. The originating UE can still offer a new media type for the session by sending an UPDATE request. If everything is alright for the originating UE and the confirmation was successful, the terminating endpoint responses with an ACK request. If the confirmation includes a body, the ACK response must have it too. (10, pp. 117.)

When the media negotiation is done, the terminating UE may send a 180 Ringing response to the originating UE. The terminating UE sends a 200 OK response to indicate that a bidirectional audio and a video media flow has started. After the media flow has started the originating UE sends the ACK request as a response for the 200 OK message. Figure 13 shows a mobile originating call procedure in a home network while figure 14 shows the same in a terminating perspective. (10, pp. 117-118.)

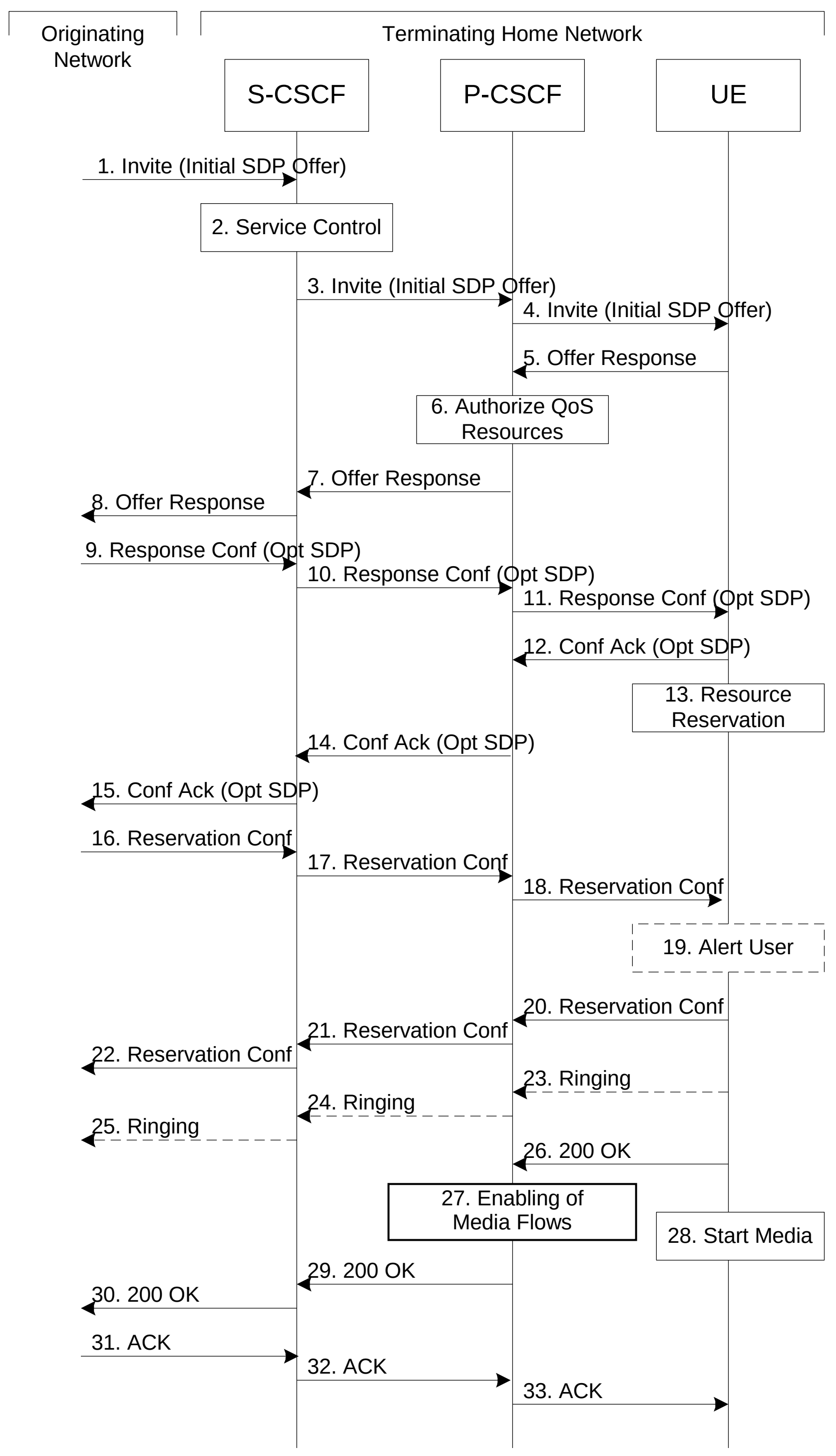


FIGURE 14. A mobile terminating procedure in a home network (10, pp. 129)

4 THE TOOL

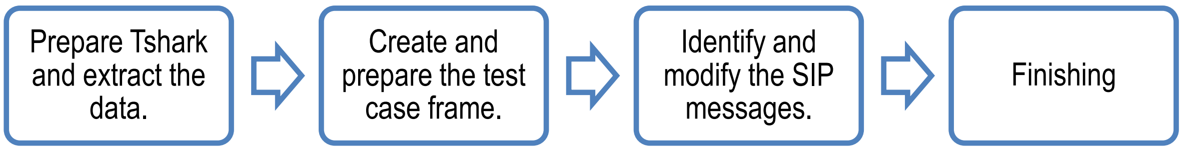
The developed generator tool for operator test case should be able to search decrypt parameters for Tshark, use Tshark software to read SIP messages from a PCAP file and convert them to a JSON file, and modify them to a correct format. With the SIP messages the tool could create a new test case file.

As there was already a script for running Tshark and parsing the SIP messages it was natural to use it as a ground of thesis work. The script was written in python. Python and Tshark are commonly used in the company and thus there should not be any problem to continue using them. The existing script could read the captured PCAP file and return it as the JSON file. From the JSON file, the script searched all the SIP messages and gave a text file where messages were easy to copy and paste to a test case file. A value of the new tool starts from there. With the old script, developer got all the possible messages from the PCAP but the new tool should only search for the necessary SIP messages and leave the unneeded, for example re-transmitted, messages behind. While the tool searches the needed messages it should also generate a new test case file letting developer only to check that a test environment is working correctly for this particular test case without breaking any other test case.

4.1 Body of the Tool

The tool can be divided into four main blocks as shown in figure 15. The first block is for preparing Tshark and extract all the SIP messages to the JSON file. The second block is for creating a copy of test case to the server’s memory as a list. In this block, the tool is also searching for the network’s parameters and build sequences for them. The third block identifies the SIP messages and also modifies them in the test case format. In the modifying part, the tool for example replaces IP addresses with tags. The fourth block is used for finishing the test case and for creating a new test case file. The finishing block for example prints the settings sequences and checks what kind of messages are available to use.

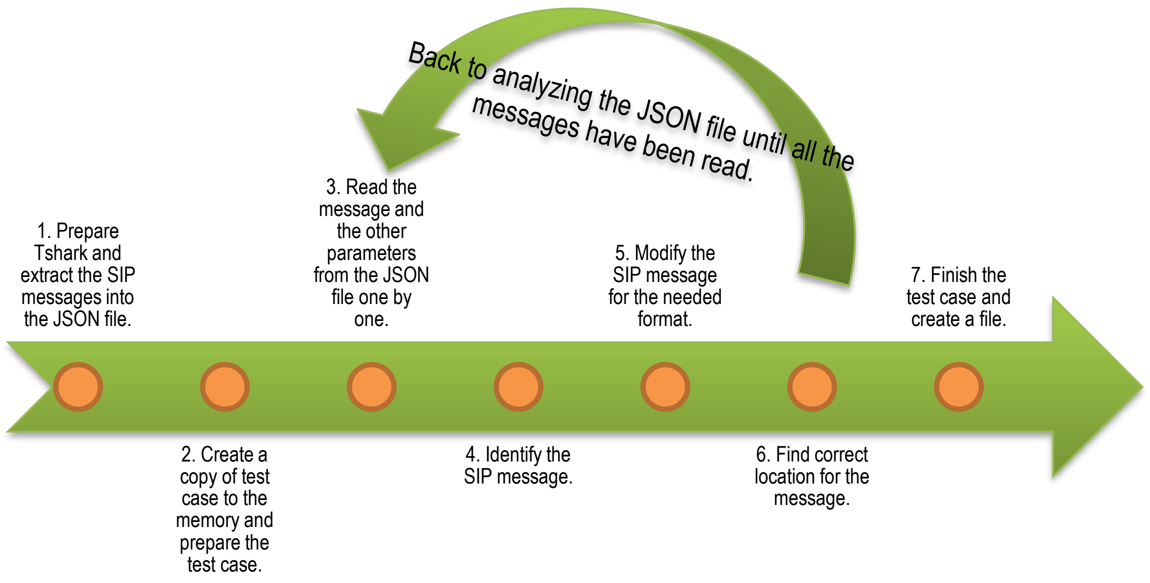
With these blocks the tool can get the SIP messages from the PCAP log file, identify them and create a “ready to execute” test case file.



