RNC Node Level Regression test of Robustness Suite

Author: Niklas Rygart

Supervisors: Johan LUNDBERG
Tobias OECHTERING

June 2011
Abstract

The main objective of this master thesis is to reconstruct randomized rendered scenarios leading to failure in the 3G system.

The incitement is the ability to easily rerun the scenario triggering a failure, take additional traces to isolate the root cause and verify corrections.

This is achieved by utilizing CPEmu (Cello Packet Platform) which emulates a complete 3G system, consisting of the RNC (Radio Network Controller), CN (Core Network), RBS (Radio Base Station), UE (User Equipment), where the three latter are simulated by 3Gsim.

A great variety of errors were discovered during the thesis in the simulated environment. This affected the ability to run traffic unimpededly and thus the quest of reconstructing the randomized rendered scenario. One of the major problems is that 3Gsim generates incorrect mobility events when run with random mobility. Another problem is that the time between the executing event are very tight, which requires an unreasonable high precision of the timing in the environment.

Approximately ten percent of the scenarios were successfully reconstructed. Although the success rate of the test runs were widely spread.

In order to tackle the problems a lot of timing parameters could be tried out, or the more realistic solution is to modify the behavior of the environment itself to prevent unacceptable timings in the random mobility.
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>3rd generation mobile telecommunications</td>
</tr>
<tr>
<td>3GPP</td>
<td>3rd Generation Partnership Project</td>
</tr>
<tr>
<td>3Gsim</td>
<td>3G simulator</td>
</tr>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode</td>
</tr>
<tr>
<td>BD</td>
<td>Broadcast Domain</td>
</tr>
<tr>
<td>CCS</td>
<td>Common Channel</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interface</td>
</tr>
<tr>
<td>CN</td>
<td>Core Network</td>
</tr>
<tr>
<td>CPPemu</td>
<td>Cello Packet Platform emulator</td>
</tr>
<tr>
<td>CT</td>
<td>Common Test</td>
</tr>
<tr>
<td>DCS</td>
<td>Dedicated Channel</td>
</tr>
<tr>
<td>EUL/HS</td>
<td>Enhanced UpLink/High Speed</td>
</tr>
<tr>
<td>FACH</td>
<td>Forward Link Access Channel</td>
</tr>
<tr>
<td>GPB</td>
<td>General purpose Processor Board</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile Communications</td>
</tr>
<tr>
<td>HHO</td>
<td>Hard HandOver</td>
</tr>
<tr>
<td>HSPA</td>
<td>High Speed Packet Access</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>IFLS</td>
<td>Inter Frequency Load Sharing</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISDN</td>
<td>Integrated Services Digital Network</td>
</tr>
<tr>
<td>Iu</td>
<td>UMTS interface between UTRAN and CN</td>
</tr>
<tr>
<td>Iub</td>
<td>UMTS interface between RNC and RBS</td>
</tr>
<tr>
<td>Iur</td>
<td>UMTS interface between RNCs</td>
</tr>
<tr>
<td>MB</td>
<td>Mobility Behavior</td>
</tr>
<tr>
<td>Mu</td>
<td>Management Interface</td>
</tr>
<tr>
<td>Mub</td>
<td>Management Interface RANOS and RBS</td>
</tr>
<tr>
<td>Mun</td>
<td>Management Interface RANOS and CN</td>
</tr>
<tr>
<td>Mur</td>
<td>Management Interface RANOS and RNC</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Mut</td>
<td>Management Interface RANOS and RXI</td>
</tr>
<tr>
<td>OSS-RC</td>
<td>Operations Support System - Radio &amp; Core</td>
</tr>
<tr>
<td>OTP</td>
<td>Open Telecom Platform</td>
</tr>
<tr>
<td>PDR</td>
<td>Packet Data Router</td>
</tr>
<tr>
<td>PSint</td>
<td>Packet Switched interactive</td>
</tr>
<tr>
<td>RAB</td>
<td>Radio Access Bearer</td>
</tr>
<tr>
<td>RACH</td>
<td>Reverse Link Access Channel</td>
</tr>
<tr>
<td>RANOS</td>
<td>Radio Access Network Operation Support</td>
</tr>
<tr>
<td>RAN</td>
<td>Radio Access Network</td>
</tr>
<tr>
<td>RAT</td>
<td>Radio Access Technology</td>
</tr>
<tr>
<td>RBS</td>
<td>Radio Base Station</td>
</tr>
<tr>
<td>RNC</td>
<td>Radio Network Controller</td>
</tr>
<tr>
<td>RNH</td>
<td>Radio Network Handler</td>
</tr>
<tr>
<td>RXI</td>
<td>Radio access network aggregator</td>
</tr>
<tr>
<td>SAI</td>
<td>Service Area Identifier</td>
</tr>
<tr>
<td>SC</td>
<td>Start Cell</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal to Noise Ratio</td>
</tr>
<tr>
<td>SPB</td>
<td>Special purpose Processor Board</td>
</tr>
<tr>
<td>TB</td>
<td>Traffic Behavior</td>
</tr>
<tr>
<td>UEH</td>
<td>User Equipment Handler</td>
</tr>
<tr>
<td>UE</td>
<td>User Equipment</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications System</td>
</tr>
<tr>
<td>URA</td>
<td>UTRAN Registration Area</td>
</tr>
<tr>
<td>UTRAN</td>
<td>UMTS Terrestrial RAN</td>
</tr>
<tr>
<td>Uu</td>
<td>The interface between UTRAN and UE</td>
</tr>
<tr>
<td>WCDMA</td>
<td>Wideband Code Division Multiple Access</td>
</tr>
<tr>
<td>tc_init</td>
<td>Initiating test case</td>
</tr>
<tr>
<td>tc_init_end</td>
<td>End initiating test case</td>
</tr>
<tr>
<td>tc_rob</td>
<td>Robustness test case</td>
</tr>
<tr>
<td>tc_rob_vrf</td>
<td>Verifying robustness test case</td>
</tr>
</tbody>
</table>
1 Introduction

This master thesis was carried out at Ericsson in Kista at the UEH (User Equipment Handler) department within RNC design.

UEH is a subsystem of the control plane in the RNC that manages the UE. The other major control plane subsystem of the RNC is RNH (Radio Network Handler) which manages the radio network and creates the transport channels in between the RNCs and the CN.

The user plane consists of the CCS, DSC and PDR subsystems. CCS controls the Common Channel and DCS the Dedicated Channels. The PDR (Packet Data Router) routes the traffic towards the CN.

The third generation mobile communication system (3G) supports both CS (circuit-switched) and PS (packet-switched) traffic in contrary to prior systems that tend to use separate subsystems for the different traffic types.

In other words the traditional telecommunication services and the evolving internet services are both fully integrated in the 3G system. The expansion in terms of variety of bit rates and bandwidths are also crucial features of the WCDMA (Wideband Code Division Multiple Access) network.

Figure 1: Interoperability of 3G WCDMA with 2G GSM systems [2]

In order to retain a stable continuous mobile service both 2G GSM (Global
System for Mobile communications) and 3G WCDMA work inter-operatively, see Figure 1. The RNC node is one of the NE (Network Elements) in the WCDMA RAN (Radio Access Network). To enable collaboration in between the two generations of mobile telecommunications, backward compatibility is a key feature. More details can be found in [2].

1.1 Background

Due to requirements of higher speed data transfer in the 3G system, new versions of the software controlling the RNC is needed. These software updates also includes new features. To ensure the new versions does not have negative impact on performance or existing features, regression testing comes in handy.

The exception cases in each test scenario that can be evaluated exist in such amount that is unfeasible to cover them all with fixed test scenarios. Therefore random traffic is used to generate the behavior of the UEs. That involves alternating sequences of calling, data sessions and idling, mobility in the RAN network and simulated interruptions such as operation and maintenance.

Regardless of what fault that is triggered, it must be possible to reconstruct the scenario in order to trace the root of the problem. This master thesis investigates a way of accomplishing such an implementation.

The test cases are written in Erlang OTP (Open Telecom Platform) based test framework. Erlang OTP comprises is a development environment for building distributed real-time high availability systems. The actual testing is performed with Common test, which is a framework for automated testing of arbitrary target nodes.

1.2 Scope

- Learn how to write and execute RNC node level test cases in the Erlang test framework
- Design a random test scenario using three UEs
- Find out how to record the sequence of actions done for each individual UE
- Find out how to be able to rerun the exact same sequence of actions in order to be able to recreate faults
2 System overview

2.1 Architecture

The WCDMA RAN for UMTS (Universal Mobile Telecommunications System) is the link between the CN and the UE, see Figure 2. The CN handles mobile subscriber related inquiries, such as balances for prepaid cards and routing of the calls.

The UE involves any type of mobile equipment, such as a laptop or a cell phone. All UE contains a USIM (UMTS Subscriber Identity Module) that contains information from the subscriber including authentication and encryption keys.

2.2 RNC

The RNC works like a coordinator for the different NE in the RAN. [4] The following is the main functionality of the RNC:

- Manage the radio resources and monitor and handle the UE mobility.
- Decide when to execute a handover process - to another cell, frequency or system (e.g. 3G to 2G).
- Control the RAB (Radio Access Bearer) state, which ensures adequate present mode of the UEs. For example if the UE initiates a call, the Speech RAB state is used, while the PsInt (Packet Switched Interactive) RAB state is used when web browsing.
- Support for UE positioning service.
2.3 RAN interfaces

![Diagram of RAN interfaces](image)

There are a bunch of different interfaces for the RAN to communicate with external devices, see Figure 3.

