LTE End-to-End Reference Solution

User Guide

1222434 1.25a

LTE End-to-End Reference Solution
User Guide
1222434 1.25a

Contributors: Continuous Computing Development Team, Naveen D'cruz.

Printed in U.S.A.

Contents

Figures

Tables

1. Preface

1.1 Objective

This document provides a usage description of the LTE eNodeB and end-to-end solution designed by Continuous Computing Corporation. This document describes the procedure to setup, configure, and demonstrate signaling and data calls using the Continuous Computing ueSim and Core Network Emulator reference implementations. The demonstration shows Continuous Computing integrated LTE software suite's capabilities as well as its ability to provide end-to-end integrated solution.

1.2 Audience

The audience includes internal and external users. External users vary from customers evaluating the Continuous Computing protocol products and end-to-end solutions. Readers need familiarity of the LTE network architecture.

1.3 Document Organization

This document contains the following sections:

Table 1: Document Organization

Section	Description
1. Preface	Provides the objective and release details.
2. Introduction	Provides an overview of the product, including the product description and features.
3. Architecture	Describes the network architecture and protocol stacks.
4. End-to-End Call Flow (Message Sequence Chart)	Provides the call flow sequence charts of the different scenarios.
5. Setup (End-to-End Solution)	Describes the setup information for this software.
6. Building Integrated Stacks	Describes the procedure to build.
7. Configuration of ueSim, eNodeB and CNE	Describes the configurations needed for ueSim, eNodeB, and CNE
8. Execution of the Setup	Describes the execution process of the demonstration for this software.
9. Troubleshooting	Describes the troubleshooting information for this software.
10. References	Lists the reference documents

1.4 Release History

The following table lists the history of changes in successive revisions to this document.

Table 2: Release History

Version	Date	Author (s)	Description
1.25a	October 20, 2011	Naveen Dcruz H	Addendum release for Radisys logo and template upgrade.
1.24a	August 30, 2011	Rajakumara DT	Updated for eNodeB GPR 4.
1.23a	April 29, 2011	Narayana moorthi	Updated document for eNodeB compilation flags and trace flags.
1.22a	January 20, 2011	Chandra Shekar	Added the features of Mobile Terminating call, Cell Selection/Reselection, Dedicated Bearer, and Timing Advance.
1.21a	October 18, 2010	Ramana Polamarasetti	Added usage of Interphase/MindSpeed boards for eNodeB.
1.2	August 04, 2010	Ashish Dobhal/ Venu Madhav	Updated document for segregated CNEs and added Product Feature Flags for MME/S-GW/PDN-GW Reference Applications.
1.1	February 03, 2010	Pankaj Kumar	Initial Release.

2. Introduction

2.1 Product Description

LTE eNodeB by Continuous Computing comprises of control and data plane protocol suites towards UE and Core Network. LTE eNodeB consists of sample applications with functionalities of Radio Resource Management, data application to exchange data from PDCP to eGTP-u and relay functionality for the translation between S1AP and RRC messages.

The sample end-to-end solution has the capabilities for establishing end-to-end data transfer between two end-points – client and server.

Continuous Computing has also developed a sample UE simulator having Uu interface protocol stacks as functions to communicate with eNodeB.

Continuous Computing has developed the Evolved Packet Core Network Nodes (CNE), including support for the S-GW, P-GW and MME network elements.

Following end point applications are supported in end-to-end solution comprising of ueSim, eNodeB, and CNE.

1. PING application
2. Video streaming through VLC player
3. Data transfer through iperf utility
4. FTP transfer
5. HTML web-browsing
6. VNC server application

2.2 Definitions and Acronyms

Table 3: Acronyms

Acronym	Description
App	Sample Application Layer
ATCA	Advanced Telecom Computing Architecture
CN	Core Network
CNE	Core Network Emulator
eNB/eNodeB	E-UTRAN Node B
e-GTP	Evolved GTP
E-UTRAN	Evolved UTRAN
GTP	GPRS Tunneling Protocol
HSS	Home Subscriber Server
MAC	Medium Access Control Protocol
MME	Mobile Management Entity
NAS	Non-access Stratum
PAL	Physical Abstraction Layer
PDCP	Packet Data Convergence Protocol

P-GW/PDN-GW/PGW	PDN Gateway
RLC	Radio Link Control Protocol
RRC	Radio Resource Control Protocol
RRM	Radio Resource Management
S1AP	S1 Application Protocol
SCTP	Stream Control Transmission Protocol
S-GW/SGW	Serving Gateway
SM	Stack Manager
LTE	Long Term Evolution
TTI	Transmission Timing Interval
TUCL	TCP/UDP Convergence Layer
UE	User Equipment
ueSim/UE Sim	UE Simulator
UTRAN	Universal Terrestrial Radio Access Network
VC	Video Client
VS	Video Server

3. Architecture

3.1 Evolved Packet System

Figure 1 shows the network architecture of Evolved Packer System as per 3GPP.

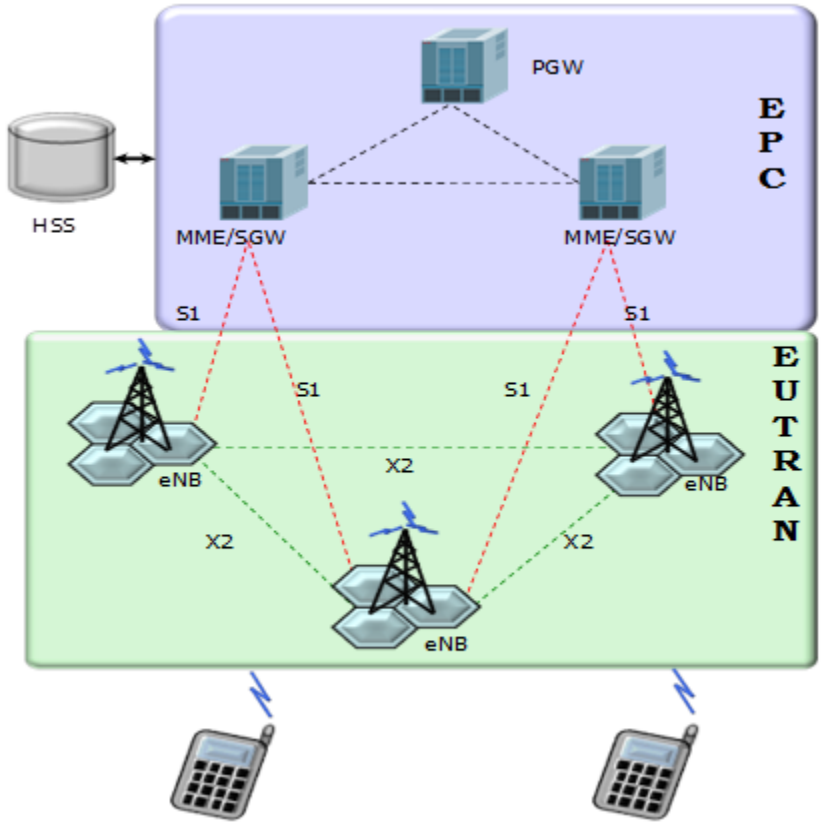


Figure 1: Evolved Packet System

3.1.1. The Evolved UTRAN

The E-UTRAN consists of eNodeBs (eNB) which provide E-UTRA user plane (PDCP/RLC/MAC/PHY) and control plane (RRC) protocol terminations toward the user equipment (UE). Green arrows display the implemented interfaces in current CCPU end-to-end solution.

Major eNodeB functionalities comprise of:

- Radio Resource Management (RRM).
- Typical base station functionalities – modulation, demodulation, coding, scheduling and so on.
- Selection of an MME at UE attachment and Routing of User Plane data towards the SGW.
- Scheduling and transmission of broadcast information (originated from the MME or O and M).
- Measurement and measurement reporting configuration for mobility and scheduling.
- Provides initial access to the network, registration, and attach/detach to/from the network.
- Location Management (tracking area update and so on).
- Handover Management – Intra-eNodeB, Inter-eNodeB, Inter-eNodeB – with change of MME, Inter-eNodeB – with same MME but different SGW, and Inter-RAT handovers.