Prepare Tshark and extract the data.
Create and prepare the test case frame.
Identify and modify the SIP messages.
Finishing

FIGURE 15. The blocks of the tool

Figure 16 goes a little bit deeper than figure 15. It shows how the test case is created step by step. The first two steps are used for preparing Tshark and for extracting the data of PCAP file into the JSON file. The steps three to six are running in loop until the very last SIP message has been analyzed and printed to the test case. These four steps read, identify and print the needed SIP messages to the test case one by one. The tool will recognize the messages which are needed for the registration, the mobile originated and/or the mobile terminated call and the de-registration scenarios.



Back to analyzing the JSON file until all the messages have been read.
1. Prepare Tshark and extract the SIP messages into the JSON file.
2. Create a copy of test case to the memory and prepare the test case.
3. Read the message and the other parameters from the JSON file one by one.
4. Identify the SIP message.
5. Modify the SIP message for the needed format.
6. Find correct location for the message.
7. Finish the test case and create a file.

FIGURE 16. The steps for generating test case

The last step starts when all the messages has been analyzed. On the last step the tool is trying to find out if there are any missing messages or if there are messages that were not named in the template by default. If there are any detection of unused or new messages, the tool modifies the needed test case’s scenario sequence or sequences. The result of the step seven is a file which can be ran in the unit test environment.

4.2 Tshark

Tshark is a network protocol analyzer which can capture data from a live network and read the already captured PCAP file. Unlike a Wireshark, Tshark is a terminal oriented software that is used from a command line. To run Tshark user needs to type “Tshark” and the wanted options to the command line as in figure 17. (29, 30.)

```
Tshark –r filename.cap
```

FIGURE 17. A simple example of Tshark command (30)

To decrypt the SIP messages with Tshark, there is a need to search parameters for Tshark. The parameters are presented in figure 18. The supported encryption algorithms are presented in figure 19 and the authentication algorithms are presented in figure 20. (31)

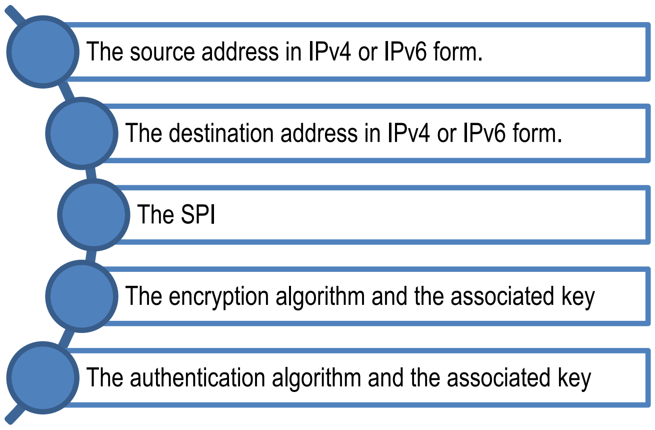

The source address in IPv4 or IPv6 form.
The destination address in IPv4 or IPv6 form.
The SPI
The encryption algorithm and the associated key
The authentication algorithm and the associated key

FIGURE 18. The parameters for Tshark decryption (31)

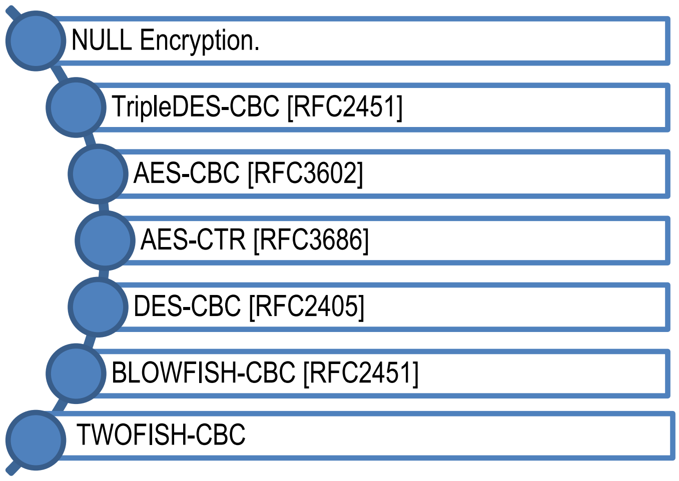


FIGURE 19. The supported encryption algorithms (31)

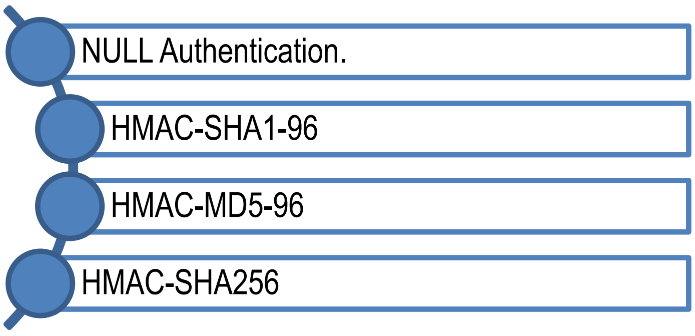


FIGURE 20. The supported authentication algorithms (31)

4.3 JSON

JavaScript Object Notation (JSON) is a text format for the serialization of structured data and is specified in RFC 7159 and in ECMA-404 (32, pp. 3; 33). In this thesis the JSON is used by Tshark to print the decrypted SIP messages from the PCAP log file to an easy-read format. The JSON file is handled in the tool by the JSON library. The following code and figure 21 gives an example how the tool reads JSON file.

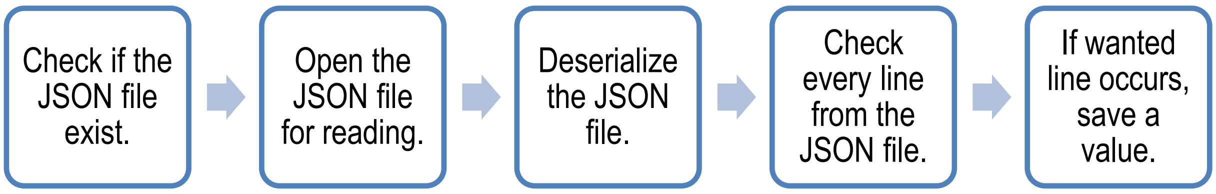

Check if the JSON file exist.
Open the JSON file for reading.
Deserialize the JSON file.
Check every line from the JSON file.
If wanted line occurs, save a value.

FIGURE 21. How the tool reads JSON file

```
#If the output.json file is found.
if os.path.isfile('output.json') == True:
        #Open the output.json file for reading.
        with open('output.json', 'r') as output_data:
            #Deserialize data of the file.
            json_data = json.load(output_data)
            #Check every line from the json_data.
            for i in json_data:
                #If sip.Method in line
                if 'sip.Method' in i['_source']['layers']:
                    #Save the value of sip.Method.
                    data.sip_method = i['_source']['layers']['sip.Method']
```

4.4 Identifying the SIP Messages

In the test case file every SIP message has a specific name to identify the message. This name is built from keywords. There are three basic keywords and they are called as a scenario, a sender and a method. There may also be more keywords like a specifier to indicate that the message is for example authenticated REGISTER request. If the message is a response there may also be a response code and a string to indicate the type of the message. Names of the request messages end with “req” and names of the response messages end with “resp”. Figure 22 shows all the possible keywords and figure 23 shows an example of a name of an initial REGISTER request. Keeping these keywords in mind the tool identifies and names the messages.