The Iu interface handles the communication between the RNC and the three different domains of the CN - CS, PS, BD (Broadcast Domain).

The air interface Uu is the connection between the UE and the RBS.

The RAN also has some management interfaces that are IP-based - the Mu interfaces, see Figure 4.

Mur (Management Interface RANOS (Radio Access Network Operation Support) and RNC) is the management interface provided by the RNC. The Mub (Management Interface RANOS and RBS) is provided by the RBS. The Mun (Management Interface RANOS and CN) is provided by the CN and within the OSS-RC (Operations Support System - Radio & Core). The OSS-RC collects data from the RNC and RBS and makes it available to the OSS-RC applications. The Mut (Management Interface RANOS and RXI) is provided by the RXI (Radio access network aggregator), which is a type of multiplexer between the RNC and RBS that handles different transport solutions.

The internal interfaces in the RAN for traffic and control signaling is the Iur and Iub. The Iur is used by the RNC for internal communication in between multiple RNCs or RNS (Radio Network Subsystem), while the Iub is used for the communication between the RNC and the RBS. Both these internal interfaces can be used either with the ATM (Asynchronous Transfer Mode) technology or by IP (Internet Protocol).
There are two types of protocol structures for the interfaces (internal & external). One implementing the RAB services - the User Plane protocols. The other manages the RABs and the connection between the UE and the rest of the RAN - the Control Plane protocols. [2]

![Figure 4: WCDMA Radio Access Network](image)

2.4 WCDMA RAN basic features

Maybe one of the most important feature is the emergency call feature. It is possible to setup an emergency call, even in a congested RAN by rejecting regular calls.

To be able to initiate a call admission control is necessary. This also includes incoming handover processes.

When using data packet services, the bandwidth demand tend to vary in time. Therefore there are different RAB states depending on the activity. For heavy usage a dedicated channel is assigned, while a switch to a common channel is performed upon dropped usage. This feature that optimizes the available resources is called channel switching. In addition the switch, to a higher or lower bit rate channel, could take place depending on the coverage and throughput as well.

The ciphering mode control is a feature handled by the CN to ensure secure data transfers between the RNC and the UE. It also coordinates the ciphering keys between the CNs. The currently used Kasumi algorithm (128 bits) was conventioned already in 3GPP '99 (3rd Generation Partnership Project).

Upon overload the congestion control feature is utilized. The feature can for instance order a channel switch to release some resources or in worst case deny new calls or even kick out some sessions. All actions is set with different prior-
ities depending on the grievousness of the congestion.

Another extensive feature for WCDMA to work is the power control. Since the bandwidth is shared it is of great importance that all UE keep an appropriate power level on basis of the corresponding position. For example less power is required to maintain the desired SNR (Signal to Noise Ratio) for UEs close to the RBS, compared to UEs at the cell border.

A positioning service is available through the Iu interface. It is based on the SAI (Service Area Identifier). [5]

2.5 3Gsim

![Figure 5: Testframework overview](image)

3Gsim is a very complex environment offering high-level commands controlling many simulated objects in one command. It is built on CPP (Cello Packet Platform) and runs on number of GPB (General purpose Processor Board) and SPB (Special Purpose Processor Board), see Figure 5. The commands can either be entered straight into the CLI (Command Line Interface) or executed from a command file. In this thesis the commands are executed through a telnet connection which is set up via Erlang. In other words the test framework is used to manoeuvre 3Gsim (and the CPPemu). The complete 3Gsim manual is available in [10].

2.6 Erlang

Erlang is a functional concurrent programming language, which the test cases are written in. It was developed to build massively scalable soft real-time systems with requirements of high availability. Beyond the telecoms it is also used within banking, computer telephony and instant messaging. A four day course was held (and attended) in the middle of March (8-11). Parts of the course
material are available in reference [3]. Erlang has a telecommunication specific platform which is called Erlang OTP with a set of Erlang libraries and design principles that for example enables debugging and inter-language application interface. Reference manual is available in [7].

2.7 Common test

Common test is a portable test server for black-box testing and can also be used for white-box testing in OTP applications and Erlang programs. It provides the possibility to run test suites automatically on local and remote targets in a large scale. The results are presented in a HTML (Hyper Text Markup Language) format. Common test has good prerequisites for regression testing, which are utilized in this thesis. A two day course was held (and attended) in April (6-7). Reference manual is available in [8].
3 Model

The scenario consists of three active UEs and one passive UE. The three UEs have different traffic and mobility behaviors. One of them calls the fourth passive UE. The RAN consists of two RBS on two different frequency bands, each with a total of six cells divided into two cell groups of three cells. The cells have various capabilities (the older standard R99 and the newer HSPA), see Figure 6.

3.1 Test suite

The test suite consists of four main parts:

- \textit{tc\_init}
- \textit{tc\_rob}
- \textit{tc\_rob\_vrf}
- \textit{tc\_init\_end}

Where the first one sets up (initiates) the described cell configuration, defines the traffic behaviors and the mobility behaviors.

The second, \textit{tc\_rob} (robustness test case) activates three UEs, each specified with its own random mobility and traffic behavior from \textit{tc\_init}. Whereupon the actual performed events are traced.

The third, \textit{tc\_rob\_vrf} (verifying robustness test case) starts three UEs with the custom mobility events recorded from \textit{tc\_rob}. After which the events performed
could be verified with the input events.

The last, \textit{tc\_init\_end} revokes the setup in \textit{tc\_init}, to make it possible to run different scenarios afterwards.

### 3.2 Traffic behavior

The traffic behavior defines which services the UEs will use.

Each traffic behavior is configured to loop when the duration (in seconds) of its last traffic activity is elapsed.

The three traffic behaviors used are:

- **\texttt{psint\_fach\_ura\_fach}** sequentially alternating between the four RAB states:
  - 4s \texttt{int\_64\_64} PSint (Packet Switched interactive) 64/64 kb/s (down/up)
  - 4s \texttt{cch} CCH (Control CHannel) Where the PSint is set up
  - 4s \texttt{int\_ura\_ura} URA (UTRAN (UMTS Terrestrial RAN) Registration Area) Disconnects Radio link but but keeps CN resources.
  - 4s \texttt{cch}

- **\texttt{3xPsIntEUL\_HS}** sequentially alternating between the five RAB states:
  - 3s \texttt{int\_eul\_hs} PSint with EUL/HS (Enhanced UpLink/High Speed) capabilities.
  - 3s \texttt{2xint\_eul\_hs} 2 parallel PSint connections
  - 3s \texttt{3xint\_eul\_hs} 3 parallel PSint connections
  - 3s \texttt{2int\_eul\_hs}
  - 3s \texttt{int\_eul\_hs}

- **\texttt{sp\_int}** sequentially alternating between the seven RAB states:
  - 2s \texttt{int\_eul\_hs}
  - 2s \texttt{speech\_12\_2\_int\_eul\_hs} Speech CS using AMR (Adaptive Multi-Rate audio codec) 12.2 kb/s + PSint with EUL/HS capabilities.
  - 2s \texttt{speech\_12\_2\_int\_0\_0} Speech + FACH (Forward Link Access Channel) 0/0 kb/s (down/up)
  - 2s \texttt{speech\_12\_2\_2xint\_eul\_hs} Speech + 2 parallel PSint
  - 2s \texttt{speech\_12\_2\_int\_eul\_hs}
  - 2s \texttt{speech\_12\_2}
  - 2s \texttt{speech\_12\_2\_int\_eul\_hs}

A complete list of available traffic behaviors are specified in the 3Gsim manual [10].
3.3 Mobility behavior

The mobility behavior defines the way the UEs moves in the RAN.

The three mobility behaviors used are:

- **random_cells_3s**
  - `leg_change_interval 3000`
  - `fach_cell_reselection_interval 3000`
  - `ura_cell_reselection_interval 3000`

- **random_cells_4s**
  - `leg_change_interval 4000`

- **random_cells_5s**
  - `leg_change_interval 5000`

Where the three `leg_change_interval` specifies the number of milliseconds to wait before randomly change location. The FACH (Forward Link Access Channel) and URA (UTRAN (UMTS Terrestrial Radio Access Network) Registration Area) reselection parameters are used in combination with traffic behaviors containing CCH RABs to enable mobility of non DCH (Dedicated CHannel).

While randomly moving around in the UTRAN the RNC keeps a list of reachable cells and their current signal strengths. This is called the monitored set cells. The cells currently in use are called the active set cells.

A complete list of available mobility behaviors are specified in the 3Gsim manual [10].

3.4 Events

Mobility events are measurement reports, sent from the UE, intended to be a decision basis for the RNC.

These are the mobility events in use:

1. Intra frequency events
   
   **1a.** Include a new cell to AS (Active Set)
   **1b.** Remove a cell from AS
   **1c.** 1a + 1b
   **1d.** Change of best cell

2. Inter frequency events
   
   **2a.** Change of best frequency
   **2b.** Current frequency bad, another frequency better
   **2d.** Current quality of frequency bad (below threshold)
2f. Current quality of frequency good (above threshold)

3. Inter RAT (Radio Access Technology) events
   3a. Like 2b but for UTRAN (UMTS Terrestrial RAN)

6. UE internal events
   6b. UE transmitted power is not enough for coverage (same result as 2d)
   6d. UE maximum transmittable power reached

3.5 Mobility file
A mobility file contains events underlying decisions of the UE handling made by the RNC.

This is the form of the input to the mobility file generator (1),

\[
[\{"duration","cell","event"\},\{"duration","cell","event"\},...]
\]

where duration is the time in milliseconds to wait after executing the event, cell is the ID of the cell, event is one of the events listed in 3.4.