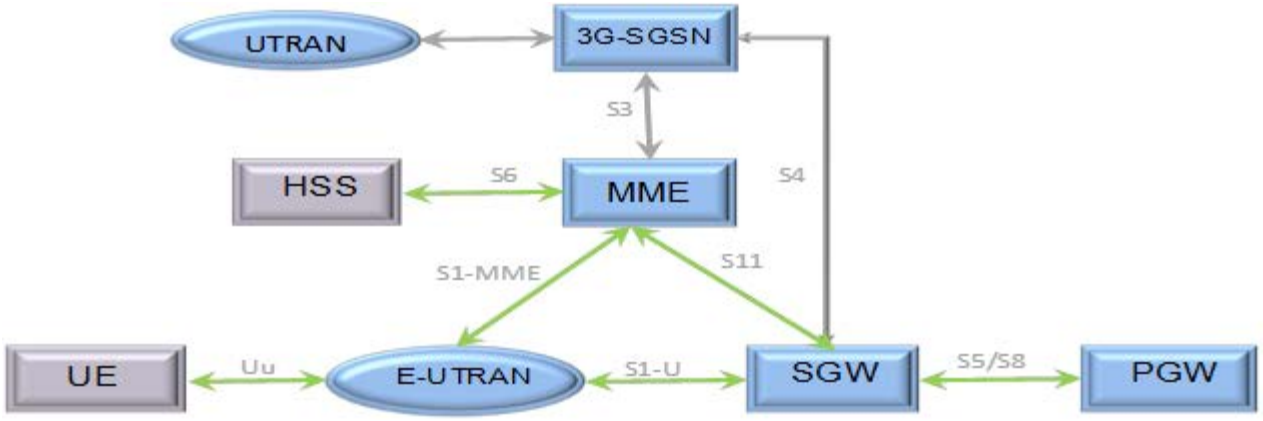


Figure 2: Evolved Packet System Interfaces

3.2 End-to-End Protocol Stack Architecture

Figure 3 shows the end-to-end protocol suite in CCPU end-to-end solution comprising of sample UE Sim, eNodeB, and Core-Network-Emulator (CNE) consisting of MME, S-GW, and PDN-GW nodes.

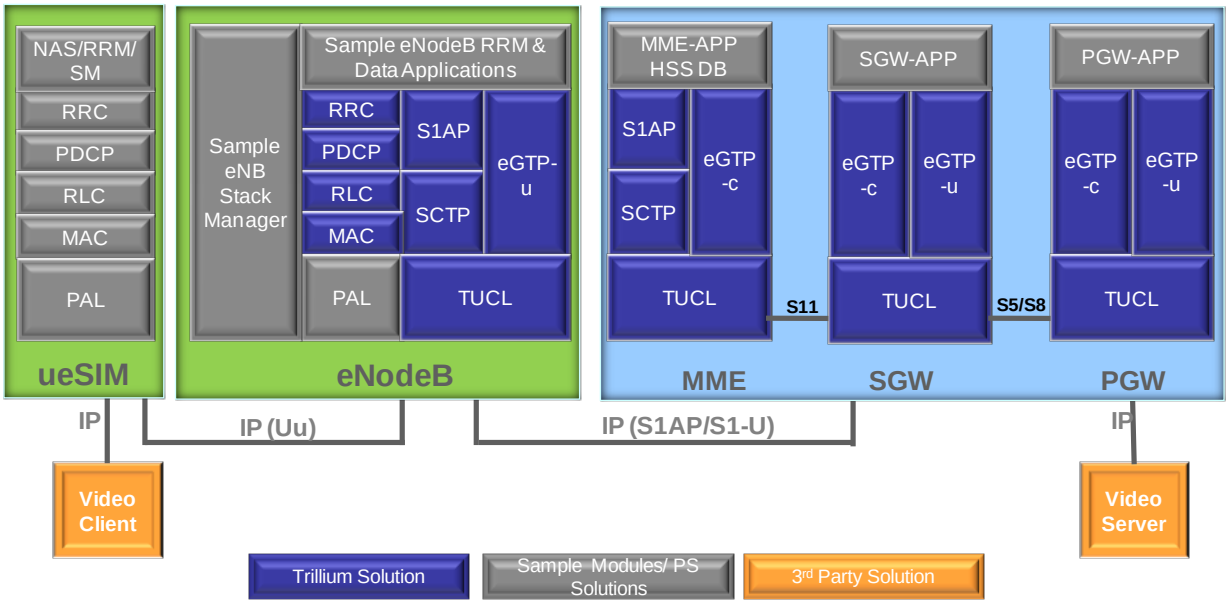


Figure 3: End-to-end Protocol Stack Architecture

Sample UE simulator (ueSim) has scalable architecture for the support of multiple UEs. Current end-to-end demo set-up has only one UE. ueSim has sample Non-Access-Stratum (NAS) and Radio-Resource-Management (RRM) functionalities for UE. NAS consists of encoding/decoding of NAS messages consisting of EMM and ESM procedural messages supporting which support UE associated procedures like attach and detach procedures.

Sample CNE applications emulate the sample data application, sample MME, S-GW, and P-GW functionalities.

Sample eNodeB application includes eNodeB data application to exchange data from PDCP to eGTP-u and vice versa, sample RRM and relay functionality for the translation between S1AP and RRC messages [Relay]. eNodeB is executed over any flavor of Linux or simple-executive (a thin OS) of Octeon processor from Cavium. Running eNodeB over Octeon is essential to achieve the LTE stringent requirement of 1ms TTI. On normal Linux, the TTI is calibrated to 5-10 ms as 1ms is not feasible.

Trillium SCTP integrated stack runs over a raw IP stack provided by platform (Linux). The services of IP stack is accessed through a convergence layer (TUCL) developed by Trillium.

Sample stack managers configure the protocol layers at eNodeB and CNEs.

eNodeB and CNEs communication is over an IP stack provided by platform (Linux). PDN-GW and Video Server, ueSim and Video Client also use platform IP stack. When eNodeB is running in Linux the communication between ueSim and eNodeB is over IP. For Cavium's Octeon execution, the communication between multiple cores executing ueSim and eNodeB is through work-queue in simple executive mode.

3.3 LTE eNodeB Architecture on Picochip Platform

Figure 4 depicts the architecture of CCPU LTE eNodeB which is executed in both pure Linux platform or over Octeon processor running simple-executive (a thin OS) and Linux user space.

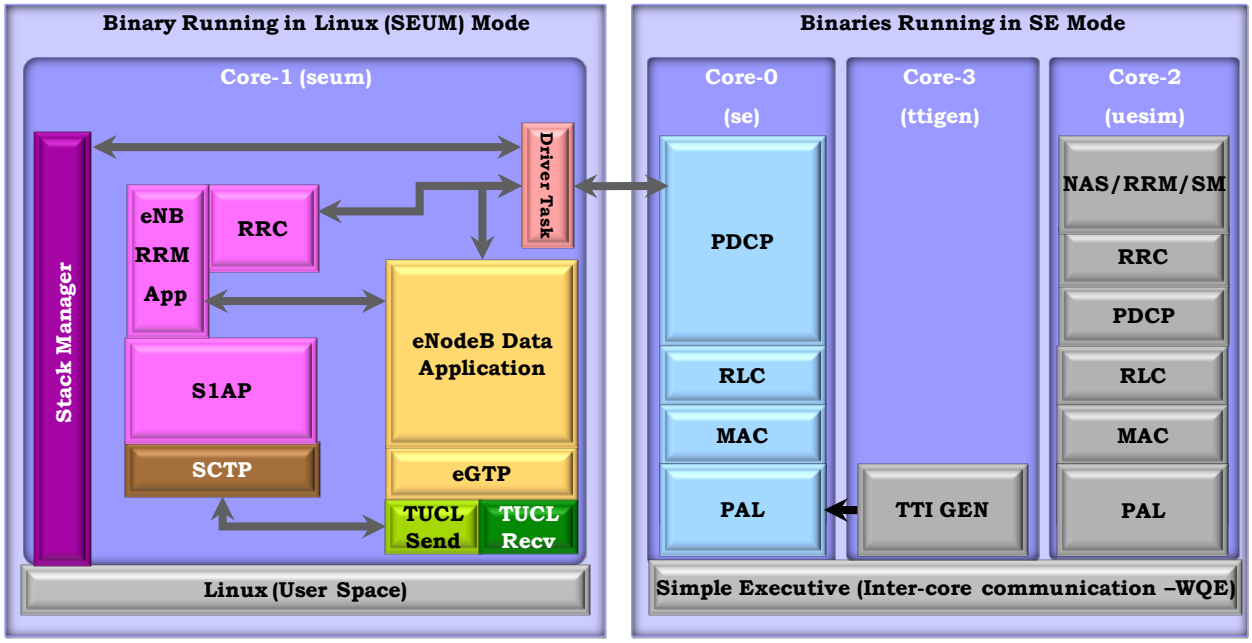


Figure 4: LTE eNodeB Architecture

- Total of two eNodeB binaries:
  - Binary Running in SE mode
    - PDCP, RLC, MAC, PAL (Physical Abstraction Layer)
  - Binary Running in Linux (SE-UM mode)
    - eNB App, S1AP, RRC, Data App, eGTP, SCTP, TUCL
- Total of one ueSim binary:
  - Binary Running in SE mode
    - ueSim App, RRC, PDCP, RLC, MAC, PAL (Physical Abstraction Layer)
- Total of four or five cores:
  - Core-1: eNB App, RRC, S1AP, SCTP, and SM
  - Core-2: Data App, eGTP, TUCL, and Driver Task
  - Core-3: PDCP, RLC, MAC, PAL
  - Core-4: TTI Generation Logic Code
  - Core-5: UE-Sim

- Total of nine threads:
  - PDCP, RLC, MAC, PAL are in one thread
  - eNB app, RRC, S1AP are in second thread
  - SCTP is third thread
  - Data App and eGTP
  - TUCL Send
  - TUCL Receive
  - Driver Task
  - Stack Manager
- Communication between binary 1 and binary 2
  - Driver Task in Linux mode listens to workQueue to receive messages from SE binary threads.