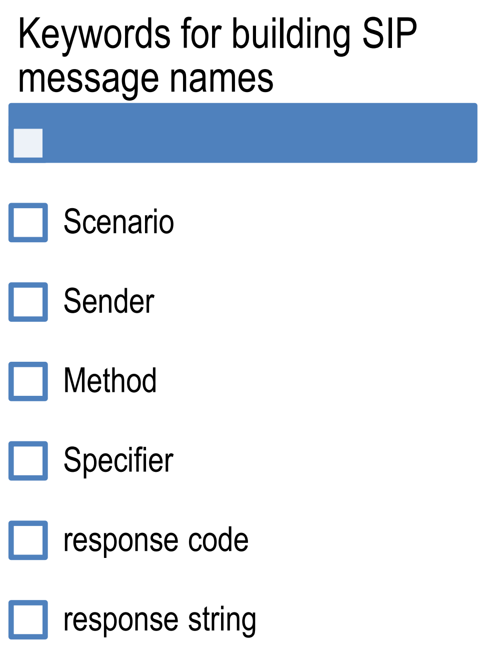


FIGURE 22. The used keywords for generating SIP message names

headers_<scenario>_<sender>_<method>_req

headers_registration_ue_register_req

FIGURE 23. An example of initial registration identifying name

In a code every keyword's value end to the character of "_". This character separates every keyword from each other. If the keyword has no value the name will not be broken. The following example code shows how the name is built in the tool:

sip_name_header = 'headers_' + scenario + sender + sip_method + response_code + specifier + type

In this thesis the possible scenarios are an IMS registration, an IMS mobile originating and an IMS terminating and a de-registration. The scenarios are identified by the REGISTER and the INVITE requests. All the messages which are caused for example by the REGISTER request, belongs to the registration scenario unless the REGISTER request is used for the de-registering. With the INVITE request it is also needed to know on which side of the connection the sender is. If the sender of the request is the UE then the scenario is mobile originated. If the request came from the network, the request is mobile terminated.

The rest of the basic keywords are the sender and the method. The sender is resolved by comparing IP addresses. The sender can be either a UE or a network. The method keyword means the method of the SIP request. It can be for example a REGISTER, an INVITE or a NOTIFY. With the response messages the method is the same as in the request which the message was a response for. However, in the REGISTER request there is a need to do deeper inspections because the request can be either for the registering or the de-registering. To choose which type the REGISTER request is, the tool needs to look inside the REGISTER request message. In the code the method is resolved by the following code but it will be modified later in the code if the scenario is deregistering instead of registering.

```
sip_method = data.sip_CSeq_method.lower() + '_'
```

To know if the message is a request or a response the tool needs to compare the CSeq header method and the SIP method as the following code shows. If they match, the message is a request. Otherwise it is a response.

```
if data.sip_CSeq_method == data.sip_method:
   #The request message
   type = 'req[]'
else:
   #The response message
   type = 'resp[]'
```

If the message is a response the tool checks the value of the response code. The values between 101 and 191 indicate that the response is a provisional response. The values that are 200 or higher indicate that the response is a final response. Knowing the kind of the response and its three digit Status-Code is important when naming for example a PRACK request which is caused from the provisional response.

To identify and to solve the other keywords the tool starts an identifying process by checking the CSeq header’s method and number. As it is described in paragraph 8.1.1.5 in RFC 3261 the CSeq is used for identifying and ordering transactions (11, pp. 38). This means that the tool can use it to identify both request and response messages. For example, if the UE receives a 200 OK message which has “1 INVITE” as a number and a method of CSeq header, it indicates that response was for the UE’s initial INVITE request and the identifying name in the test case may be “headers_mo_call_nw_invite_200_resp”. Using the CSeq header as an identifier, it gives a possibility to write only one function to identify both the request and the response messages. The following titles are reflecting methods of the CSeq method.

4.4.1 REGISTER

There may be several REGISTER requests and responses in one log file. To separate these requests the tool uses several indicators. One is CSeq number. Every request raises CSeq number by one (11, pp. 73). If the initial REGISTER request’s CSeq number is one then the authenticated REGISTER request’s CSeq number would be two. Because every REGISTER request has its own unique CSeq number, responses for the REGISTER request are identified by the CSeq number. For example, if the authenticated REGISTER request was successful with “2 REGISTER” CSeq header value, the network will send the 200 OK response with the same CSeq number and CSeq method as it was in the authenticated REGISTER request.

To be sure if the message was the initial or the authenticated REGISTER, the tool also checks if there is a response parameter with a value RES in the Authorization header. This indicates that the REGISTER request was an authenticated REGISTER request and the request was a response for the 401 Unauthorized response. (27, pp. 108-109.)

In a de-registering case the tool also checks the Expires header value. The Expires header indicates how long the registering is effective. The value of zero indicates the server to start the de-registering process immediately. (11, pp. 61, 171.)

The code to handle the REGISTER request and its responses can be found from appendix 1. Figure 24 shows the logic of the identification.

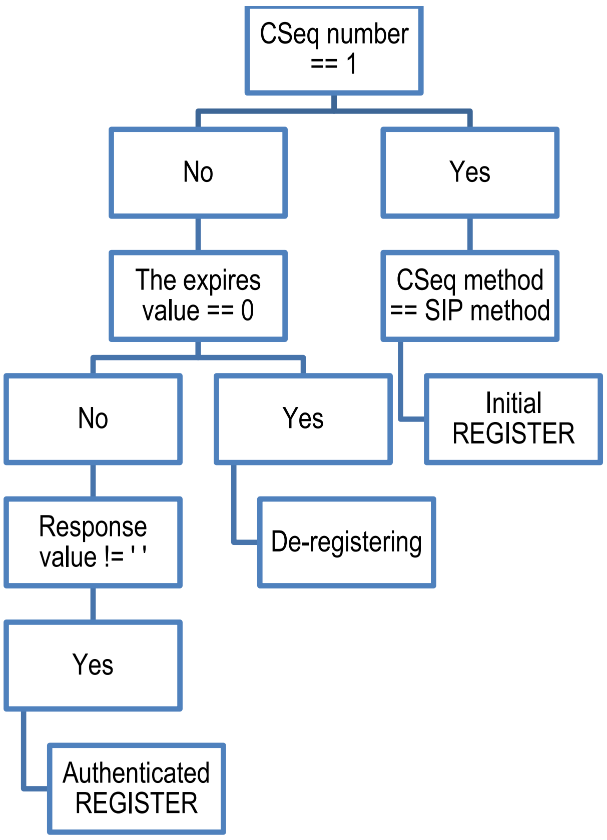


FIGURE 24. The logic to identify the REGISTER request

4.4.2 SUBSCRIBE

Identifications for the SUBSCRIBE requests are the Expires and the Event fields. The Expires header shows the same in the SUBSCRIBE request than in the REGISTER requests. The value of zero indicates de-subscribing in SUBSCIRE request. The Event header shows an event or a class of events that are under subscribing. For example, if the Event header field is pointing to the register state and the Expires header contains a positive number it tells that the message which has a SUBSCRIBE in the CSeq header as a method, is a subscribe for the register state. The tool has to remember the CSeq number from the requests to make an identification for its response messages. The code to handle the SUBSCRIBE request and its response can be found from appendix 2. Figure 25 shows the logic of the identification. (15, pp. 7-8.)