If the event is one of the double events, i.e. 1c, the string cell consists of two cell IDs, separated with a dash, e.g. "1014-1011". Events where the cell is implied, the cell string will solely contain a dash ".-".

Instead of defining a mobility behavior, in the 3Gsim command that creates the UE, a mobility file can be specified. This is utilized in the \texttt{tc\_rob\_vrf} test case.

3.6 Cells
When an UE is created with mobility a start cell is specified. Depending on which mobility event received from the UE, the RNC makes a decision whether change of frequency or cell is appropriate. If a cell does not have support for HSPA (High Speed Packet Access) and the UE requests a change to DCH the RNC will take the decision to make a HHO (Hard HandOver) to the corresponding cell on the other frequency band within the same location.

3.7 Configuration & fine tuning
Initially the time interval for the three UEs were equally spaced with two, three and four seconds respectively for the traffic behavior. The interval for the mobility behaviors were set to four, three and five seconds. Unfortunately there were some issues with the concurrent execution of the commands in 3Gsim, which led to a solution with behaviors containing intervals that did not match the multiples of the other UEs. Later on a checkpoint upgrade was performed and this did no longer seem to affect the outcome and were consequently changed back to its initial values.
The 3Gsim command (2) stops all UEs.

\[ 3g\text{sim stou } - i \text{ all} \]  

(2)

This is executed before the end of the simulation. Unfortunately the command caused 3Gsim to hang. A possible solution was to stop the UEs one by one (3) to prevent the hanging. This approach was tried out, but was not as successful as reckoned.

\[ 3g\text{sim stou } - i \text{ 000001} \]  

(3)

A lot of synchronization problems arose as a consequence to a checkpoint upgrade of the network, which triggers a version change in both 3Gsim and the RNC node. This affected the whole test department for a several weeks and caused problems to run traffic at all. The upgrade was necessary to be able to make use of the new functions that the test case required. One possible workaround to this was to reset the RBS and redefine them again. But that did not seem to have the desired effect on the synchronization problems. Later on another attempt was applied to fix this issue. The approach was to restart 3Gsim after changing the SPM (Special Purpose Processor Module) allocation for all cells to reside on the same SPB (Special Purpose Processor Board). Unfortunately, this did not cure the problems ether. Finally the fault was isolated to the Iub frame synchronization triggered by 3Gsim.

In combination with the synchronization problems the license key for the RNC application expired, which contributed to some errors that was quite hard to track, because there were no error message generated by the void license key.

To facilitate verification of the test case runs two post checks were added to the suite `tc_rob_postcheck_tgsim` and `tc_rob_postcheck_exc_err`. Where the prior shows exceptions in 3Gsim. For example if incorrect RAB states were established or a mobility event could not be performed. While the latter shows the if there were any exception errors in the RNC. Two of these post checks were also added to the reconstructing test case.

There was a delay in the start of the random mobility of approximately 4.5 seconds compared to the fixed mobility that started immediately. Therefore a start delay was added to the mobility file to compensate for this.

This turned out to be harder to specify than just changing the start delay of the mobility events once. Sometimes when the emulator was restarted the start time was drifting which required a reconfiguration of the delays that was really time consuming.
4 Results

The test case tc\_rob parsed the output of the 3Gsim log and created a mobility file successfully.

In the test case tc\_rob\_vrf the scenario was reconstructed by using the mobility file generated in tc\_rob with some difficulty. Sometimes the UEs hanged which resulted in a discontinuation of execution of all remaining mobility events.

The hangings were in most cases caused by the mobility event 2b, when the cell the event intended to change to was not listed in the monitored set cells, because the measurement report, sent by the UE, arrived of some reason at the very same moment (millisecond) as the change of frequency was ordered. Since the precision of the execution was not sufficient (below one millisecond), a 50 ms delay was introduced before any 2b events was executed. This was compensated by subtracting 50 ms from the event after to prevent affection of the remaining events.

Initially the start cells were set manually. But in order to reduce affection of the different capabilities of the cells, a function choosing random start cells for each UE was implemented.

Even if the delays were configured properly upon every restart, there were non-executing events, mostly because the scenario looked differently for every test run. One set of configurations with random start cells and 6900, 9150 and 10250 milliseconds start delay of the mobility for the three UEs respectively, gave the result presented in Table (1). The table shows one test run per line with the amount of executed mobility events in respective test case. The right column tells whether there was a match between the test cases or not. The gray background indicates that the whole test case was reconstructed successfully.

Another set of configurations with random start cells and 2400, 3400 and 4400 milliseconds start delay of the mobility for the three UEs respectively, gave the result presented in Table (2).

The FACH and URA reselection intervals in the mobility behavior for the first UE did not work properly, because reselection of cells in URA and FACH is just possible with random mobility, and cannot be specified in custom mobility. So the reselection interval for URA and FACH were set to 100000 ms to prevent change of active cell during URA and FACH.

Yet another error was discovered. 3Gsim random mobility generates incorrect events in combination with Hard Handover. 3Gsim is sending mobility 2d to make a cell the best cell even though it is no longer part of the active set. A TR (Trouble Report) has been issued on 3Gsim and until it is solved, HHO have to be disabled and thus also the inter frequency coverage relations. Table 3 shows the result of those test runs.

Since the inter frequency coverage relations might have changed the premises of the test, a test without the delay for the 2b events were run. The results are
<table>
<thead>
<tr>
<th>rob</th>
<th>rob_vrf</th>
<th>rob == rob_vrf != 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>UE1</td>
<td>UE2</td>
<td>UE3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1: Success rate (number of mobility events) for test runs with start delay 6900, 9150 & 10250 ms. (Crashed test runs suppressed.)
<table>
<thead>
<tr>
<th>rob</th>
<th>rob_vrf</th>
<th>rob == rob_vrf != 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>UE1</td>
<td>UE2</td>
<td>UE3</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Success rate (number of mobility events) for test runs with start delay 2400, 3400 & 4400 ms. (Crashed test runs suppressed.)

<table>
<thead>
<tr>
<th>rob</th>
<th>rob_vrf</th>
<th>rob == rob_vrf != 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>UE1</td>
<td>UE2</td>
<td>UE3</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Success rate (number of mobility events) for test runs without inter frequency coverage relations. (Crashed test runs suppressed.)
shown in Table 4.

A suspected fault regarding the reuse of the \textit{tc\_init} run in between each test case led to another test run with \textit{tc\_init\_end} included in each test group to reset the network. Results are shown in Table 5.
Table 4: Success rate (number of mobility events) for test runs without 2b delay. (Crashed test runs suppressed.)
Table 5: Success rate (number of mobility events) for test runs with `tc_init_end` included. (Crashed test runs suppressed.)
5 Conclusions

Due to timing inconsistency an exact scenario is hard to regenerate in this simulated environment. But to regenerate a scenario that will trigger roughly the same scenario, hopefully triggering the same faults, is possible under the right circumstances.

One prerequisite is that the initial mobility event will be executed at the same time for all runs of the tc.rob_vrf cases. Another is that the random mobility renders correct events. Today 3Gsim generates faulty events, which leads to the infeasibility of the custom mobility to generate the same (faulty) sequence of events. In other words the few times 3Gsim manages to generate the events correctly the custom mobility will be able to reconstruct that scenario and consequently pass that particular test run.

One of the faulty behaviors showed when the random mobility executed events at the exact same millisecond, which should not be possible. For the reconstructing test case to be able to rerun the same scenario it has to have a precision of less than one ms. For a realistic scenario 100 ms or at most 50 ms of precision could be achieved, because the time to establish or change a RAB state slightly varies. Sometimes this problem arises with a few milliseconds difference, which still makes it very difficult to get exact same sequence. The most frequently events with this behavior is the 1d and 2d events.

One way to ensure the premises are fulfilled is simply to disregard the test runs where the delay differs between tc.rob and tc.rob_vrf. The disadvantage of this method is that it will require a lot of test runs in order to be able to draw some conclusions whether the suite fulfills the criterion.

Further with this method we do not know how many tests to be run to get one useful case. But that can be solved by making a function that checks if the prior test run had matching initial events.

Another thing to take into account is what size of the delay to accept to consider it as a successful one. This can be tricky because in a simulated environment there will always be some time difference. A rule of thumb could be to at most accept a third of the smallest trigger or interval of the traffic/mobility behaviors.

Since the timing is different results are varying a lot, external help for analyzing what the source of the fault is needed. One strategy is to modify the timing of the random mobility events in order to prevent the executed events to be too tight. Although this will require a new revision of the whole 3G simulator.

Until then the only decent solution seems to try out a tremendous amount of parameters in 3Gsim and the RNC in hope for a combination that gives a precision sufficient to recreate the scenario.
6 Discussion

Further investigation of whether the license based tool QuviQ check [9], that is used for automated testing, could be utilized to run a regression test of the kind performed in this thesis.

Quviq check also comes with the advantage that it is able to test against models of mobility. That enables verification of the cell capabilities utilization. For example it would be possible to discover passed test cases handled in a non optimal way.

Due to the varying success when running test cases with identical setup, there is most definitely something that is misconfigured in the environment.