3.4 LTE eNodeB Architecture on Mindspeed Platform

Figure 5 depicts the architecture of CCPU LTE eNodeB implementation for mindspeed platform split across Upper ARM and Lower ARM running linux and 4GMX as operating system.

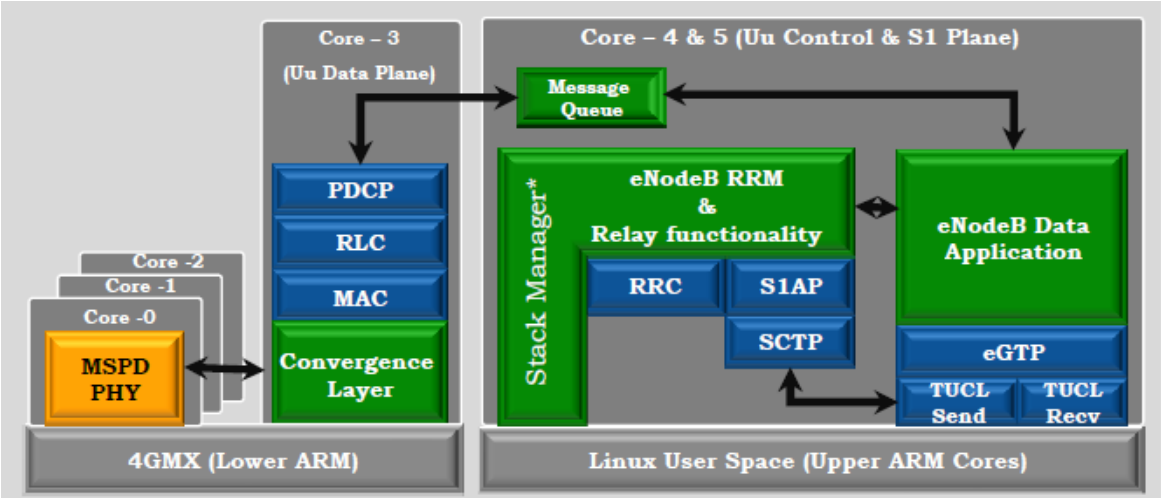


Figure 5: LTE eNodeB Architecture Mindspeed Platform

- Total of two eNodeB binaries:
  - Binary Running in Lower ARM
    - PDCP, RLC, MAC, PAL (Physical Abstraction Layer)
    - Physical layer implementation.
  - Binary Running in Upper ARM
    - eNB App, S1AP, RRC, Data App, eGTP, SCTP, TUCL
- Total four or five cores:
  - Core-4 and 5 on Upper ARM: eNB App, RRC, S1AP, SCTP, and SM, Data App, eGTP, TUCL, and Driver Task
  - Core-3 on Lower ARM: PDCP, RLC, MAC, PAL
  - Core-0 to 2 on Lower ARM: Physical layer implementation
- Communication between binary 1 and binary 2
  - Shared memory is used for communication between upper ARM and lower ARMS

4. Setup (End-to-End Solution)

This section describes the hardware and software components and setup required for executing the CNEs (binaries) in end-to-end environment to demonstrate the functions and interactions of CNEs with rest of nodes like UE Sim, eNodeB, external client and server. Currently the demonstration performed is the video streaming from video-server to video-client machines.

eNodeB and ueSim (Uu stack) is executed on Linux or Cavium's Octeon simple executive (a thin OS).

Note: MME, S-GW **and** P-GW together form core-network emulators (CNE). These applications are sample applications and **must** be treated as reference applications with limited set of functionality**.**

4.1 End-to-End Demo with Uu Stacks on Linux

Figure 6 shows the test setup of end-to-end demo running LTE eNodeB, CNEs, and ueSim on Linux.

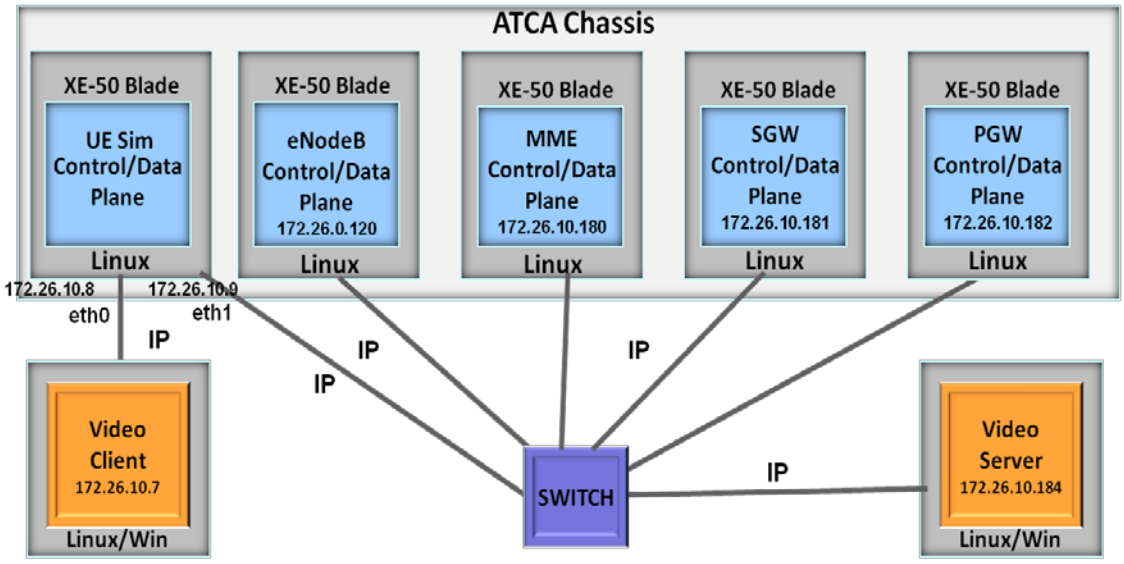


Figure 6: End-to-End Demo with Uu Stacks on Linux

4.1.1. Hardware

The hardware requirements are as follows:

- One Linux machine (desktop/server) with an Ethernet card to run CNE.
  - With virtual IP configured on the same Linux machine (desktop/server) having an Ethernet card is used to run MME, S-GW, and PDN-GW. However, based on hardware available these nodes are run on different Linux machines as well.
- One Linux machine (desktop/server) with an Ethernet card to run eNodeB.
- One Linux machine (desktop/server) with two Ethernet cards to run UE simulator.
- One machine (laptop/desktop) to run the video client. This is windows or Linux machine.
  - It is connected directly to one of the interfaces of UE simulator.
  - It must not have any other connectivity other than the direct connection to UE simulator.
- One machine (laptop/desktop) to run the video server. This is windows or Linux machine.
- One 8 ports gig L2 switch and 5-6 Ethernet cables.

4.1.2. Software

The software requirements are as follow:

- Both PDN-GW and UE simulator require libpcap and libpcap-devel packages v0.9.1 or later.
- CNEs, eNodeB and UE simulator run on Linux operating system with kernel version 2.6.
- The video client and video server both are VLC player version 0.9 or higher.
- The laptop running video client is assigned a static IP address and have the IP address of UE simulator as its default gateway.
- In the UE simulator a route rule is be added for the video client machine with its own IP address as gateway and the interface connecting them as the device.

  For example, ***route add –host <ue_client_ip_addr> gw <ue_sim_ip_addr_connected_to_ue_client> dev < device_connected_to_ue_client>***

Note: For better data throughout the performance, the CNE, eNodeB, and ueSim are executed on Continuous Computing XE50 blades - Quad Core Intel® Xeon® Processor L5408 (12M Cache, 2.13 GHz, 1066 MHz FSB, 12.46 GB RAM) With Dual processor Configuration in ATCA chassis**.**

4.2 End-to-End Demo with Uu Stacks on MindSpeed

Figure 7 shows the test setup of end-to-end demo running LTE eNodeB on MindSpeed EVM board, ueSim, and CNEs on Linux.