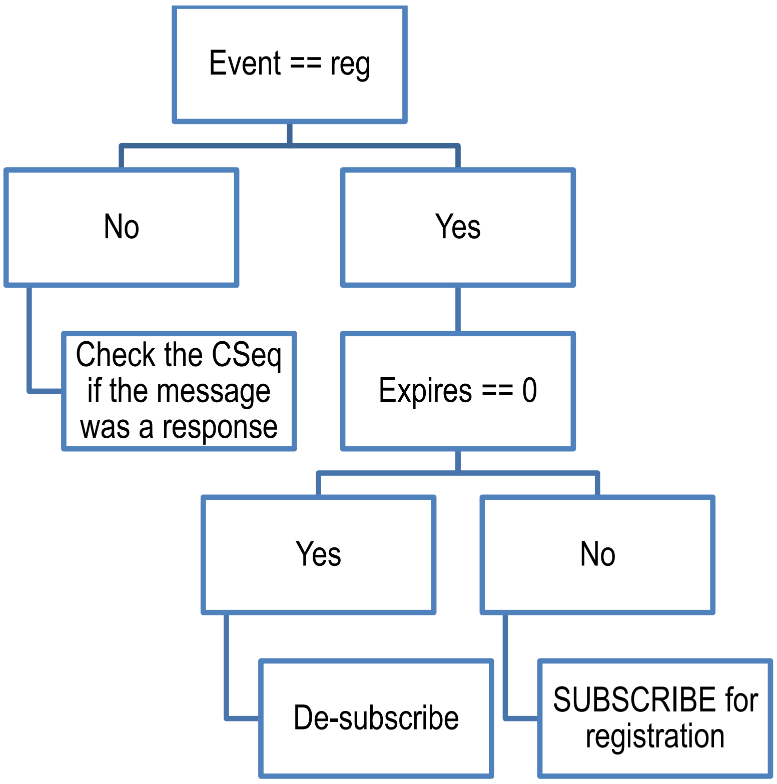

Event == reg
No
Yes
Check the CSeq if the message was a response
Expires == 0
Yes
No
De-subscribe
SUBSCRIBE for registration

FIGURE 25. The logic to identify the SUBSCRIBE request and response

4.4.3 NOTIFY

A NOTIFY request and its response indicators are an Event header and a Subscription-State header. The Event header shows which event or a class of events is under notification. The Subscription-State indicates the state of subscription. The Subscription-State header has three different states: an Active, a Terminated and a Pending state. The “Active” means that the subscription has been accepted and it is valid. The “Terminated” indicates that the subscription is terminated and no longer valid. The “Pending” tells that the subscription has been received but for some reason the subscription cannot be accepted yet. For example, if the Event header is indicating the register state and the Subscription-State has the “Terminated” it means that the message is for the de-registration scenario. The response is identified with the CSeq which indicates the request it is used for. The code to handle the NOTIFY request and its response can be found from appendix 3. Figure 26 shows the logic of the identification. (15, pp. 9, 14-15.)

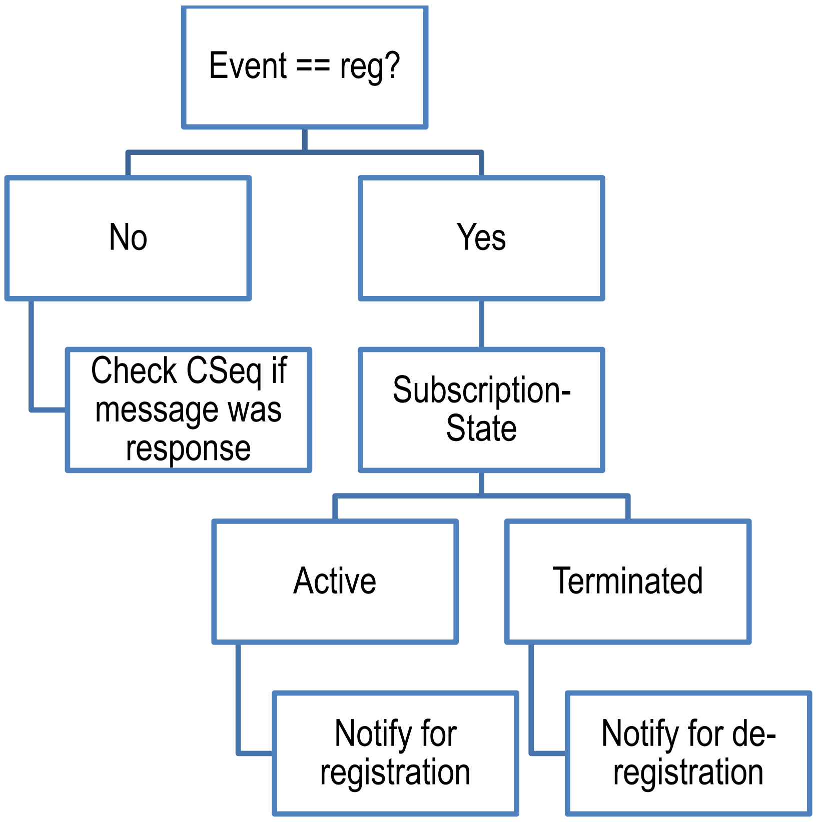


FIGURE 26. The logic to identify the NOTIFY request and its response

4.4.4 INVITE

There are a couple of questions for an INVITE requests and responses which must be answered before identifying the INVITE request. The INVITE request can be either mobile originated or mobile terminated. For this the tool needs to compare the IP addresses of the UE and the P-CSCF to solve the type of the INVITE request. The tool also needs to remember the type of the INVITE request. This information is needed for example with an UPDATE request which the UE uses to update its own INVITE request. This UPDATE request needs for example a mobile originated keyword, “mo_call”, in its identifying name in the test case.

Another question concerns the call. To request a call establishment there are two options for the INVITE request. For the call establishment the sender needs to mark the media stream as a **sendrecv** or leave the body without a direction attribute. The other possible direction attributes which are used to make the difference between the call establishment and the other INVITE requests are **inactive**, **recvonly** and **sendonly**. (20, pp. 29; 34, pp. 18.)

Also, good indicators for the initial INVITE is a tag of **To** header. RFC 3261 refers that To tag must not be contained in a request which is outside of a dialog. A dialog is only created when a request gets a non-failure response. Because of that, the initial INVITE request does not have To tag. The code for handling the INVITE request and its response can be found from appendix 4. Figure 27 shows the logic of the identification. (11, pp. 36, 70.)

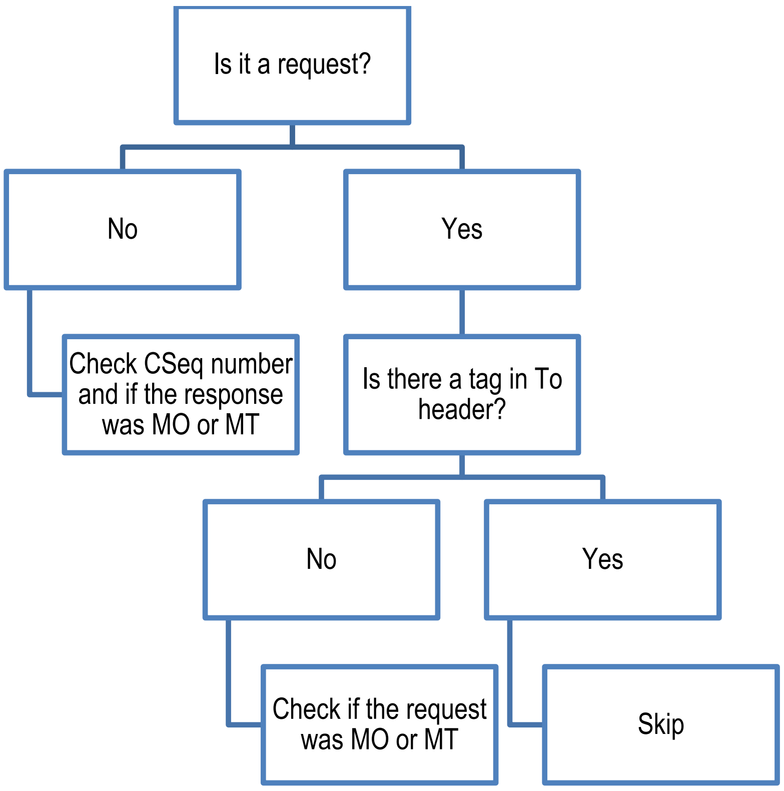

Is it a request?
No
Yes
Check CSeq number and if the response was MO or MT
Is there a tag in To header?
No
Yes
Check if the request was MO or MT
Skip

FIGURE 27. The logic to the identify the INVITE request and its response

4.4.5 UPDATE

Because this thesis does not include any other INVITE requests but the request for a send-receive call, an UPDATE request and its response are easy to identify. The UPDATE request is used to update the session before a final response is sent to the initial INVITE request (19, pp. 2). To know if the UPDATE request is for the initial INVITE request the tool needs to check if the final response was sent or received for the initial INVITE request. If yes, the UPDATE request and its final response is needed for the test case. The code for identifying the UPDATE request and response can be found from appendix 5. Figure 28 shows the logic of the identification.