In order to find out if there is a way to solve the environment issues and suppress the time difference between the initial event of tc Rob and tc Rob vrf expert advice of 3Gsim has to take place.
Appendix

This master thesis was supervised by

- Johan Lundberg at Ericsson
- Tobias Oechtering at the Royal Institute of Technology
References


[4] RNC Introduction Flow LZU00000455 Ericsson AB Rev 1A


Test suite

#0. BASIC INFORMATION

- File: ueh_robustness_SUITE.erl
- Description: Test Suite containing randomized robustness test case
  mainly with UEH focus

-module(ueh_robustness_SUITE).

#2. EXPORT TESTCASES

- Test server callbacks:
  See each function below

- export ( [ all/0, suite/0, groups/0, init_per_suite/1, end_per_suite/1, init_per_group/2, end_per_group/2, init_per_testcase/2, end_per_testcase/2 ]).

- Test cases
  Defines all test cases.
  Functions without arguments (xxx/0) are optional.

- export ( [ tc_rob/vrf/1, parse_3gsim_log/2, tc_init/1, tc_init_end/1, tc_rob_postcheck_tgsim/1, tc_rob_vrf_postcheck_tgsim/1, tc_rob_postcheck_exc_err/1, tc_rob_vrf_postcheck_exc_err/1] ).

#3. INCLUDE FILES

- Mandatory include files

  - include_lib("common_test/include/ct_hrl").
  - include( "/vobs/rnc/uteUsers/rcc/rcc_support/include/rcc_def_hrl").
  - include( "/vobs/rnc/uteUsers/rcc/rcc_support/include/rcc_support_hrl").
  - include( "/vobs/rnc/uteUsers/rcc/rcc_support/include/rcc_config_hrl").

- My included files

- My defines
```prolog
% If using real or emu RNC
% % % #4. SUITE LEVEL DEFINITIONS
% % %

% @spec all() -> GroupsAndTestCases | {skip , Reason}
% where
% GroupsAndTestCases = [{group , GroupName} | TestCase]
% GroupName = atom()
% TestCase = atom()
% Reason = term()
% @doc
% Return a list in which order the test case groups and
% the test cases are to be run.
% Note: if a test case is not listed it won't be executed.
% @end
% all() ->
% [{group , tgrp_all}].

% Adding suite level definition
suite() ->
  [{timetrap , {hours , 5}},
   {userdata , [{level , site}]} ,
   {stylesheet , " /home/enikryg/public_html/log_stylesheets.css"}].

% @spec init_per_suite(Config0)
% -> Config1 | {skip , Reason} | {skip_and_save , Reason , Config1}
% where
% ConfigX = [ tuple()]
% Reason = term()
% @doc
% <pre>
% init_per_suite contains initializations
% that are common for all test cases in the suite,
% and that are only to be performed once. The following is done:
% - Open telnet port to RNC and 3Gsim
% - Open MoShell port to RNC
% - Start time server on CPPemu
% - Build up board list in RNC MoShell with command BP all
% </pre>
% @end

init_per_suite(Config) ->
% Enable logging of all commands to CT log
% udda_log : pretty_print_ct (true),
% Compile required modules
ok = RCC_MAKE_RCC,
```
%% Call standard ueh init function
ConfigUpdated = ueh_helper:init_per_suite_ueh(Config),

%% Add any test suite specific init actions here
%% Return test suite config
\{conf, random_conf()\} | ConfigUpdated].

@spec end_per_suite(Config0)
-> void() | \{save_config, Config1\}
where
ConfigX = [tuple()]
@doc
<pre>
end_per_suite is meant to be used for cleaning up after init_per_suite. The following is done:
- Close telnet port to RNC and 3Gsim
- Close MoShell port to RNC
- Delete post check control file
</pre>  
<end>
end_per_suite(Config) ->

%% Test suite specific init actions
%% Call standard function to close mos and telnet connections and disable standard traces
ok = ueh_helper:end_per_suite_ueh(Config),
ok.

@spec init_per_testcase(TestCase, Config0)
-> Config1 | \{skip, Reason\} | \{skip_and_save, Reason, Config1\}
where
TestCase = atom()
ConfigX = [tuple()]
Reason = term()
@doc
<pre>
This part contains actions which must be done before each test case.
The following is done:
- A check is made to see if the testcase name is "te_init" or "te_init_end". No action is performed for those test cases.
- Then a check is made to see if the current test case is a post check test case by checking for string "_postcheck" in the test case name.
- If it is not a postcheck testcase, the following actions are done:
  - Target monitor is started.
  - MoShell undo mode is started to save undo commands to a file.
  - Test case name, start time and allowed postcheck testcases are written to the postcheck control file.
- If it is a postcheck testcase, the following actions are done:
  - A search for the whole test case name is done in the postcheck control file.
  - It is checked if the main test case returned "TcOk".
  - If both criterias above are fulfilled, the postcheck testcase is run, otherwise it is skipped.
</pre>
init_per_testcase (te init, Config) ->
  ueh helper: init_per_testcase_ueh (te init, Config, ?MODULE);

init_per_testcase (te init_end, Config) ->
  ueh helper: init_per_testcase_ueh (te init_end, Config, ?MODULE);

init_per_testcase (TestCase, Config) ->
  ueh helper: check te init run (Config, ?MODULE).
  TgsPid = rcc support: get handle (?RCC_TGS_PID, Config).
  MonConf = ct get config (tgs monitor) ++
  [{telnet, ct get config (host)}],
  MonConfTgs = ct get config (tgs monitor) ++
  [{telnet, ct get config (tgs host)}],
  NewConfig = [{?RCC TESTCASE, TestCase} | Config].
  ok, RncMonPid = rcc th: init_tm (NewConfig, ?RCC_RNC_MONITOR),
  ok, TgsMonPid = rcc th: init_tm (NewConfig, ?RCC_TGS_MONITOR),
  {ok, _} = rcc mos: start mos undo mode (NewConfig),
  case RncMonPid of
    undefined ->
      %% This is postcheck test case- do nothing
      ok;
    _ ->
      %% Stop and delete all UE's in case some are hanging from previous
      %% crashed testcase
      rcc tgs: stop_del_ue (TgsPid, "all"),
      %% Start target monitor
      cpp tm: start_monitor (RncMonPid, MonConf, RncPid),
      cpp tm: start_monitor (TgsMonPid, MonConfTgs, TgsPid),
      %% Add the code for extra traces
      MosPid = rcc support: get handle (?RCC_MOS_PID, NewConfig).
      rcc th: start config traces (TestCase, Config, MosPid)
  end,

  %% Return config parameters
  [{TestCase, TestCase},
   {?RCC_RNC_MON_PID, RncMonPid},
   {?RCC_TGS_MON_PID, TgsMonPid} | NewConfig].

@spec end_per_testcase (TestCase, Config0)

-> void() | {save_config, Config1} | {fail, Reason}

where
  TestCase = atom()
  ConfigX = [tuple()]
  Reason = term()

@doc
<pre>
This part contains actions which must be done after each test case.
The following is done:
  - A check is made to see if the testcase name is "te init" or
    "te init_end". No action is performed for those test cases.
  - Then a check is made to see if the current test case contains string
    "postcheck."
  - If it is not a postcheck testcase, the following actions are done:
    - Target monitor is stopped.
    - Any remaining UE's are stopped and deleted in 3Gsim.
    - All Mo changes are restored by running the test case undo file.
    - The end time and testcase verdict is written to the postcheck control
      file.
</pre>
end_per_testcase(tc_init, Config) ->
    Config;
end_per_testcase(tc_init_end, Config) ->
    Config;
end_per_testcase(TestCase, Config) ->
    RncPid = rcc_support:get_handle(?RCC_RNC_PID, Config),
    RncMonPid = rcc_support:get_handle(?RCC_RNC_MON_PID, Config),
    ModuleBoards = rcc_support:get_boards(module, Config),
    _ = rcc_support:clean_if_testcase_crash(Config),
    case RncMonPid of
        undefined ->
            %% This is postcheck test case - do nothing
            ok;  
        _ ->
            %% Remove traces
            rcc_th:stop_traces(RncPid, ModuleBoards),
            rcc_th:stop_tm(Config, ?RCC_RNC_MONITOR),
            rcc_th:stop_tm(Config, ?RCC_TGS_MONITOR),
            rcc_mos:stop_mos_undo_mode(Config),
            %% Remove any custom traces
            MosPid = rcc_support:get_handle(?RCC_MOS_PID, Config),
            rcc_th:stop_config_traces(TestCase, Config, MosPid)
    end,
    Config.

% @spec groups() -> [Group]
% @doc
% <pre>
% Group = [GroupName, Properties, GroupsAndTestCases]
% GroupName = atom()
% Properties = [parallel | sequence | Shuffle | {RepeatType, N}]
% GroupsAndTestCases = [Group | {group, GroupName} | TestCase]
% TestCase = atom()
% Shuffle = shuffle | {shuffle, Seed}
% Seed = [integer, integer, integer]
% RepeatType = repeat | repeat_until_all_ok |
% repeat_until_all_fail | repeat_until_any_ok |
% repeat_until_any_fail
% N = integer | forever
% Description:
% A test case group is a set of test cases that share configuration functions and execution properties.
% To specify in which order groups should be executed
% (also with respect to test cases that are not part of any group), tuples on the form [group, GroupName]
% should be added to the all/0 list.
% Return a list of group definitions.
% </pre>
% @end
% groups() ->
% [[tgrp_init, []],
%  [tc_init]],
% [tgrp_rob, [sequence],
%  [tc_rob,
%   [tgrp_rob_pc, [parallel],
%    [tc_rob_postcheck_tgsim, tc_rob_postcheck_exc_err]]]],
{tgrp_rob.vrf, [sequence],
 [tc_rob.vrf,
 [tgrp_rob.vrf.pc, [parallel],
 [tc_rob.vrf_postcheck_tgsim, tc_rob.vrf_postcheck_exc.err]]],
 {tgrp_init_end, []},
 [tc_init_end}],
 {tgrp_tcs, []},
 [{tgrp_tc, [sequence], tc_rob, tc_rob.vrf}],
 [tgrp_pc, [parallel],
 [tc_rob_postcheck_tgsim, tc_rob_postcheck_exc.err,
 tc_rob.vrf_postcheck_tgsim, tc_rob.vrf_postcheck_exc.err]]},
 {tgrp_all, []},
 [tc_init, {group, tgrp_tcs}, tc_init_end}]}.