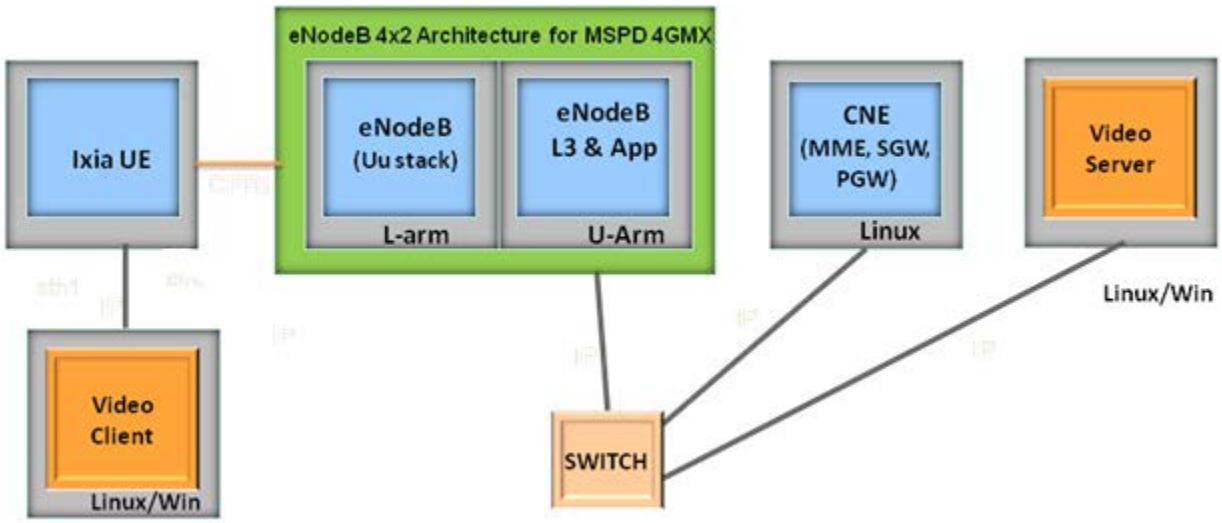


Figure 7: End-to-End Demo with Uu Stacks on MindSpeed Board

4.2.1 Hardware

The hardware requirements are as follows:

- One Linux machine (desktop/server) with an Ethernet card to run CNE.
  - With virtual IP configured on the same Linux machine (desktop/server) having an Ethernet card is used to run MME, S-GW, and PDN-GW. However, based on hardware available these nodes are run on different Linux machines as well.

- EVM from MindSpeed to run eNodeB.
- One Linux machine (desktop/server) with two Ethernet cards to run UE simulator.
- One machine (laptop/desktop) to run the video client. This is windows or Linux machine.
  - It is connected directly to one of the interfaces of UE simulator.
  - It must not have any other connectivity other than the direct connection to UE simulator.
- One machine (laptop/desktop) to run the video server. This is windows or Linux machine.
- One 8 ports gig L2 switch and 5-6 Ethernet cables.

4.3 End-to-End Demo with Uu Stacks on Cavium

Figure 8 shows the test setup of end to end demo running LTE eNodeB, ueSim on Cavium and CNEs on Linux.

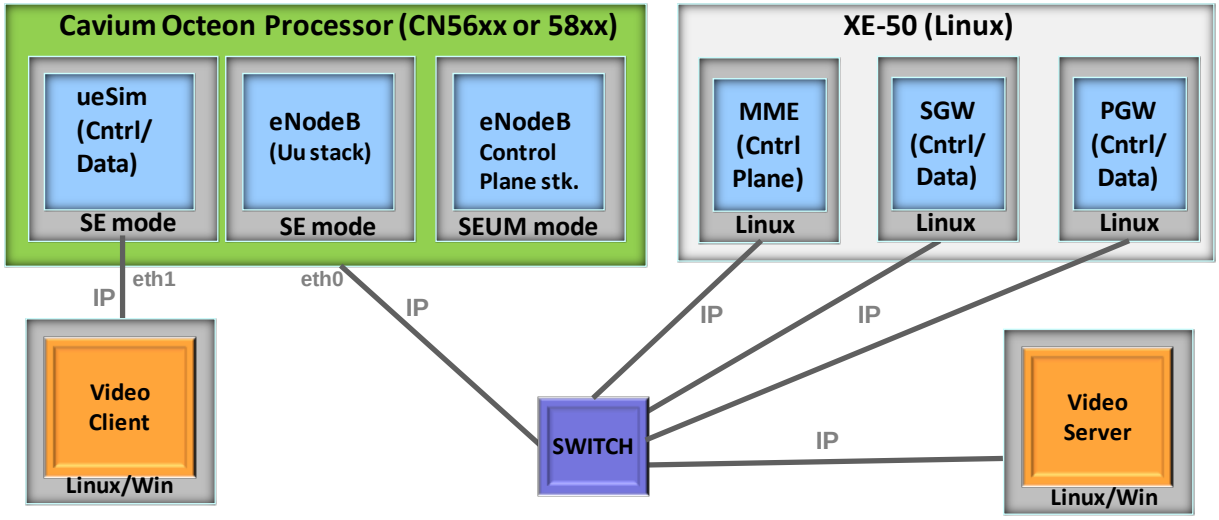


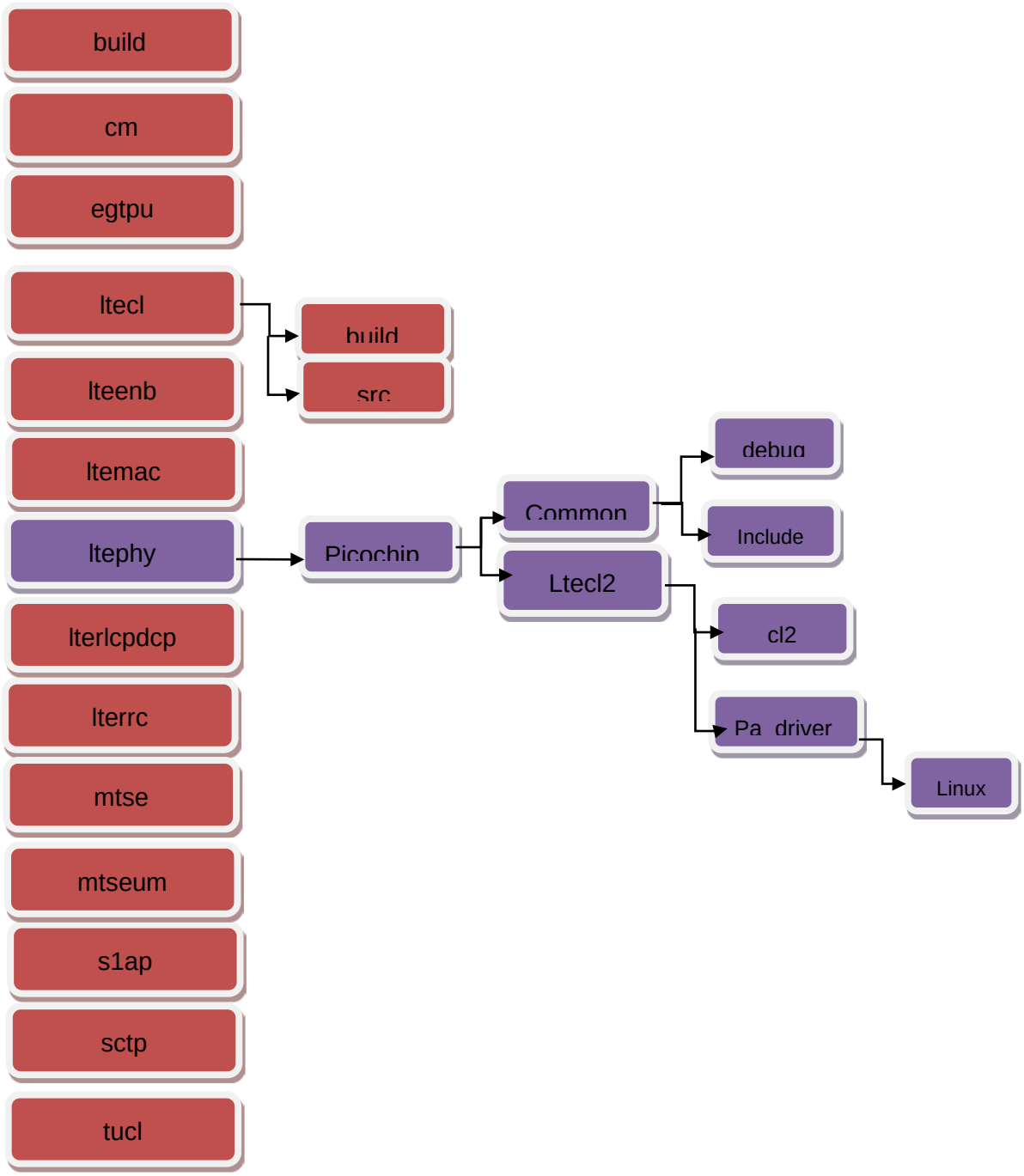
Figure 8: End-to-End Demo with Uu Stacks on Cavium

- eNode and ueSim are executed on Cavium OCTEON Multi-Core Processor board in simple executive (a thin OS).
- Sample CNEs are executed on Continuous Computing XE-50 blade in ATCA chassis in Linux.
- Video client and server are executed on Linux or windows machines as above.
- Cavium blade, XE50 blade, and Video Server are connected to private Ethernet switch.
- Machine (laptop/desktop) running Video Client is directly connected to Cavium board.
- ueSim and eNodeB run in Simple Executive (SE) mode on Cavium board.