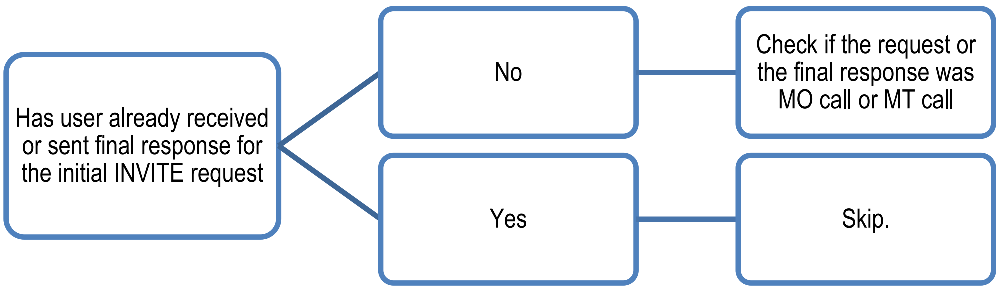

Has user already received or sent final response for the initial INVITE request
No
Check if the request or the final response was MO call or MT call
Yes
Skip.

FIGURE 28. The logic to the identify the UPDATE request and response

4.4.6 PRACK

A PRACK is used to offer reliability for provisional responses (16, pp. 2). If a PCAP file has multiple PRACK requests the tool needs to know which one is the needed request. This is done by comparing the CSeq number of the initial INVITE request and the RAck header's CSeq number of the PRACK request. To generate identifying name for both the response and the request, the tool needs to check whether the message was mobile originated or mobile terminated. The code for identifying the PRACK request and response can be found from appendix 6. Figure 29 shows the logic of the identification.

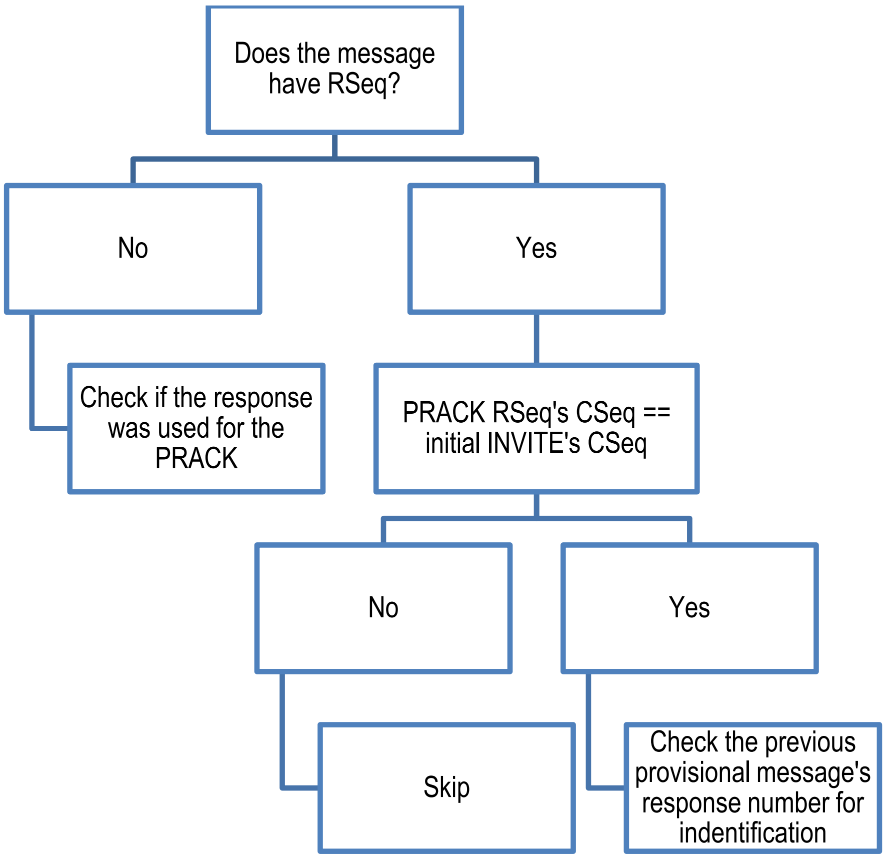

Does the message have RSeq?
No
Yes
Check if the response was used for the PRACK
PRACK RSeq's CSeq == initial INVITE's CSeq
No
Yes
Skip
Check the previous provisional message's response number for indentification

FIGURE 29. The logic to identify the PRACK request and response

4.4.7 ACK

An ACK is used to offer reliability for the final responses (11, pp. 78). With the ACK request the tool needs to check the CSeq header number that matches with the INVITE CSeq number. Appendix 7 gives an example how the ACK is identified. To identify the ACK request the tool needs to check the CSeq number. Figure 30 shows the logic of the identification.

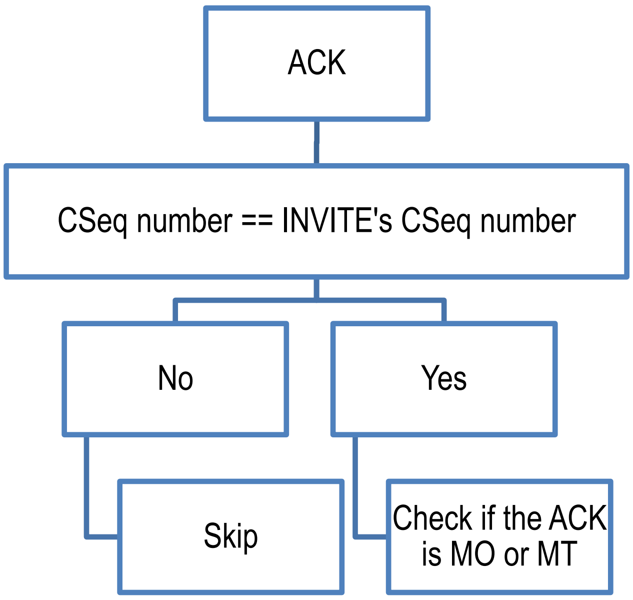

ACK
CSeq number == INVITE's CSeq number
No
Yes
Skip
Check if the ACK is MO or MT

FIGURE 30. The logic to identify the ACK request

4.4.8 BYE

A BYE is used to terminate the session or attempted session (11, pp. 89). With the BYE request and response, the tool needs to check the tag of To header that matches with the dialog’s To header tag and whether the message was mobile originated or mobile terminated. In appendix 8 is the code for identifying the BYE request and response. Figure 31 shows the logic of the identification.

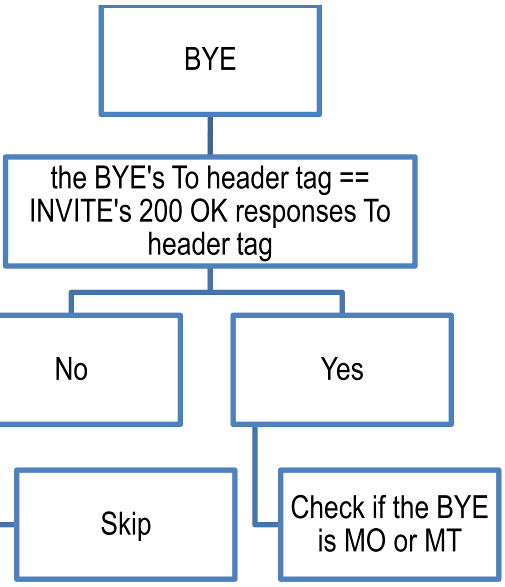

BYE
the BYE's To header tag == INVITE's 200 OK responses To header tag
No
Yes
Skip
Check if the BYE is MO or MT

FIGURE 31. Logic to identity BYE request and response

4.5 The Test Case

The generated test case is based on an already existing template. It is built up with two main blocks and several sub blocks as shown in figure 32. Not all of the sub blocks are introduced in this thesis. The messages block introduces all the test case's SIP messages. The messages block has been organized by scenarios. Figure 33 shows an example of an empty SIP message.