@spec init_per_group (GroupName, Config0) ->
 Config1 | {skip, Reason} | {skip_and_save, Reason, Config1}

where
 GroupName = atom()
 ConfigX = [tuple()]
 Reason = term()

@doc
<pre>
init_per_group is called before each test case
in the suite. It typically contains initialization
which must be done for each test case.

Note that init_per_testcase is also executed, no matter if the case
belongs to a group or not.

Return a list of configuration param/value tuples.
</pre>
@end

init_per_group (GroupName, Config) ->
 % Initialization needed for the group
t:comment("--------
--------
--------
--------
--------
--------
--------
--------"),
 Config.

@spec end_per_group (GroupName, Config0) ->
 void() | {save_config, Config1} | {fail, Reason}

where
 GroupName = atom()
 ConfigX = [tuple()]
 Reason = term()

@doc
<pre>
end_per_group is called after each test case has
finished, giving the opportunity to perform clean-up
after init_per_group.

Note that init_per_testcase is also executed, no matter if the case
belongs to a group or not.

Return ok.
</pre>
@end
end_per_group(GroupName = Config) ->
  Cleanup what was initiated in init_per_group
  ct:comment("----------- ----------- -----------"),
  ok.

#5. TEST CASES

@spec tc_init (Config) ->
ok | exit() | {skip , Reason} | {comment , Comment} |
  {save_config , Config1} | {skip_and_save , Reason , Config1}
where
ConfigX = [ tuple()]
Reason = term()
Comment = term()

@doc
<TITLE>
This is an init test case specifically for the robustness test suite. It must be executed before any of the other test cases. init_per testcase and end_per testcase will not be run for this TC.
The following is done:
- MoShell undo mode is started
- Activate the required features
- Set channel switching timers to low values in order to trigger downswitch and failed testcase due to throughput problems.
</TITLE>

TgsPid = rcc_support:get_handle(?RCC_TGS_PID , Config),
MosPid = rcc_support:get_handle(?RCC_MOS_PID , Config),

Setup RNC Configuration

Start undo mode
ueh_helper:open_tc_init_undofile(Config , ?MODULE),

Disable MoShell confirmation for bl/deb/set
ok , _ = cpp_mos:expect(MosPid ,
  "nb bldebs_set_confirmation = 0",
  ["bldebs_set_confirmation = 0"],
  [[timeout , ?RCC_MOS_TIMEOUT], line , send]),

Restoring UtranCell changes from RABest test suite since settings have been saved in CV by mistake
ok , _ = cpp_mos:expect(MosPid , "set utran_cell = Iub-8-1 tcell 3",
  ["1 MOs set"], [timeout , ?RCC_MOS_TIMEOUT], line , send)),
ok , _ = cpp_mos:expect(MosPid , "set utran_cell = Iub-8-1 uarfcnUl 9612",
  ["1 MOs set"], [timeout , ?RCC_MOS_TIMEOUT], line , send)),
ok , _ = cpp_mos:expect(MosPid , "set utran_cell = Iub-8-1 uarfcnDl 10562",
  ["1 MOs set"], [timeout , ?RCC_MOS_TIMEOUT], line , send)),
ok , _ = cpp_mos:expect(MosPid , "set utran_cell = Iub-8-2 tcell 4",
  ["1 MOs set"], [timeout , ?RCC_MOS_TIMEOUT], line , send)),
ok , _ = cpp_mos:expect(MosPid , "set utran_cell = Iub-8-2 uarfcnUl 9612",
  ["1 MOs set"], [timeout , ?RCC_MOS_TIMEOUT], line , send)),
ok , _ = cpp_mos:expect(MosPid , "set UtranCell = Iub-7-3 transmissionScheme 0",
  ["Total"], [timeout , ?RCC_MOS_TIMEOUT], line , send)),

Activate the required features

% Set features state for all features to Disabled.
{ok, _} = cpp_mos:expect(MosPid, 
  "set rncfeature features state 0",
  ["Total"], [{timeout, ?RCC_MOS_BIG_TIMEOUT}, line, send]),

Activate all RabCombination features
{ok, _} = cpp_mos:expect(MosPid, 
  "set rncfeature=rabcombination features state 1",
  ["Total"], [{timeout, ?RCC_MOS-big_TIMEOUT}, line, send]),

Activate additional licences
{ok, _} = cpp_mos:expect(MosPid, 
  "set rncfeature=HsdpaMobilityPhase features state 1",
  ["Total"], [{timeout, ?RCC_MOS-big_TIMEOUT}, line, send]),

Time in s after which a connection with low throughput is downswitched
% from HS-DSCH to CELL_FACH
{ok, _} = cpp_mos:expect(MosPid, 
  "set channelswitching hsdschInactivityTimer 3",
  ["1 MOs set"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),

Time in 0.1 s after which a CPC activated connection with low
throughput is downswitched from CELL_DCH to CELL_FACH. This parameter
is used instead of hsdschInactivityTimer for CPC activated
connections.
{ok, _} = cpp_mos:expect(MosPid, 
  "set channelswitching hsdschInactivityTimerCpc 30",
  ["1 MOs set"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),

Time in 0.1 s after which an inactive RB in a RAB combination
containing PS Interactive is released.
{ok, _} = cpp_mos:expect(MosPid, 
  "set channelswitching inactivityTimeMultiPsInteractive 30",
  ["1 MOs set"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),

Time in s after which an inactive connection in CELL_FACH state is
downswitched to URA_PCH state.
{ok, _} = cpp_mos:expect(MosPid, 
  "set channelswitching inactivityTimer 3",
  ["1 MOs set"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),

Lock HSDPA to create R99 cells
{ok, _} = cpp_mos:expect(MosPid, "bl tranancel iub=3-1,Hsdsch ",
  ["1 MOs set"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
{ok, _} = cpp_mos:expect(MosPid, "bl tranancel iub=3-6,Hsdsch ",
  ["1 MOs set"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
{ok, _} = cpp_mos:expect(MosPid, "bl tranancel iub=4-3,Hsdsch ",
  ["1 MOs set"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),

Add the below settings so that the output files match the baseline
files
{ok, _} = cpp_mos:expect(MosPid, 
  "set handover fddHoSupp 1",
  ["1 MOs set"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
{ok, _} = cpp_mos:expect(MosPid, 
  "set rncfunction hscellChangeAllowed 1",
  ["1 MOs set"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),

Activate Dch and Hspa IFLS features
{ok, _} = cpp_mos:expect(MosPid, 
  "set rncfeature=DchLoadSharing features state 1",
  ["Total"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
{ok, _} = cpp_mos:expect(MosPid, 
  "set rncfeature=HspaLoadSharing features state 1",
  ["Total"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
{ok, _} = cpp_mos:expect(MosPid, 
  "set rncfeature=HhratMobilityHsdpaEul features state 1",
  ["Total"], [{timeout, ?RCC_MOS_TIMEOUT}, line, send]),

Set handover and rncfunction parameters
{ok, _} = cpp_mos:expect(MosPid,
    "set RncFunction=1, Handover=1 ifHsHyst 20",
    ["1 MOS set "] , [{timeout, ?RCC_MOS_TIMEOUT}], line, send}),
{ok, _} = cpp_mos:expect(MosPid,
    "set RncFunction=1$ hsOnlyBestCell 0",
    ["1 MOS set "] , [{timeout, ?RCC_MOS_TIMEOUT}], line, send}),

% Remove utranrelation from Iub-3 and Iub-4 to other Iub's
ok = cpp_mos:send(MosPid,
    del UtranCell=Iub-3-[1-6], utranrelation=Intra-Iub-6),
{ok, _} = cpp_mos:expect(MosPid, "y",
    ["MOS deleted "] , [{timeout, ?RCC_MOS_TIMEOUT}], line, send}),
ok = cpp_mos:send(MosPid,
    del UtranCell=Iub-3-[1-6], utranrelation=Inter-Iub-6),
{ok, _} = cpp_mos:expect(MosPid, "y",
    ["MOS deleted "] , [{timeout, ?RCC_MOS_TIMEOUT}], line, send}),

% % Coverage relations cannot be define until TR HO40784
% % has been fixed in 3gsm.
% %
% % HS selection coverage relations
ok = cpp_mos:send(MosPid,
    "cr utranCell=Iub-4-3, coveragerelation=Iub-4-3_Iub-3-5",
    {_ok, _} = cpp_mos:expect(MosPid,
        "RncFunction=1, UtranCell=Iub-3-5",
        ["Proxy ID ="], [{timeout, ?RCC_MOS_TIMEOUT}], line, send)),
ok = cpp_mos:send(MosPid,
    "cr utranCell=Iub-3-6, coveragerelation=Iub-3-6_Iub-3-3",
    {_ok, _} = cpp_mos:expect(MosPid,
        "RncFunction=1, UtranCell=Iub-3-3",
        ["Proxy ID ="], [{timeout, ?RCC_MOS_TIMEOUT}], line, send)),