5. eNodeB Directory Structure

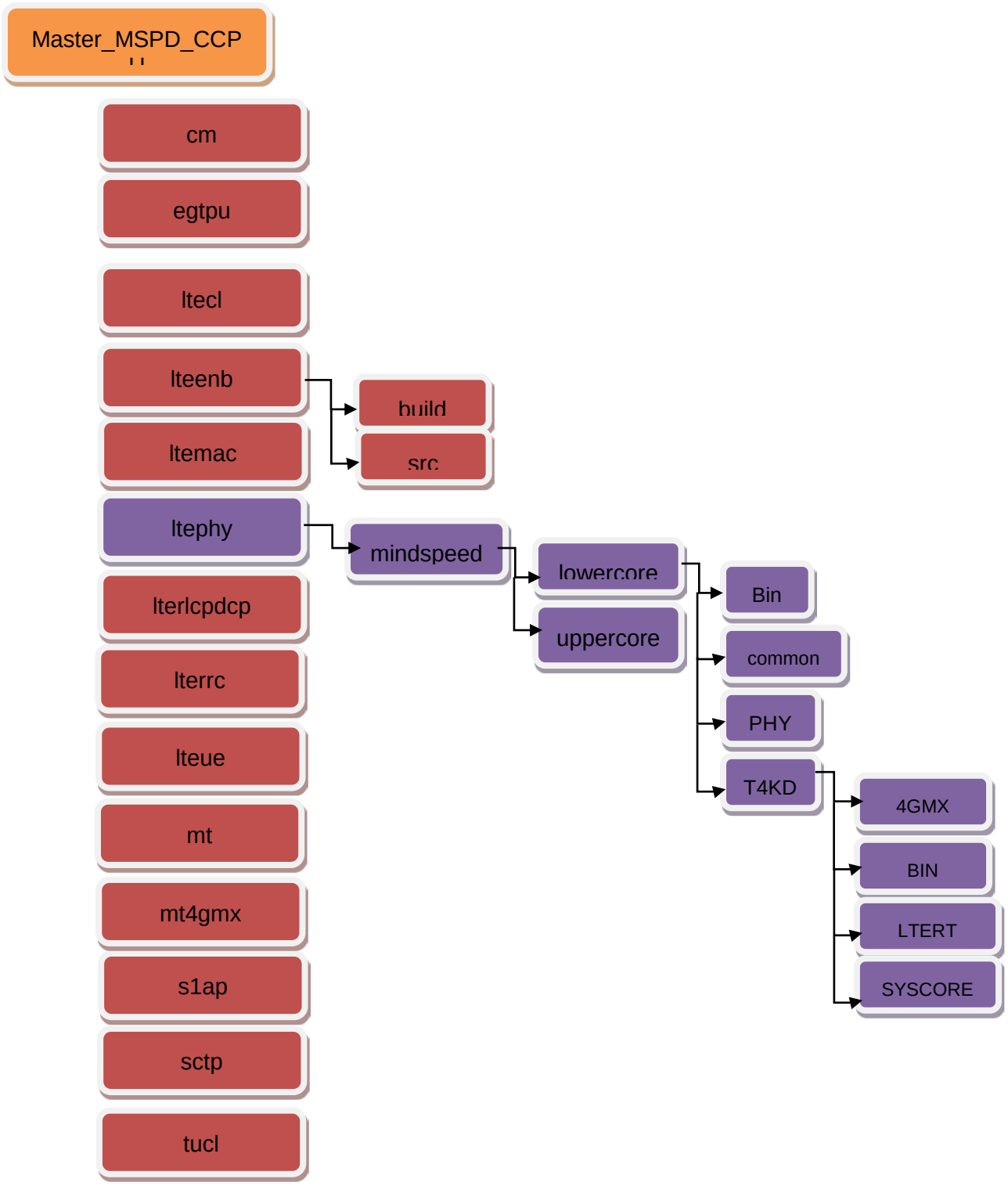
This section describes the directory structure for the eNodeB code base.

5.1 PicoChip Directory Structure



Note : The blocks in ***red*** color are CCPU specific directory the ones in ***Purple*** are Picochip specific.

5.2 Mindspeed Directory Structure



Note : The blocks in ***orange*** and ***red*** color are CCPU specific directory the ones in ***Purple*** are Picochip specific.

6. Building Integrated Stacks

Each of the entities/nodes is executable in a unique directory.

Prerequisites:

Following set of library files is to be present:

1. Set of integrated Trillium dependent protocol layers library files for respective entity.
2. MT/Cavium SSI library file
3. Common library file

User needs to build required entity/node individually.

6.1 Build Core Network Emulators (CNE)

The CNE binaries for MME, S-GW, and P-GW simulation are built on any Linux machine. For more details refer to *LTE CNE User Guide*.

6.1.1 MME

6.1.1.1 Product Feature Flags

LTE-MME Reference Application supports certain features that are conditionally compiled. Following are the list of compile time flags that are enabled to invoke the corresponding features:

Flag	Description
VB	MME Main Flag.
VB_DEBUG	Debug Flag. Enable to compile the code for debug printing in LTE-MME Reference Application.
LCLVB	Loose Couple Layer Management Flag.
LCLVBMILVB	To enable Loose Coupling from Protocol Stack towards Stack manager.
LCSMMILVB	To enable Loose Coupling from Stack manager towards Protocol Stack.
VB_USTA	Alarm Flag. Enable to compile the code for generating unsolicited status in LTE-MME Reference Application.
VB_MME	MME Flag. Enable to compile MME support.
LCSMVBMILVB	To enable Loose Coupling from Stack manager towards Protocol Stack.
VB_PERF_MEAS	Performance Measurement Flag. Enable to compile the code to measure performance inLTE-MME Reference Application.
VB_MME_AUTH	Authentication flag. Enable to compile Authentication Support in LTE-MME Reference Application.
VB_MME_NW_INIT_DETACH	To enable Network initiated Detach Procedure
EG_REL_930	To enable release 9 upgrade for eGTP
EG_PHASE2	To enable release 9 upgrade for eGTP

6.1.1.2 Build MME

The make file for building MME is **vb.mak**. The binary is built by the following command:

- ***make –f vb.mak BLDENV=lnx_acc acc***

This builds the sample stack manager and sample application of MME and then links with the required integrated stack libraries (TUCL, SCTP, S1AP and EGTP). It creates the executable **vb_acc** in the build directory.

Note: To compile the libraries, delete the dependent products libraries **libvb_xxxyyeg.a** and **libvb_<xxxyysz>.a** --- where;

xxx is **lnx** for **Linux** or **sol** for **Solaris**

yy for **32** bit or **64** bit

6.1.2 S-GW

6.1.2.1 Product Feature Flags

LTE S-GW Reference Application supports certain features that are conditionally compiled. Following are the list of compile time flags that are enabled to invoke the corresponding features:

Flag	Description
QO	SGW Main Flag.
QO_DEBUG	Debug Flag. Enable to compile the code for debug printing in LTE-SGW Reference Application.
LCLQO	Loose Couple Layer Management Flag.
LCQOMILQO	To enable Loose Coupling from Protocol Stack towards Stack manager.
LCSMMILQO	To enable Loose Coupling from Stack manager towards Protocol Stack.
QO_USTA	Alarm Flag. Enable to compile the code for generating unsolicited status in LTE-SGW Reference Application.
QO_SGW	SGW Flag. Enable to compile SGW support.
LCSMQOMILQO	To enable Loose Coupling from Stack manager towards Protocol Stack.
EG_REL_930	To enable release 9 upgrade for eGTP
EG_PHASE2	To enable release 9 upgrade for eGTP

6.1.2.2 Build S-GW

The make file for building SGW is **qo.mak**. The binary is built by the following command:

- ***make –f qo.mak BLDENV=lnx_acc acc***

This builds the sample stack manager and sample application of S-GW and then links with the required integrated stack libraries (TUCL and eGTP). It creates the executable **qo_acc** in the build directory.