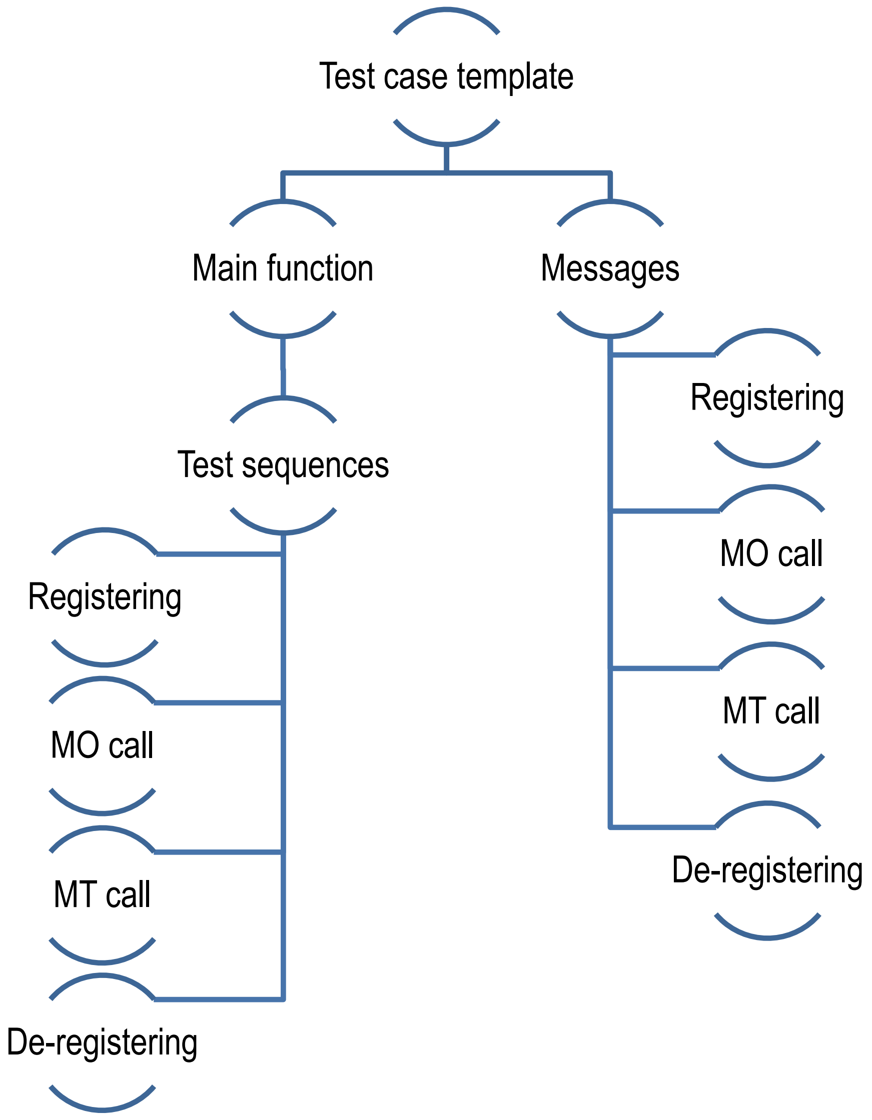


FIGURE 32. The blocks of the template of the test case

```
const char headers_registration_nw_notify_reg_active_req[] = {
    ""
};
const char body_registration_nw_notify_reg_active_req[] = {
   ""
};
```

FIGURE 33. An example of an empty header and an empty body message in the test case template

The second block is the main function. The main function is used to run for example the test environment initialization but also the test sequence. Inside of the test sequence there are four sub sequences, one for each scenario. Figure 34 shows an example of the registration sequence.

```
UT_SEQ_RUN(ua_op_registration_ipsec_seq, ut_data_ptr,
   headers_registration_ue_register_req, headers_registration_nw_register_401_resp,
   headers_registration_ue_register_authorization_req, headers_registration_nw_register_200_resp,
   headers_registration_ue_subscribe_req, headers_registration_nw_subscribe_200_resp,
   headers_registration_nw_notify_reg_active_req, body_registration_nw_notify_reg_active_req,
   headers_registration_ue_notify_reg_resp);
```

FIGURE 34. An example of the registration sequence

To get successful result the tool may still need to modify the test case a little bit. The tool checks if some of the message names do not have the needed content. This may occur because the network did not support these messages. The tool also checks if there are more messages added than in the default template. If some of these cases are detected, the tool needs to modify test case’s sequences a little bit. Figure 35 shows the modified registration sequence.

```
UT_SEQ_RUN(/****INSERT SEQUENCE****/, ut_data_ptr,
   headers_registration_ue_register_req, body_registration_ue_register_req,
   headers_registration_nw_register_401_resp, headers_registration_ue_register_authorization_req,
   body_registration_ue_register_authorization_req, headers_registration_nw_register_200_resp,
   headers_registration_ue_subscribe_req, body_registration_ue_subscribe_req,
   headers_registration_nw_subscribe_200_resp, headers_registration_nw_notify_reg_active_req,
   body_registration_nw_notify_reg_active_req, headers_registration_ue_notify_reg_resp);
```

FIGURE 35. An example of the modified registration sequence

The tool reads the template file into the memory and because the tool is written in python, test case is in list format. The list format makes modifying easier. The tool searches the wanted SIP message name from the list and replaces the next element with the SIP message. Figure 36 shows an example of the list structure from the messages block. Another possibility for adding the SIP messages and the other modifies to the test case is to create a copy of the template and then always create a new file with the modifies when it is needed and delete the old one. Reading the template to the memory and then modifying the elements is easier than always creating a new file with the modifies.

```
const char headers_registration_ue_register_req[] = {\n
```
```
    ""\n
```
```
};\n
```

FIGURE 36. An example of the list of the elements from the test case template after reading the template file to memory

At the end of the test case generating the tool will remove all unnecessary texts, for example C preprocessors and comments, from the file. The tool also generates a name for the file and prints the earlier searched settings sequences to the main function. For generating a name for the file, the tool uses a mobile country code (mcc) and a mobile network code (mnc) which can be found from the PCAP log. It searches the correct country with these parameters from the file where all customers are listed.

5 CONCLUSION AND DISCUSSION

The main objective of the thesis was to develop a tool which can generate a test case file for a company’s unit test environment. For that the tool had to read a PCAP log file using a third-party software, Tshark. The PCAP log is created in the field testing while verifying new UA features in the reference hardware. The PCAP logs describe how the UE communicates with the network. Every operator’s IMS is its own complete system and they all behave differently. This occurs because IP Multimedia Subsystems are built by multiple different hardware vendors where each hardware vendor has its own configuration. Because of that it is very important to test new features and bug fixes that the new code does not break anything. With the generated test case it is possible to test new features before going to verify them on the field and to perhaps save a lot of resources.

The subject of the thesis was not very clear to me before starting but the topic was still very interesting and gave me great opportunities to study more about SIP messaging and IMS. Because of my limited knowledge of IMS and SIP, I started by studying IMS from a general point of view. After I had better understanding what IMS is, I started studying protocols of IMS and in that area focusing on the SIP and the SDP. When the SIP and the SDP were studied it was time to study the unit test environment and the test cases itself. In the test case the main great question was which kind of facts must be kept in mind when developing a new tool. The messages must be kept as original as they were in the PCAP log but still there were needs to modify them for example by replacing some values with tags.

The next great question was related to Tshark. There is a lot of information what is possible to get from Tshark. One way was just to get a message out but for getting analyzing easier it was also needed to get more information without reading them from the message. This kind of information are for example the sender’s IP addresses and the CSeq header method. At the same time when working with Tshark I also worked with identifying SIP messages and tried to find out what is optimal output of Tshark. The final task was trying to get the generated test cases successful in the unit test environment for making sure that the tool works correctly and gives the wanted result.