% % HS loadsharing coverage relations
ok = cpp_mos:send(MosPid,
    "cr utranCell=Iub-3-4, coveragerelation=Iub-3-4_Iub-4-1",
    {_ok, _} = cpp_mos:expect(MosPid,
        "RncFunction=1, UtranCell=Iub-4-1",
        ["Proxy ID ="], [{timeout, ?RCC_MOS_TIMEOUT}], line, send)),
ok = cpp_mos:send(MosPid,
    "cr utranCell=Iub-3-4, coveragerelation=Iub-3-4_Iub-3-2",
    {_ok, _} = cpp_mos:expect(MosPid,
        "RncFunction=1, UtranCell=Iub-3-2",
        ["Proxy ID ="], [{timeout, ?RCC_MOS_TIMEOUT}], line, send)),
ok = cpp_mos:send(MosPid,
    "set coveragerelation=Iub-3-4_Iub-4-1 relationCapability",
    "dchLoadSharing=0, hsCellSelection=0, hsLoadSharing=0",
    ["1 MOS set "] , [{timeout, ?RCC_MOS_TIMEOUT}], line, send}),
ok = cpp_mos:send(MosPid,
    "set coveragerelation=Iub-3-4_Iub-3-2 relationCapability",
    "dchLoadSharing=0, hsCellSelection=0, hsLoadSharing=0",
    ["1 MOS set "] , [{timeout, ?RCC_MOS_TIMEOUT}], line, send}),

ok = cpp_mos:send(MosPid,
    "set UtranCell=Iub-3-4 hsIfHotMarginUsers 30",
    ["1 MOS set "] , [{timeout, ?RCC_MOS_TIMEOUT}], line, send}),
ok = cpp_mos:send(MosPid,
    "set UtranCell=Iub-3-4 hsIfHotMarginUsers 1",
    ["1 MOS set "] , [{timeout, ?RCC_MOS_TIMEOUT}], line, send}),
ok = cpp_mos:send(MosPid,
    "set UtranCell=Iub-3-4 hsIfMode 2",
    ["1 MOS set "] , [{timeout, ?RCC_MOS_TIMEOUT}], line, send}),

36
{ok, _} = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−1 hsfIsMarginUsers 20",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
{ok, _} = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−1 hsfIsThreshUsers 5",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
{ok, _} = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−1 iflsMode 2",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),

%% DCH loadsharing coverage relations
ok = ccpp_mos:send(MosPid,  
  "cr utranCell=Iub−4−2,coveragerelation=Iub−4−2,Iub−4−4",  
  ["Proxy ID ="], ),
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:send(MosPid,  
  "cr utranCell=Iub−4−2,coveragerelation=Iub−4−2,Iub−4−6",  
  ["Proxy ID ="], ),
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "RncFunction=1,UtranCell=Iub−4−4",  
  ["Proxy ID ="], ),
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set coverageRelation=Iub−4−2,Iub−4−4 relationCapability "  
  "dchLoadSharing=1,hsCellSelection=0,hsLoadSharing=0",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set coverageRelation=Iub−4−2,Iub−4−6 relationCapability "  
  "dchLoadSharing=1,hsCellSelection=0,hsLoadSharing=0",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−2 dchIfsMarginCode 90",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−2 dchIfsMarginPower 90",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−2 dchIfsThreshCode 1",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−2 dchIfsPower 1",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−2 iflsMode 2",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−2 dchIfsMarginCode 60",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−2 dchIfsMarginPower 60",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−2 dchIfsThreshCode 2",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−2 dchIfsPower 2",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
ok = ccpp_mos:expect(MosPid,  
  "set UtranCell=Iub−4−4 iflsMode 2",  
  ["1 MOS set ", ]),  
{timeout, ?RCC_MOS_TIMEOUT}, line, send]),
Save undo file

ucl helper: close_tc_init_undo_file (Config, MODULE).

Setup 3GSim Configuration

Triggers = get_conf (tb, Config).
TriggersN =
  [string: join (lists: duplicate (4, lists: nth (1, Triggers)), ",")],
  [string: join (lists: duplicate (5, lists: nth (2, Triggers)), ",")],
  [string: join (lists: duplicate (7, lists: nth (3, Triggers)), ",")],

Define any specific Traffic Behaviours

Traffic Behaviour "dch_to_fach_to_ura"

3xPsIntEUL_HS

Ue 000001

\[ ok \] = cpp_mos: expect (MosPid, "[timeout, ?RCC_MOS_TIMEOUT], line, send),

\[ ok \] = cpp_mos: expect (MosPid, "set UtranCell=lab--3--6 dchIfsMarginCode 50", 
  ["1 MS set ", 
  [timeout, ?RCC_MOS_TIMEOUT], line, send],

\[ ok \] = cpp_mos: expect (MosPid, "set UtranCell=lab--3--6 dchIfsMarginPower 50", 
  ["1 MS set ", 
  [timeout, ?RCC_MOS_TIMEOUT], line, send],

\[ ok \] = cpp_mos: expect (MosPid, "set UtranCell=lab--3--6 dchIfsThreshCode 2", 
  ["1 MS set ", 
  [timeout, ?RCC_MOS_TIMEOUT], line, send],

\[ ok \] = cpp_mos: expect (MosPid, "set UtranCell=lab--3--6 dchIfsThreshPower 2", 
  ["1 MS set ", 
  [timeout, ?RCC_MOS_TIMEOUT], line, send],

\[ ok \] = cpp_mos: expect (MosPid, "set UtranCell=lab--3--6 iflsMode 2", 
  ["1 MS set ", 
  [timeout, ?RCC_MOS_TIMEOUT], line, send],

3xPsIntEUL_HS

Ue 000002

\[ ok \] = cpp_telnets: expect (TgsPid, "\#g3sim cb init64,20s psint_fach_ura_fach", "\#EXECUTED"),

\[ ok \] = cpp_telnets: expect (TgsPid, "\#g3sim mb psint_fach_ura_fach --wait_before=start 2000" 
  "--triggers "+ lists: nth (1, TriggersN)++ 
  "--rab_state int_64,64,cch_int_ura,ura,cch"
  "--ul_packet_rate 50,0,20 --dl_packet_rate 50,0,20"
  "--valid_rab_states +cch_int_ura,ura,cch --mm --,--,--" 
  "--sm activate,1,--,--", "\#EXECUTED"),

\[ ok \] = cpp_telnets: expect (TgsPid, "\#g3sim mb 3xPsIntEUL_HS --mm --,--,--,-- --triggers" 
  "--ul_lists: nth (2, TriggersN)++ --wait_before=start 2000" 
  "--rab_state int_eul_3xint_eul_hs,3xint_eul_hs," 
  "2xint_eul_hs, int_eul_hs", "\#EXECUTED"),

\[ ok \] = cpp_telnets: expect (TgsPid, "\#g3sim mb 3xPsIntEUL_HS --valid_rab_state ++,++", 
  "--sm activate,1,activate_2,activate_3", "\#EXECUTED"),

\[ ok \] = cpp_telnets: expect (TgsPid, "\#g3sim mb 3xPsIntEUL_HS --ul_rlc_sdu_size 50,70,70,50", 
  "--ul_packet_rate 50,50,50,50"
  "--dl_rlc_sdu_size 100,100,100,100"
  "--dl_packet_rate 70,50,70,50", "\#EXECUTED"),

\[ ok \] = cpp_telnets: expect (TgsPid, "\#g3sim mb 3xPsIntEUL_HS --ul_rlc_sdu_size 2,0,50,70,0,0" 
  "--ul_packet_rate 2,0,50,20,0,0"
  "--dl_rlc_sdu_size 2,0,100,100,0,0"
  "--dl_packet_rate 2,0,25,17,0,0", "\#EXECUTED"),

38
\{ok, \} = cpp_telnêt:expect(TgsPid, "3gsm mb 3xBIntEUL_HS -ul_rlc_sdu_size 3 0,0,100,80,0", [+"EXECUTED"], [timeout, ?RCC_MOS_TIMEOUT], \{prompts, unlimited\}, send)).

%% Traffic Behaviour "speech + PsInt"
%% Ue 0000003
\{ok, \} = cpp_telnêt:expect(TgsPid, "3gsm mb sp_int 64,20s sp_int", [+"EXECUTED"], [timeout, ?RCC_MOS_TIMEOUT], \{prompts, unlimited\}, send)).

%% Copy random cells mobility behavior and reduce leg change interval
%% Ue 000001
\{ok, \} = cpp_telnêt:expect(TgsPid, "3gsm cb random_cells random_cells 3s", [+"EXECUTED"], [timeout, ?RCC_MOS_TIMEOUT], \{prompts, unlimited\}, send)).

%% Copy random cells mobility behavior and reduce leg change interval
%% Ue 000002
\{ok, \} = cpp_telnêt:expect(TgsPid, "3gsm cb random_cells random_cells 4s", [+"EXECUTED"], [timeout, ?RCC_MOS_TIMEOUT], \{prompts, unlimited\}, send)).

%% Copy random cells mobility behavior and reduce leg change interval
%% Ue 000003
\{ok, \} = cpp_telnêt:expect(TgsPid, "3gsm mb random_cells random_cells 5s", [+"EXECUTED"], [timeout, ?RCC_MOS_TIMEOUT], \{prompts, unlimited\}, send)).