Note: To compile the libraries, delete the dependent products library **libvb_<xxxyy>_eg.a** --- where; **xxx** is **lnx** for **Linux** or **sol** for **Solaris** and **yy** for **32** bit or **64** bit.

6.1.3 P-GW

6.1.3.1 Product Feature Flags

LTE P-GW Reference Application supports certain features that are conditionally compiled.
Following are the list of compile time flags that are enabled to invoke the corresponding features:

Flag	Description
AV	PGW Main Flag.
LCLAV	Loose Couple Layer Management Flag.
LCAVMILAV	To enable Loose Coupling from Protocol Stack towards Stack manager.
LCSMAVMILAV	To enable Loose Coupling from Stack manager towards Protocol Stack.
AV_USTA	Alarm Flag. Enable to compile the code for generating unsolicited status in LTE-PGW Reference Application.
AV_PERF	Performance Measurement Flag. Enable to compile the code to measure performance in LTE-PGW Reference Application.
EG_REL_930	To enable release 9 upgrade for eGTP
EG_PHASE2	To enable release 9 upgrade for eGTP

6.1.3.2 Build P-GW

The make file for building P-GW is **av.mak**. PDN-GW makes use of the Libpcap APIsfor this the library (**libpcap.a**) must be present in the build directory. The relevant/associated header files of this library of PCAP must be present in **/usr/include/.** The binary is built by the following command:

- ***make –f av.mak BLDENV=lnx_acc acc***

This builds the sample stack manager and sample application of PDN-GW and then links with the required integrated stack libraries (TUCL and EGTP). It creates the executable **av_acc** in the build directory.

Note: To compile the libraries, delete the dependent products library **libvb_<xxxyy>_eg.a** --- where;

xxx is **lnx** for **Linux** or **sol** for **Solaris**

and **yy** for **32** bit or **64** bit.

6.2 Build ueSim

6.2.1 Product Feature Flags

LTE ueSim supports features that are conditionally compiled. Following are the list of compile-time flags that are enabled to invoke the corresponding features.

Flag	Description
UESIM_TRIGGER_DRB_EST	Enable the Dedicated Bearer initiation from UE Simulator.
TA_NEW	Enable the Timing Advance Feature.
SI_NEW	Enable the SIBS for Cell Selection/Reselection.
TFU_TDD	Enable the TDD mode. If disabled, it is FDD mode.
LTE_TDD	Enable the TDD mode. If disabled, it is FDD mode.
UE_RAD_CAP	Enable the UE capability support..
UE_SIM_ENABLE_PCAP	Enable capture/send of IP packets from/to UE client must be enabled by default.
ENB_PERF_UL_DATA	Enable pumping of UL data packets from UeSim for performance testing of UL Data. This flag must be enabled when **UE_SIM_ENABLE_PCAP** is disabled.
UE_SUPPORT_RLC_UM_MODE	Enable the UE support for RLC UM Mode.

6.2.2.1 Build ueSim for Linux

The ueSim binary for Linux is built on any Linux machine.

- Go to **ueSim** compilation directory.
- Issue the command
  - **make -f ue_stack.mak BLDENV=lnx_e2e_acc acc**
- The above build command generates **ueSim** binary.

6.2.2.2 Build ueSim for Cavium

The ueSim binary for Cavium has built using cross compiler on Linux machine installed with Cavium SDK. This binary is executed in simple executive (thin OS) mode.

- Go to **ueSim** compilation directory.
- Issue the command
  - **make -f ue_stack.mak BLDENV=cav_e2e_acc acc**
- The above build command generates **ueSim** binary.
- Copy **uesim** binary to **/tftpboot/**
  - **cp ueSim /tftpboot/**

6.3 Build eNodeB

6.3.2 Product Feature Flags

LTE eNodeB Reference Application supports features that are conditionally compiled. Following are the list of compile-time flags that are enabled to invoke the appropriate features.

Flag	Description
TA_NEW	Enable the Timing Advance Feature. When the timing alignment feature is enabled, the Timing Advance command is sent to UE from the eNodeB.
SI_NEW	Enable SIBS for Cell Selection/Reselection.
RGR_SI_SCH	Enabled along with SI_NEW for SIBs of Cell Selection/Reselection. When the flags are enabled the SIB3, SIB4 and SIB5 used for Cell Selection / Re-selection by the UE are sent.
LTE_TDD	Enable the TDD mode. If disabled, it is FDD mode.
VE_RELAY	Enable the functionality of relaying of RRC to S1AP and S1AP to RRC. This flag needs to be enabled always.
EU_DAT_APP	Enable the functionality of relaying of PDCP to eGTP-U and eGTP-U to PDCP. This flag needs to be enabled always.
EGTP_U	Enable eGTP-U functionality.
KW_PDCP	Conditionally compiles code to allow stack to support the LTE PDCP protocol.
KW	Conditionally compiles code to allow stack to support the LTE RLC protocol.
DEBUGP	Enable the debug prints on the screen.
LCEGUIEGT	Enable loosely coupled EGT interface.
LCVELICTF	Enable loosely coupled CTF interface.
LCVELINHU	Enable loosely coupled MAC interface.
LCPJUIPJU	Enable loosely coupled interface PJ to NH.
LCVELIRGR	Enable loosely coupled RGR interface.
HI_MULTI_THREADED	Enable this flag to run TUCL in multi-threaded mode. The S_SINGLE_THREADED flag must not be defined.
SS_M_PROTO_REGION	Enable to provide the multi region support from SSI.
LTE_ENB_PERF	Enable for Uu performance setup without core network contact.
VE_PERF_MEAS	Enabled for measuring the Uu performance.
RGR_RRM_TICK	Enable the LTE MAC layer to provide the TTI ticks to RRM (that is, eNodeB application.
VE_SRB2_SUPPORT	Enable SRB2 support.
VE_SUPPORT_RLC_UM_MODE	Enable to support RLC functioning in UM mode.
LTE_LNX_AFFINITY	Enable to set a particular affinity [CPU/processor] for different system threads on Linux.

Flag	Description
VE_PERF_DL_DATA	Enable DL data for performance test.
LTE_ENB_PAL	This flag is enabled when Physical abstraction layer is used (that is, IP communication).
TRC0	Enable Multiprocessor Operating System support functions. Trace macro takes the function name as a parameter. If TRC0 is disabled, the trace is mapped to a null statement. Otherwise, the macro is mapped to print the function name.
TRC1	Enable Multiprocessor Operating System interface functions. Trace macro takes the function name as a parameter. If TRC1 is disabled, the trace is mapped to a null statement. Otherwise, the macro is mapped to print the function name.
TRC2	Enable Module support functions. Trace macro takes the function name as a parameter. If TRC2 is disabled, the trace is mapped to a null statement. Otherwise, the macro is mapped to print the function name.
TRC3	Enable Module interface functions. Trace macro takes the function name as a parameter. If TRC3 is disabled, the trace is mapped to a null statement. Otherwise, the macro is mapped to print the function name.
TRC4	The TRC4 trace flag is used to enable combination of MOS and module functions. It means to enable TRC0, TRC1, TRC2, TRC3 and TRC4. Trace macro takes the function name as a parameter. If TRC4 is disabled, the trace is mapped to a null statement. Otherwise, the macro is mapped to print the function name
TRC5	Enable all levels of trace macros.
VE_PICO	Enable the Pico chip reference application requirements
YS_PICO	Enable the Pico chip convergence layer requirements
VE_WITHOUT_CNE	Enable along with VE_PERF_MEAS for performance measurement without CNE
VE_SM_LOG_TO_FILE	Enable for initializing log file information's
WIRESHARK_LOG	Enable for Wireshark logging
VE_LNX_PCAP_PERF	Enable the libcap functionality
VE_PERF_DL_DATA	Enable the performance for DL Data
UE_RAD_CAP	Enable the UE capability support
MSPD	Specific configuration and setting related to Mindspeed
LTEMAC_MIMO	Enable the MIMO support.
VE_UM_MODE	Enable unidirectional UM mode DRB at RLC layer
VE_UM_MODE_BI_DIR	Enable bidirectional UM mode DRB at RLC layer
SS_4GMX_LCORE	Enable 4GMX platform specific functionalities.
VE_DL_CQI	DL Periodic CQI reporting is disabled.
LTEMAC_DRX	Enable the DRX support.
VE_SIBS_ENBLD	Enable SIB1 and SIB2 scheduling. This flag should be enabled always.