The goal was to create a tool and a good platform for the future developing and as a result of the thesis there is the tool which can detect messages which belong to the IMS scenarios of registration, mobile originated and terminated call and de-registration. However, there are many messages for other scenarios and a few requests which are not yet included in the tool, for example CANCEL and OPTIONS requests and scenarios of SMS. There is also no support for a compact SIP format which is required in RFC 3261 for implementations (11, pp. 32).

REFERENCES

1. MediaTek Inc. Overview. Available at: https://www.mediatek.com/about/mediatek Date of data acquistion: 13 March 2018.

2. MediaTek Inc. GLOBAL OFFICES. Available at: https://www.mediatek.com/about/office-locations Date of data acquisition: 13 March 2018.

3. Spirent. White Paper. IMS Architecture. The LTE User Equipment Perspective. Available at: https://www.spirent.com/~/media/White%20Papers/Mobile/IMS_Architecture_White_Paper.pdf Date of data acquisition: 27 October 2017.

4. The 3rd Generation Partnership Project. IMS. Available at: http://www.3gpp.org/technologies/keywords-acronyms/109-ims Date of data acquisition: 27 October 2017.

5. Radio-Electronics.com. IMS, IP Multimedia Subsystem Tutorial. Available at: http://www.radio-electronics.com/info/telecommunications_networks/ims-ip-multimedia-subsystem/tutorial-basics.php Date of data acquisition: 27 October 2017.

6. Radio-Electronics.com. IMS Architecture. Available at: http://www.radio-electronics.com/info/telecommunications_networks/ims-ip-multimedia-subsystem/ims-architecture.php Date of data acquisition: 27 October 2017.

7. Poikselkä, Miikka – Mayer, Georg, Khartabil, Hisham – Niemi, Aki 2004. The IMS: IP Multimedia Concepts and Services in the Mobile Domain. Chichester: John Wiley & Sons Ltd.

8. The 3rd Generation Partnership Project. TS 22.228. Service requirements for the Internet Protocol (IP) multimedia core network subsystem (IMS); Stage 1. 3GPP. Available at: http://www.3gpp.org/ftp/Specs/archive/22_series/22.228/22228-f20.zip Date of data acquisition: 3 November 2017.

9. Gutierrez Cesar, Jurisic Andrijana, Yoon Sang Ui, Zoicas Adrian, Arzelier Claude, Boswarthick David, Clayton Michael, van Der Veen Hans, Dietze Claus, Jorgensen Per Johan, Krause Joern, Kymalainen Kimmo, Meredith John, Pope Maurice, Rodermund Friedhelm, Salmeron Lidia, Sasaki Tsukasa, Sultan Alain, Thomasen Gert, Usai Paolo. 2013. Overview of 3GPP Release 5. Summary of all Release 5 Features. Available at: http://www.3gpp.org/ftp/Information/WORK_PLAN/Description_Releases/Rel-05_description_20160107.zip Date of data acquisition: 31 October 2017.

10. The 3rd Generation Partnership Project. TS 23.228. IP Multimedia Subsystem. Stage 2. Available at: http://www.3gpp.org/ftp/Specs/archive/23_series/23.228/23228-f00.zip Date of data acquisition: 5 November 2017.

11. The Internet Engineering Task Force. SIP: Session Initiation Protocol. Available at: https://tools.ietf.org/html/rfc3261 Date of data acquisition: 19 March 2018.

12. The 3rd Generation Partnership Project. TS 29.165 Intern-IMS Network to Network Interface (NNI). Available at: http://www.3gpp.org/ftp/Specs/archive/29_series/29.165/29165-f10.zip Date of data acquisition: 9 November 2017.

13. The Internet Engineering Task Force. Session Initiation Protocol (SIP) INFO Method and Package Framework. Available at: https://tools.ietf.org/html/rfc6086 Date of data acquisition: 9 November 2017.

14. The Internet Engineering Task Force. Session Initiation Protocol (SIP) Extension for Instant Messaging. Available at: https://tools.ietf.org/html/rfc3428 Date of data acquisition: 9 November 2017.

15. The Internet Engineering Task Force. SIP-Specific Event Notification. Available at: https://tools.ietf.org/html/rfc6665 Date of data acquisition: 15.3.2018.

16. The Internet Engineering Task Force. Reliability of Provisional Response in the Session Initiation Protocol (SIP). Available at: https://tools.ietf.org/html/rfc3262 Date of data acquisition: 19 March 2018.

17. The Internet Engineering Task Force. Session Initiation Protocol (SIP) Extension for Event State Publication. Available at: https://tools.ietf.org/html/rfc3903 Date of data acquisition: 9 November 2017.

18. The Internet Engineering Task Force. The Session Initiation Protocol (SIP) Refer Method. Available at: https://tools.ietf.org/html/rfc3515 Date of data acquisition: 9 November 2017.

19. The Internet Engineering Task Force. The Session Initiation Protocol (SIP) UPDATE Method. Available at: https://tools.ietf.org/html/rfc3311 Date of data acquisition: 19 March 2018.

20. GSMA Association. IR.95. SIP-SDP Inter-IMS NNI Profile. Available at: https://www.gsma.com/newsroom/wp-content/uploads//IR.95-v3.0.pdf Date of data acquisition: 20 March 2018.

21. The Internet Engineering Task Force. SDP: Session Description Protocol. Available at: https://tools.ietf.org/html/rfc4566 Date of data acquisition: 14 March 2018.

22. The 3rd Generation Partnership Project. TS 23.002. Network architecture. Available at: http://www.3gpp.org/ftp/Specs/archive/23_series/23.002/23002-e10.zip Date of data acquisition: 8 December 2017.

23. The 3rd Generation Partnership Project. TS 33.203. 3G security; access security for IP-based services. Available at: http://www.3gpp.org/ftp/Specs/archive/33_series/33.203/33203-f00.zip Date of data acquisition: 17 December 2017.

24. The Internet Engineering Task Force. Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH). Available at: https://tools.ietf.org/html/rfc7321 Date of data acquisition: 17 December 2017.

25. The 3rd Generation Partnership Project. 3G security; Network Domain Security (NDS); IP network layer security. Available at: http://www.3gpp.org/ftp/Specs/archive/33_series/33.210/33210-e00.zip Date of data acquisition: 11 December 2017.

26. Mediatek Inc. Creating VoLTE UA operator test cases. Internal document. Date of data acquisition: 11 December 2017.

27. The 3rd Generation Partnership Project. TS 24.229. IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3. Available at: http://www.3gpp.org/ftp/Specs/archive/24_series/24.229/24229-f10.zip Date of data acquisition: 15 March 2018.

28. The Internet Engineering Task Force. Session Initiation Protocol (SIP) Basic Call Flow Examples. Available at: https://tools.ietf.org/html/rfc3665. Date of data acquisition: 1 March 2018.

29. Wireshark. Tshark - Dump and analyze network traffic. Available at: https://www.wireshark.org/docs/man-pages/tshark.html Date of data acquisition: 3 January 2018.

30. Wireshark. Tshark: Terminal-based Wireshark. Available at: https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html Date of data acquisition: 3 January 2018.

31. Wireshark. ESP Payload Decryption / ESP Authentication Checking. Available at: https://wiki.wireshark.org/ESP_Preferences.
Date of data acquisition: 3 January 2018.

32. The Internet Engineering Task Force. The JavaScript Object Notation (JSON) Data Interchange Format. Available at: https://tools.ietf.org/html/rfc7159 Date of data acquisition: 16 March 2018.

33. Python Software Foundation. json — JSON encoder and decoder. Available at: https://docs.python.org/3.5/library/json.html Date of data acquisition: 16 March 2018.