%% Mobility intervals
Intervals = get_conf(mb, Config),
"++ lists: nth(3, Intervals),
["EXECUTED"],
[timeout, ?RCC_MOS_TIMEOUT, {prompts, unlimited}, send]",

%% Define imsi 000001 - 000010 with default HLR subscriber data profile
rcc_tgs:define_hlr_subs(TgsPid,"000001","10").

ok | exit() | {skip, Reason} | {comment, Comment} |
{save_config, Config1} | {skip_and_save, Reason, Config1}

%% where
ConfigX = [ tuple() ]
Reason = term()
Comment = term()
@doc
<pre>
This is an init end test case for IFLS feature.
It must be executed at the end of
the test suite to restore all changes. The following is done:
- MoShell undo file is run to restore MO changes
- 3Gsim definitions are deleted.
</pre>
@end
tc_init_end(Config) ->
    rcc_tgs:cleanup_3gsim(Config),
%% Undo MO settings from tc_init command
ueh_helper:run_tc_init_undofile(Config, ?MODULE).

@spec tc_rob(Config0)
tc_rob(Config) ->

%% UEs
{UEs, SCs, .., .., Delays} = get_conf(Config),
%% Read config parameters
TgsPid = rcc_support:get_handle(?RCC_TGS_PID, Config),
TestCase = rcc_support:get_handle(?RCC_TESTCASE, Config),
RncPId = rcc_support:get_handle(?RCC_RNC_PID, Config),
ModuleBoards = rcc_support:get_boards(module, Config),
%% Setup 3GSim Configuration
%% Sleep 1s
%% Create UE  [sc = start cell, c = cell to start in,
3gsm cu=help, bn=b=den man ringer till]
{ok, _} = cpp_telnet:expect(TgsPid,
    "3gsm cu =i "++ lists: nth(1, UEs)++
    " -sc "++ lists: nth(1, SCs)++" -c s"
    " -tb paint_fach NRA_fach -mb random_cells_3s"
    " -capabilities eul -el all -ns",
    ["EXECUTED"],
    [timeout, ?CU_TIMEOUT, {prompts, unlimited}, send]),

{ok, _} = cpp_telnet:expect(TgsPid,
    "3gsm cu =i "++ lists: nth(2, UEs)++
    " -sc "++ lists: nth(2, SCs)++" -c s"
    " -tb 3xPsIntEUL_HS -mb random_cells_4s"
    " -capabilities eul -el all -ns",
    [timeout, ?CU_TIMEOUT, {prompts, unlimited}, send])
}
EXECUTED

(ok,) = ccpp_telnet:expect(TgsPid,
"3gsm cu -i 000004 -c 1015"
"-tb cs_mgmt_capabilities r99 -ns",
["ENDED"]);

 onPostExecute(TgsPid,
"3gsm stau -i all",
["ENDED"]);

(ok,) = ccpp_telnet:send(TgsPid,
"3gsm cu -i ++lists:nth(3,UES)++
"-sc ++lists:nth(3,SCs)++ -c a"
"-tb sp_int -mb random_cells 5s -capabilities eul"
"-el all -ns -bn 000004",
["ENDED"]);

Start ueidtrace on UE

lists.foreach(
  fun(Ue) ->
    rcc_th_run:start_ueidtrace(RncPid, ModuleBoards, Ue)
  end, UEs),

cpp_timer:sleep(ct:get_config(host), ct:get_config(cpp_time_server), 1),

(ok,) = ccpp_telnet:expect(TgsPid,
"3gsm stau -i all",
["ENDED"]);

Sleep 30s
cpp_timer:sleep(ct:get_config(host), ct:get_config(cpp_time_server), 30),

Stop and delete Ue
do(
  ok = ccpp_telnet:send(TgsPid,
"rm -f /c/3gsm/measurement_data/ue"++Ue++".mob")
end, UEs),

Get monitor file name
FileName = ccpp_lm:monitor_filename({tgs_monitor, TestCase, Config, rcc_support:get_log_dir(Config), ct:get_config(monitor_log_parser)}),

Purge old mobility files
lists.foreach(
  fun(Ue) ->
    ok = ccpp_telnet:send(TgsPid,
"rm -f /c/3gsm/measurement_data/ue"++Ue++".mob")
  end, UEs),

Result from parsing logfile
UeTriggers = parse_3gsm_log(FileName, Config),

Verify triggers
UeTriggersLength = plist_length(UeTriggers),
UeTriggersLengthUe = plist_length_trig(UeTriggers),
PathShort = "log_private/++
atom_to_list(prop lists:get_value(testcase, Config))++
.short.log";

case (UeTriggersLength) of
  [] -> ct:comment(iolib fwrite(
    "<font color="gold" style="font-weight: bold">NO TRAFFIC</font><br><br>"nn",
    [get_config(Config)]));
  <> -> ct:comment(iolib fwrite(
    "<p><a href="Short log"</a> Short log</p>"
nn",
    [UeTriggersLengthUe, PathShort, get_config(Config)]))
end,
GetDelay = fun(Ue) ->
    {lists:nth(lst_to_integer(Ue), Delays),"",""} end,

CreateMobFile = lists:foreach(
    fun (
        {Ue , Triggers}
    ) ->
        ueh3gsimhelper:ts_create_mobility_file(TgsPid, GetDelay(Ue) | delay2b(Triggers), "ue"++Ue++".mob")
    end , UeTriggers ) .

-tagging
tc:log(yellow_bg , "UeTriggers (p) : "p"n". [UeTriggersLength , UeTriggers]) ,

%% Wait before start
GetDelay = fun(Ue) ->
    {lists:nth(lst_to_integer(Ue), Delays),"",""} end,

%% Create mobility file
.CreateMobFile = lists:foreach(
    fun (Ue , Triggers)
    ->
        ueh3gsimhelper:ts_create_mobility_file(TgsPid, GetDelay(Ue) | delay2b(Triggers), "ue"++Ue++".mob")
    end , UeTriggers ) .

%% @spec tc_rob_vrf(Config0)
%% tc_rob_vrf(Config) ->
%%
%% UEs
% {UEs , SCs} = get_conf(Config) ,
%%
%% Read config parameters
TgsPid = rcc_support:get_handle(?RCC_TGS_PID, Config),
testCase = rcc_support:get_handle(?RCC_TESTCASE, Config),
RncPid = rcc_support:get_handle(?RCC_RNC_PID, Config),
ModuleBoards = rcc_support:get_boards(module, Config),

cpp_telnet:expect(TgsPid,
    "3gsm cu -i "++lists:nth(1,UEs)++
    " -e "++lists:nth(1,SCs)++" -aac s"
    " -tb 3xPsIntEUL_HS "++ lists:nth(1,UEs)++" .mob"
    " -capabilities eul -el all -ns",
    ["EXECUTED"],
    [timeout, ?CU_TIMEOUT, {prompts, unlimited}, send]),

cpp_telnet:expect(TgsPid,
    "3gsm cu -i "++lists:nth(2,UEs)++
    " -e "++lists:nth(2,SCs)++" -aac s"
    " -tb 3xPsIntEUL_HS "++ lists:nth(2,UEs)++" .mob"
    " -capabilities eul -el all -ns",
    ["EXECUTED"],
    [timeout, ?CU_TIMEOUT, {prompts, unlimited}, send]),

cpp_telnet:expect(TgsPid,
    "3gsm cu -i "++lists:nth(3,UEs)++
    " -e "++lists:nth(3,SCs)++" -aac s"
    " -tb sp_int -mf uc"++ lists:nth(3,UEs)++" .mob"
    " -capabilities eul -el all -ns"
    " -bn 000004",
    ["EXECUTED"],
    [timeout, ?CU_TIMEOUT, {prompts, unlimited}, send]),

%% Start ueidtrace on UE
%%
lists:foreach(
    fun(Ue) ->
        rcc_th_run:start_ueidtrace(RncPid, ModuleBoards, Ue)
    end , UEs)
end,

{ok,1} = ccpp_telnet:expect(TgsPid, "3gsm sta "++ lists:nth(1,UEs)++
    " -e "++lists:nth(1,SCs)++" -aac s"
    " -tb 3xPsIntEUL_HS "++ lists:nth(1,UEs)++" .mob"
    " -capabilities eul -el all -ns",
    ["EXECUTED"],
    [timeout, ?RCC_AOS_TIMEOUT, {prompts, unlimited}, send]),

42
Sleep 30s

cpp_timer: sleep(ct: get_config(host), ct: get_config(cpp_time_server), 30),

Stop and delete Ue
{ok, _} = cpp_telnet: expect(TgsPid,
"3gsim stou -i "++lists:nth(1,UEs)++"++lists:nth(3,UEs),
"ENDED")
[{timeout, ?STOU_TIMEOUT}, {prompts, unlimited}, send]),
rcct_gts: stop_del_ue(TgsPid, "all"),
rcct_support: check_ue_ctxt_release(Config, 10),

Get monitor file name
FileName = cpp_tm: monitor_filename([{tgs_monitor, TestCase, Config},
rcct_support: get_log_dir(Config),
ct: get_config(monitor_log_parser)}),

Result from parsing logfile
UeTriggers = parse_3g_sim_log(FileName, Config),

Verify executed triggers
UeTriggersLength = plist_length(UeTriggers),
UeTriggersLengthUe = plist_length(UeTriggers),
PathShort = "log_private/"++atom_to_list(proplists:get_value(testcase, Config))++".short.log",
case (UeTriggersLength) of
[] -> ct: comment(io_lib: fwrite(
"<font color=\"gold\">\nNO TRAFFIC</font><br>&nbsp;\n"[get_conf(Config)])];
_ -> ct: comment(io_lib: fwrite(
"<p "nca href="\"\"">Short log</p><a href="\"">\n"[UeTriggersLengthUe, PathShort, get_conf(Config)]))))
end,
ct: log(yellow_bg, "Verification of \nUeTriggers (\p): \p\n", [UeTriggersLength, UeTriggers]).