Flag	Description
LSZV1	Release 9 up gradation for S1AP
SS_TICKS_SEC	This indicates the tick resolution for timers. if **SS_TICKS_SEC=10** it means resolution is 100 ms If **SS_TICKS_SEC=100** it means resolution is 10 ms (Default Configuration). if **SS_TICKS_SEC=1000** it means resolution is 1 ms Basically **(1/SS_TICKS_SEC) * 1000** gives the resolution unit it terms of ms.
DISABLE_RRCSEC In 'CMENBE2EOPTS' within lteenb/build/enodeb_uarm.mak	To run over RF with commercial UEs (Applicable in Mindspeed platform only)
#define YS_MS_TX_MODE 4 In ltecl/ys_ms.h	To run over RF with commercial UEs (Applicable in Mindspeed platform only)

Timer resolution at different layers in the integrated stack.

SL No	Layer	Location	Timer Resolution value (Default Configuration).	Effective Timer Resolution(Default Configuration).
1	eGTP	File : lteenb/ve_sm_eg_cntrl.c Function : smBuildEgGenCfg() Parameter : cfg->timerRes = 1000;	1000	Timer Resolution *(**(1/SS_TICKS_SEC) * 1000) = 10 Sec.**
2	RLC/PDCP	File : lteenb/ve_sm_kw_cntrl.c Function : smBuildKwGenCfg() Parameter : genCfg->timeRes = 1;	1	Timer Resolution *(**(1/SS_TICKS_SEC) * 1000) = 10 ms.**
3	SCTP	File :lteenb/ve_sm_sb_cntrl.c Function : smBuildSbGenCfg() Parameter : cfg->timeRes = 1;	1	Timer Resolution *(**(1/SS_TICKS_SEC) * 1000) = 10 ms.**
4	S1AP	File : lteenb/ve_sm_sz_cntrl.c Function : smBuildSzGenCfg() Parameter : cfg->timeRes = (S16)VESM_LSZ_TIMERES;	10	Timer Resolution *(**(1/SS_TICKS_SEC) * 1000) = 100 ms.**

Note: If SS_TICKS_SEC value is changed appropriate changes are to be done in the timer resolution values configured for each layer.

6.3.2.1 Build eNodeB for Linux

The eNodeB integrated stack binary for Linux has built on any Linux machine.

- Go to **eNodeB** compilation directory.
- Issue the command
  - **make lnxe2e**
- The above build command generates **enodeb** binary.

6.3.2.2 Build eNodeB Integrated Stack for Cavium

The ***enb_se*** and ***enb_seum*** are two eNodeB binaries***.*** Refer to LTE eNodeB architecture on Picochip Platform section 3.3. The eNodeB binary for Cavium has built using cross compiler on Linux machine installed with Cavium SDK-1.9.0.

6.3.2.2.1 Build eNodeB SE and SEUM binaries (executed in Simple Executive Mode and Linux User space)

- Go to **eNodeB** compilation directory.
- Issue the command
  - **make cav_e2e_acc**
- The above build command generates **enb_se** binary in obj1 and **enb_seum** binary in obj2
- Copy **enb_se** and **enb_seum** binaries to **/tftpboot**/
  - **cp enb_se /tftpboot/**
  - **cp enb_seum /tftpboot/**

6.3.2.3 Build eNodeB Integrated Stack for ARM Platform

The ***enodeb*** and ***ltearm.elf*** are two eNodeB binaries. Refer to LTE eNodeB architecture Mindspeed Platform section 3.4. The Upper ARM eNodeB binary for ARM is built using cross compiler on Linux machine installed with **arm-none-linux-gnueabi-gcc**. The Lower ARM ltearm.elf is built on Windows with ***ARMCC*** compiler installed in it.

6.3.2.3.1 Build eNodeB binary for Upper ARM and Lower ARM

Upper ARM binary

- Go to **eNodeB** compilation directory.
  - **cd lteenb\build**
- Issue the command
  - **make mse2e**
- The above build command internally invokes ***enodeb_uarm.mak*** makefile with '**acc BLDENV=lnx_e2e_ms**' build environment and generates ***enodeb*** binary in ***obj*** directory
- Command to clean up the binary and object files
  - **make msclean**

Lower ARM binary

- Change director to the following location:
  - **cd .\ltephy\mindSpeed\lowercore\T4KDev\LTERT**
- Command to compile CCPU code and PHY
  - **build_fdx_ccpu_complete.bat dummy lte release**
- The above build command internally invokes ***enodeb_larm.mak*** makefile and generates ***ltearm.elf*** binary in same directory
- Command to clean up the binary and object files
  - **clear.bat**

7. Configurations for CNE, eNodeB and ueSim

Following sections explains the various configuration files and details to be taken care before executing various binaries at different nodes.

7.1 Configuration settings for Core Network Emulators

7.1.1 Configuration settings for MME

1. Go to the executable folder of MME where the **vb_acc** binary is present.
2. **vbsm_cfg.txt** and **vb_hss_ue.db** files must be present in executable folder of MME.
3. Edit the **vbsm_cfg.txt** file with the appropriate entries.
4. Snapshot of the **vbsm_cfg.txt** configuration file as follows:
   a. For demonstration purpose, user needs to update only the IP Addresses of modules.
   b. Update following fields related to IP addresses:
      i. **VBSM_ENB_ADDR**: Provide IP address of Server with eNodeB executable ready.
      ii. **VBSM_MME_IPADDR**: Provide IP address of Server with MME executable ready.
      iii. **VBSM_SGW_IP_ADDR**: Provide IP address of Server with S-GW executable ready.
      iv. **VBSM_PGW_IP_ADDR**: Provide IP address of Server with PDN-GW executable ready.

File: vbsm_cfg.txt

```
# EGTP related
VBSM_EG_DFLT_PORT            2123              # eGTP Default port
VBSM_EG_NONDFLT_PORT         2124              # eGTP Non-Default port
VBSM_EG_DFLT_HOST_NAME   "ccin.ccpu.com"  # eGTP host name

# eNodeB related
VBSM_ENB_ADDR              "172.26.0.120"      # IP address of eNodeB
VBSM_ENB_PORT               36412              # eNodeB port

# MME related
VBSM_MME_IPADDR            "172.26.10.180"     # MME IP address
VBSM_MME_SCTP_PORT          36412              # MME SCTP port

VBSM_SGW_IPADDR            "172.26.10.181"     # SGW IP address
VBSM_SGW_PORT               2123               # SGW port

VBSM_PGW_IPADDR            "172.26.10.182"     # PGW IP address
VBSM_PGW_PORT               2125               # PGW port
VBSM_UE_NUM                 1
VBSM_SCTP_UDP_SERV_TYPE        1          #service type,default 0 SCTP
# Debug mask to be set; each represent different debug masks to be set (1 and 0 to unset)
#in the form
|LVB_DBGMASK_INFO|LVB_DBGMASK_ERROR|LVB_DBGMASK_TRC|LVB_DBGMASK_MEM

VBSM_MME_DBG_MASK        1111
VBSM_DBG_MASK            1111
# Timer configuration for network initiated detach procedure
```

```
# timer value is interms of seconds
VBSM_NW_INIATED_DETACH_TIMER    1000
```

User must enter the UE details in **vb_hss_ue.db** file. Match with the required UE or UESIM under test for this setup, especially IMSI parameter.

7.1.2 Configuration settings for S-GW

1. Go to the executable folder of S-GW where the **qo_acc** binary is present.
2. **qosm_cfg.txt** file must be present in executable folder of S-GW.
3. Edit the **qosm_cfg.txt** file with the appropriate entries.
4. Below is the snapshot of the **qosm_cfg.txt** configuration file.
5. For demonstration purpose, user needs to update only the IP Addresses of modules.
6. Update following fields related to IP addresses:
7. **QOSM_SGW_IP_ADDR**: Provide IP address of Server with S-GW executable ready.