34. GSMA Association. IR.92. IMS Profile for Voice and SMS. Available at: https://www.gsma.com/newsroom/wp-content/uploads//IR.92-v11.0.pdf Date of data acquisition: 2 March 2018.

```
if data.sip_CSeq_method == 'REGISTER':
    #If the CSeq method of the SIP message is REGISTER.
    #If the CSeq number is one then request is initial REGISTER.
    if data.sip_CSeq_num == '1':
        #Add registration_ as scenario.
        scenario = 'registration_'
    if 'Expires: 0' in data.sip_message:
        #If there is "Expires: 0" string in the message content.
        #REGISTER request is de-registration.
        if data.dereg_resp == '0':
            #This checks if there was already another de-registering message.
            #Only first one is needed.
            #Then save message's CSeq number.
            #And add strings as scenario and sip_method.
            data.deregister_CSeq = data.sip_CSeq_num
            scenario = 'deregistration_'
            sip_method = 'deregister_'
        else:
            #If the de-registering message was not the first one.
            print('Another de-registering message. Skipping.')
    elif (re.search('(?<=response=\").*?(?=\",)', data.sip_message)) is not None:
        #If the CSeq was not for the de-registering then check if the message have
        #value in the Authorization header of response.
        #If yes, then message is for
        #authentication.
        if data.reg_auth_resp == '0':
            #For skipping re-registration situations.
            scenario = 'registration_'
            specifier = 'authorization_'
            #Save the CSeq number of the message.
            data.register_auth_CSeq = data.sip_CSeq_num
        else:
            print('Reregistering. Skipping.')
```

```
elif data.deregister_CSeq == data.sip_CSeq_num:
    if data.sip_response_code is not None:
        #De-registering response.
        if 200 <= int(data.sip_response_code) <= 299:
            #If the message's response code is between 200 and 299
            #it is response for the successful de-registration.
            #Raise dereg_resp flag to 1 for indicating that
            #there is no need for others de-registration messages.
            data.dereg_resp = '1'
            scenario = 'deregistration_'
            sip_method = 'deregister_'

elif data.register_auth_CSeq == data.sip_CSeq_num:
    #Authorized registration response
    if data.sip_response_code is not None:
        if 200 <= int(data.sip_response_code) <= 299:
            #If the message's response code is between 200 and 299
            #it is response for successful authorized registration.
            #Raise reg_auth_resp flag to 1 for indicating that
            #there is no need for others registration messages.
            data.reg_auth_resp = '1'
            scenario = 'registration_'
else:
    print('Unknow request/response. (CSeq:', data.sip_CSeq_method,')')
```

```
elif data.sip_CSeq_method == 'SUBSCRIBE':
    #If the CSeq method of the SIP message is SUBSCRIBE
    #Then check if event header of the SIP message is 'reg'.
    #For now only “reg” is allowed.
    if data.sip_event == 'reg':
        #If the message have "Expires: 0" string
        #then its method is desubscribe and the scenario is de-registration.
        if 'Expires: 0' in data.sip_message:
            scenario = 'deregistration_'
            sip_method = 'desubscribe_'
            data.reg_desubscribe_CSeq = data.sip_CSeq_num
        else:
            #If there is no string as "Expires: 0"
            #it is subscribe and scenario is registration.
            scenario = 'registration_'
            data.reg_subscribe_CSeq = data.sip_CSeq_num
    elif data.reg_subscribe_CSeq == data.sip_CSeq_num:
        #This is for response of subscribing.
        scenario = 'registration_'
    elif data.reg_desubscribe_CSeq == data.sip_CSeq_num:
        #This is for response of de-subscribing.
        scenario = 'deregistration_'
        sip_method = 'desubscribe_'
```

```
elif data.sip_CSeq_method == 'NOTIFY':
    #Check if the event header has "reg" string.
    if data.sip_event == 'reg':
        #Check if Subscription-State
        #is active or terminated. Pending state can be skipped.
        if 'active' in data.sip_subscription_state:
            #Save keywords and CSeq number,
            specifier = 'reg_active_'
            scenario = 'registration_'
            data.reg_actived_notify_CSeq = data.sip_CSeq_num
        elif 'terminated' in data.sip_subscription_state:
            specifier = 'reg_terminated_'
            scenario = 'deregistration_'
            data.reg_terminated_notify_CSeq = data.sip_CSeq_num
    elif data.reg_actived_notify_CSeq == data.sip_CSeq_num:
        #For active notify response
        specifier = 'reg_active_'
        scenario = 'registration_'
    elif data.reg_terminated_notify_CSeq == data.sip_CSeq_num:
        #For terminated notify response
        specifier = 'reg_terminated_'
        scenario = 'deregistration_'
```

```
elif data.sip_CSeq_method == 'INVITE':
    #Check if the sip method is INVITE. This is indicating for request.
    if data.sip_method == 'INVITE':
        #Check if there is "a=sendonly", “a=recvonly” or "a=inactive" in SIP
        #message. If yes then pass. Script will skip them.
        if 'a=sendonly' in data.sip_message or 'a=inactive' in data.sip_message or
        'a=recvonly' in data.sip_message:
            pass
        #Check if "a=sendrecv" is in SIP message. This indicate that sender wants
        #to establish the call.
        elif 'a=sendrecv' in data.sip_message:
            #Check To tag. If To tag is missing, INVITE request is initial. If To tag is
            #there then skip.
            if data.sip_to_tag is None:
                #Call is either MT or MO.
                #Save CSeq number
                data.initial_invite = data.sip_CSeq_num
                if sender == 'ue_':
                    #mo_call
                    data.mo_call = '1'
                    scenario = 'mo_call_'
                else:
                    #mt_call
                    data.mt_call = '1'
                    scenario = 'mt_call_'
        #if sendonly, inactive, recvonly and sendrecv are missing then behave as
        #the sendrecv is in message.
        else:
            if data.sip_to_tag is None:
                #Call is either MT or MO.
                #Save the CSeq number.
                data.initial_invite = data.sip_CSeq_num
                if sender == 'ue_':
                    #mo_call
                    data.mo_call = '1'
                    scenario = 'mo_call_'
                else:
                    #mt_call
                    data.mt_call = '1'
                    scenario = 'mt_call_'
    else:
        #For responses.
        if data.mo_call == '1':
            scenario = 'mo_call_'
        elif data.mt_call == '1':
            scenario = 'mt_call_'
```

```
elif data.sip_CSeq_method == 'UPDATE':
    #Check if INVITE request has receive its final response.
    if data.initial_invite_resp == '0':
        #If this is for initial INVITE then check if UPDATE request
        #is either MO or MT.
        if data.mo_call == '1':
            scenario = 'mo_call_'
        elif data.mt_call == '1':
            scenario = 'mt_call_'
```

```
elif data.sip_CSeq_method == 'PRACK':
    if data.sip_RSeq_CSeq_num is None:
        #response
        if data.prack_initial_CSeq == data.sip_CSeq_num:
            #This is for the response of PRACK which was response for 1xx initial
            #invite response.
            if data.mo_call == '1':
                scenario = 'mo_call_'
            elif data.mt_call == '1':
                scenario = 'mt_call_'

            if data.provisional_code is not None:
                response_code = data.provisional_code + '_'
                data.provisional_code = None

    elif data.sip_RSeq_CSeq_num == data.initial_invite:
        #Only PRACK for Initial invite is allowed.
        if data.prack_first_only == '1':
            #Only first PRACK is allowed. There may have for some reason multiple
            #PRACK request with same CSeq number.
            if data.RSeq_num == data.sip_RAck_RSeq:
                #This checks provisional response's RSeq number and PRACK's
                #RSeq number.
                #Check if the messages was either MO or MT.
                if data.mo_call == '1':
                    scenario = 'mo_call_'
                elif data.mt_call == '1':
                    scenario = 'mt_call_'
                #Save the three-digit code of provisional response to the header name.
                #Save PRACK's CSeq number for its response.
                response_code = data.provisional_code + '_'
                data.RSeq_num = ''
                data.prack_initial_CSeq = data.sip_CSeq_num
                #Only first PRACK is allowed.
                data.prack_first_only = '0'
```

```
elif data.sip_CSeq_method == 'ACK':
    #Compare the CSeq number to the INVITE request’s CSeq number.
    if data.initial_invite == data.sip_CSeq_num:
        if data.mo_call == '1':
            scenario = 'mo_call_'
        elif data.mt_call == '1':
            scenario = 'mt_call_'
```

```
elif data.sip_CSeq_method == 'BYE':
    #Check that tag is same as it was in dialog opener response.
    if data.to_tag_initial == data.sip_to_tag:
        #Check if message is either MO call or MT.
        if data.mo_call == '1':
            scenario = 'mo_call_'
        elif data.mt_call == '1':
            scenario = 'mt_call_'
```