@spec tc_rob_postcheck_tgsim(Config0)

tc_rob_postcheck_tgsim(Config) ->
Expected = ",
rcct_pc: tgs_exception_and_error_check(Config, Expected).

@spec tc_rob_vrf_postcheck_tgsim(Config0)

tc_rob_vrf_postcheck_tgsim(Config) ->
Expected = ",
rcct_pc: tgs_exception_and_error_check(Config, Expected).

@spec tc_rob_postcheck_exc_err(Config0)

tc_rob_postcheck_exc_err(Config) ->
Expected = "",
rcct_pc: exception_and_error_check(Config, Expected).

@spec tc_rob_vrf_postcheck_exc_err(Config0)

tc_rob_vrf_postcheck_exc_err(Config) ->
Expected = "",
rcct_pc: exception_and_error_check(Config, Expected).

#6. TEST SUITE SPECIFIC FUNCTIONS
Parse 3gsim log file to mobility file format

parse_3gsim_log (File, Config) ->

% Read file
{ ok, Data } = file:read_file (File),

% Pick events
Pattern = "\\[(\{23\}\})\\]+?UE:\\(\\d+\\)\\)+?event=\\(\\d\\)\\)+?", 
"\(\d\\)+?\\(\d\\)+\)\(\d\\)+?\\(\d\\)+?\\(\d\\)+?\)

{ match, M } ->
Matches = M;
nomatch ->
Matches = []
end,

% Exit if no match
{ length (Matches) of
0 -> %ct: comment("<font color="gold">"NO TRAFFIC</font>") ,
ct: pal (error, "NO TRAFFIC") ;
999 -> ok
end ,

% Sort out (distinct) UEs
AllUEs = [Ue | [Ue] ] <- Matches],
UEs = lists:usort (AllUEs),

% Print log links
print_log_links (File, Config, Data),

% Verify captures
ct: log ( yellow_bg , "Matches (~p) : ~p~n" , [ length (Matches) , Matches ] ) ,

% Fix format for events
UeActions = lists:map(
fun ([A,B,C,D,E]) ->
[ A,B,C,D++"++E ] ;
[A,B,C] ->
A
end , Matches)

% Match element positions & convert to unix timestamp
UeEvents = [[U, logtime_to_unixtime (H),T,M] | [H,U,M,T] <- UeActions],

% Riformat to tuple and calculate time diff
UeTriggers = lists:map(
fun (Ue) ->
Events = [T | [H,T] <- UeEvents, H=Ue],
[Ue, time_diff_tuple (Events)]
end , UEs)

% Calculate time difference of tuple elements in list (in ms)
time_diff_tuple ( [[X1, V1, W1] | [X2, ...] ] ) = Rem) ->
[[[integer_to_list (round ((X2-X1)+1000)), V1,W1] | time_diff_tuple (Rem)]
+[[integer_to_list (100000)],V,W]]; 
time_diff_tuple ( [[ ] ] ) ->
[];

% Converts logtime to unixtime: 2011-02-01 03:48:00.748 -> 434586454234.748
logtime_to_unixtime (Logtime) ->
Pattern = "](\\d\{4\})-(\\d\{2\})-(\\d\{2\})\) 
" 
"(\\d\{2\})-(\\d\{2\})-(\\d\{3\})\\(\\d\{3\})\)

{ match, Matches } = re:run (Logtime, Pattern,
[ [capture, all but first, list] ] ),
MatchesInt = [list_to_integer (M) | M <- Matches],
[ Year, Month, Day, Hour, Min, Sec, Msec ] = MatchesInt,
calendar: datetimeto_ gregorian_seconds(
    \{(Year, Month, Day), (Hour, Min, Sec)} \} - 62167219200 + Msec/1000.

% Print log links & generate short log
print_log_links(File, Config, Data) =>
  ct:log(blue_bg, "tgs_monitor log link"n <a href=""s""/>"s</a>"n",
    ['log_private"">+filename:basename(File), File]),
Pattern = "+(Mobility Ind|target RAB state|event=[frozen]+",
  case re:run(Data, Pattern, [[capture, first, list], global]) of
    (match, Lines) =>
      DataShort = format_log_lines(Lines),
      Mat = length(Lines);
    nomatch =>
      DataShort = [],
      Mat = 0
  end,

FileShort = atom_to_list(proplists:get_value(testcase, Config))++
  "short.log",
DirShort = proplists:get_value(priv_dir, Config),
ct:log(blue_bg, "tgs_monitor SHORT (p) log link"n <a href=""s""/>"s"</a>"n",
    [Mat, "log_private="/FileShort, DirShort++FileShort]),
file:write_file(DirShort++FileShort,
    io_lib:format("˜w˜n˜p˜n" ,
      [str2int(get_conf(Config)), DataShort])).

% Reformat log lines
format_log_lines(Lines) =>
  Pattern = "\(\(\{(23)\}\).+UE: (\{6\}): (\{\}).\)
  LinesReformat = lists:map(
    fun(Line) =>
      {match, Cap} = re:run(Line, Pattern,
        [[capture, all_but_first, list]]),
      Cap
    end, Lines),

% Sort out (distinct) UEs
AllUEs = [Ue || \[\[Ue\]\] \]< LinesReformat],
UEs = lists:usort(AllUEs),

% Sort by UE & Show relative time
lists:map(
  fun(Ue) =>
    List = [[logtime_to_unixtime(T)|A] || [T|U[A]] <-
      LinesReformat, \[Ue\]<linesReformat],
    {Ue, rel_time(List)}
  end, UEs).

% Calculate time difference relative to the first list element
rel_time([[T0|R] | Rem]) =>
  [string:join(["000.000"|R], " ") | rel_time(Rem, T0)];
rel_time([]) =>
  []
rel_time([[T1|R] | Rem], T0) =>
  Tf = "000000"+integer_to_list(round(1000*(T1-T0))),
  T = string:substr(Tf, length(Tf)-3, 3++)++
    string:substr(Tf, length(Tf)-2),
  [string:join(["T"|R], " ") | rel_time(Rem, T0)];
rel_time([[], .]) =>
  []

% Add 50 ms to 2B events & subtract 50 ms from the following events
delay_2b([[A1,B1, 2D=C1] || [[A2,B2,C2] | Rem]]) =>

  A1i = integer_to_list(list_to_integer(A1)+50),
  A2i = integer_to_list(list_to_integer(A2)-50),
  [[A1i,B1,C1] || [A2i,B2,C2] | delay_2b(Rem)];
delay_2b([[T | Rem]]) =>
\[ T | \text{delay}_{2b}(\text{Rem}); \\]
\[ \text{delay}_{2b}([]) \rightarrow [] . \]

% Get length of triggers proplist
plist_length_trig(Plist) \rightarrow
lists: map(\nfun([K, V]) \rightarrow
[K, length(V)],
Plist).

% Get length of proplist
plist_length([[]]|\{V\}|\text{Rem}) \rightarrow
[length(V) | \text{plist_length}(\text{Rem})];
plist_length([]) \rightarrow
[] .

% Generate random config
random_conf() \rightarrow
UEs = ["000001","000002","000003"],
SCs = int2str(rand(1013,1024,3)), %1013–1024 %13,14,15 %14,23,24
TBs = ["4000","5000","2000"],
MBs = ["3000","4000","5000"], %3,4,5 %11,7,9
Delays = ["2250","3240","4260"],
\{UEs, SCs, TBs, MBs, Delays\}.

% Generate a list of random integers
rand(L, H, N) \rightarrow
random: seed(now()),
lists: map(\nfun(_,) \rightarrow
L+random: uniform(H-L),
end, lists: seq(1,N)).

% Converts a list of integers to strings
int2str(List) \rightarrow
lists: map(\nfun(Elem) \rightarrow
integer_to_list(Elem),
end, List).

% Converts a list of strings to integers
str2int([Elem | \_.]=Int) when is_integer(Elem) \rightarrow
list_to_integer(Int);
str2int([Elem | \text{Rem}]) when is_list(Elem) \rightarrow
\{str2int(Elem), str2int(\text{Rem})\};
str2int(Tuple) when is_tuple(Tuple) \rightarrow
str2int(\text{tuple_to_list}(Tuple));
str2int([]) \rightarrow
[] .

% Get configuration parameters
get_conf(Config) \rightarrow
proplists: get_value(conf, Config).

get_conf(Uc, Config) when is_integer(Uc) \rightarrow
M = tuple_to_list(get_conf(Config)),
MT = edoc_lib:transpose(M),
lists: nth(Uc, MT).
get_conf(Type, Config) when is_atom(Type) \rightarrow
Conf = tuple_to_list(get_conf(Config)),
Types = \{ue,1\}, \{sc,2\}, \{tb,3\}, \{mb,4\}, \{delay,5\},
TypesIndex = proplists: get_value(Type, Types),
lists: nth(TypesIndex, Conf).

% End of test suite