File: qosm_cfg.txt

```
Configuration file for S-GW
#####  NOTE: Do NOT change the order of the parameters #####

# EGTP related
QOSM_EG_C_DFLT_PORT          2123              # eGTP-C port
QOSM_EG_C_S11_PORT           2124              # eGTP-C port
QOSM_EG_C_S5S8_PORT          2125              # eGTP-C port
QOSM_EG_U_DFLT_PORT          2152              # eGTP-U port
QOSM_EG_DFLT_HOST_NAME    "ccin.ccpu.com"      # eGTP host name
QOSM_SGW_IPADDR           "172.26.10.181"      # SGW IP address
QOSM_UE_NUM                  1

# Debug mask to be set; each represent different debug masks to be set (1 and 0 to unset)
#in the form
|LQO_DBGMASK_INFO|LQO_DBGMASK_ERROR|LQO_DBGMASK_TRC|LQO_DBGMASK_MEM

QOSM_SGW_DBG_MASK            0000
QOSM_SM_DBG_MASK             0000
```

7.1.3 Configuration settings for P-GW

1. Go to the executable folder of S-GW where the **av_acc** binary is present.
2. **avsm_cfg.txt** file must be present in executable folder of S-GW.
3. Edit the **avsm_cfg.txt** file with the appropriate entries.
4. Below is the snapshot of the **avsm_cfg.txt** configuration file**.**
5. Update the IP Addresses of modules for the purpose of demonstration.
6. Update following fields related to IP addresses:
7. **AVSM_PGW_DFLT_ADDR**: Provide IP address of Server with PDN-GW executable ready.
8. **AVSM_PGW_DFLT_EXGW_ADDR:** Provide IP address of external server [For example, Video Server] which intends to communicate with PDN-GW.
9. **AVSM_PGW_UE_START_ADDR**: Provide IP address of Server where UE client [For example: Video Client] is present. This is the IP address assigned by PDN-GW for the first UE. PDN-GW then uses this IP Address as a reference for new assignment to a new UE.
10. Make sure the source port number of VLC Server or the data generator is configured with port less than **AV_PGW_MAX _WELL_KNOWN_IP_POR**T specified in **av.h** file.
11. **AVSM_PGW_IP_NUM: Number of IP addresses configured for UE Clients.**

File: avsm_cfg.txt

```
Configuration file for PDN-GW
#####  NOTE: Do NOT change the order of the parameters #####

# EGTP related
AVSM_EG_C_DFLT_PORT          2123              # eGTP-C port
AVSM_EG_C_NONDFLT_PORT       2124              # eGTP-C port
AVSM_EG_U_DFLT_PORT          2152              # eGTP-U port
AVSM_EG_DFLT_HOST_NAME     "ccin.ccpu.com"     # eGTP host name

# PDN Gateway (PGW) related
AVSM_PGW_DFLT_ADDR         "172.26.10.182"     # PGW IP address
AVSM_PGW_DFLT_EXGW_ADDR    "172.26.10.184"     # External Server address (Video Server)
AVSM_PGW_UE_START_ADDR      "172.26.10.7"      # Starting address of UE (Video Client)
AVSM_PGW_UE_NUM              1                 # Number of UEs supported
AVSM_PGW_IP_NUM            10

# SGW related
AVSM_SGW_IPADDR            "172.26.10.181"     # SGW IP address
AVSM_SGW_PORT               2152               # SGW port

# Debug mask to be set; each represent
#different debug masks to be set (1 and 0 to unset)
#in the form
|LAV_DBGMASK_INFO|LAV_DBGMASK_ERROR|LAV_DBGMASK_TRC|LAV_DBGMASK_MEM

AVSM_PGW_DBG_MASK          0000
AVSM_DBG_MASK              0000
```

7.2 Configuration settings for eNodeB

1. Go to the executable folder of **eNodeB/seum** (Cavium) where the **enb_seum** binary is present.
2. Edit the **ve_cfg.txt** file with the appropriate entries.
   a. For demonstration purposes, update only the IP Addresses of modules.
   b. Update following IP address:
      i. **enbIpAddr:** Provide IP address of Server where it is intended to run eNodeB.
      ii. **mmeIpAddr:** Provide IP address of Server where it is intended to run MME.
      iii. **sctpIpAddr:** Provide IP address of Server where it is intended to run eNodeB.
   c. **inactvTmrVal:** Specifies the time in milliseconds after which Inactivity timer in the eNodeB expires.
   d. **maxExpires:** Specifies the number of Expiries after which inactivity of the UE is detected.
   e. sctpUdpServiceType: Specifies 0 for SCTP over RAW IP and 1 for SCTP over UDP

File: ve_cfg.txt snapshot

```
cellId       = 1
modType      = 0
duplexMode   = 1
maxUeSupp    = 100
mcc0         = 0
mcc1         = 0
mcc2         = 0
mnc0         = 0
mnc1         = 0
mnc2         = 0
trackAreaCode = 0
freqBandInd   = 1
enbIpAddr     = 172.26.0.120
mmeIpAddr     = 172.26.10.180
sctpIpAddr    = 172.26.0.120
hiDbg        = 0
sbDbg        = 0
szDbg        = 0
egDbg        = 0
veDbg        = 0
nhDbg        = 0
kwDbg        = 0
rgDbg        = 0
ysDbg        = 0
smDbg        = 1
inactvTmrVal  = 90000
maxExpires    = 50
sctpUdpServiceType = 1
```

3. For PAL, edit the **ys_cfg.txt** file in the folder where **enb_seum** binary is present.
   a. For demonstration purposes, update only the following IP addresses:
      i. **YS_IP_ADDR_ENB: Provide IP address of the server where it is intended to run eNodeB.**
      ii. **YS_IP_ADDR_UE: Provide IP address of the machine where it is intended to run the ueSim.**
      iii. **YS_TTI_TMR_VAL_CFG: To configure TTI value.**

File: ys_cfg.txt

```
# Configuration file for PAL
YS_PORT_ENB          6789            # eNodeB port
YS_PORT_UE           9876            # UE port
YS_IP_ADDR_ENB       172.26.0.120    # eNodeB address
YS_IP_ADDR_UE        172.26.10.9     # ueSim address towards eNodeB
YS_TTI_TMR_VAL_CFG   1000000         #  TTI value 1000000 -> 1ms TTI, 10000000-> 10ms TTI
```

7.3 Configuration settings for ueSim

7.3.1 Cavium Configuration

Go to **lteue** folder and change the following macros in **ue_app.h** file appropriately.

```
UE_APP_SELF_ETH_INF_ADDR    172.26.10.8      # ueSim address towards VC
UE_APP_SELF_PDN_ADDR        172.26.10.7      # Address where VC is running
UE_APP_DATA_SRVR_ADDR       172.26.10.184    # Address where VS is running
UE_APP_DATA_SRVR_PORT       8080             # VS port
```

After making the changes, go to build folder and build the **uesim** binary.

7.3.2 Linux Configuration

1. Go to the folder where **uesim binary is present**.
2. Edit the **uesim_cfg.txt** file with the appropriate entries.
   a. Update only the IP Addresses of modules for demonstration purpose.
   b. Update following IP address:
      i. **UE_APP_SELF_ETH_INF_ADDR:** Provide IP address of Server through which ueSim is directly connected to UE client [VC].
      ii. **UE_APP_SELF_PDN_ADDR:** Provide IP address of Server where it is intended to run VC.
      iii. **UE_APP_DATA_SRVR_ADDR:** Provide IP address of Server where it is intended to run VS.

File: uesim_cfg.txt

```
# Configuration file for UE-SIM
UE_APP_SELF_ETH_INF_ADDR    172.26.10.8      # ueSim address towards VC
UE_APP_SELF_PDN_ADDR        172.26.10.7      # Address where VC is running
UE_APP_DATA_SRVR_ADDR       172.26.10.184    # Address where VS is running
UE_APP_DATA_SRVR_PORT       8080             # VS port
```

3. For PAL, edit the **ys_cfg.txt** file in the folder where **uesim** binary is present.
   a. For demonstration purposes, update only the following IP addresses:
      i. **YS_IP_ADDR_ENB: Provide IP address of the server where it is intended to run eNodeB.**
      ii. **YS_IP_ADDR_UE: Provide IP address of the machine where it is intended to run the ueSim.**

File: ys_cfg.txt

```
# Configuration file for PAL
YS_PORT_ENB          6789          # eNodeB port
YS_PORT_UE           9876          # UE port
YS_IP_ADDR_ENB       172.26.0.120  # eNodeB address
YS_IP_ADDR_UE        172.26.10.9   # ueSim address towards eNodeB
```

4. For UL data traffic, destination port must be less than **UE_APP_DATA_SRVR_PORT** defined in **uesim_cfg.txt** file.
5. Check route information and verify route is created for Video Client. Use the following command to add the route rule, if it does not exist:
   - ***route add –host <IP address of UE client> gw < UE_APP_SELF_ETH_INF_ADDR> dev eth0***

7.4 Configuration setting of Video Client

Check route information and verify route is created. Use the following command, for example:

- ***route add –host <IP address of VC client machine> gw <IP address of ueSim machine> dev eth0***

8 Execution of the Nodes

This section describes the execution of nodes.

8.1 CNE Execution Steps

Execute the following steps to start the CNE'S:

a. MME Emulator -> **vb_acc** binary
b. S-GW Emulator -> **qo_acc** binary
c. PDN-GW Emulator -> **av_acc** binary

8.2 UESim Execution steps

8.2.1 Linux Platform

On Linux Machine 1, start the UeSim.

a. ueSim -> **uesim** binary

8.2.2 Cavium Platform

On Cavium Machine: Use hyper-terminal for accessing Cavium board.

a. Copy uesim binary in the tftpboot directory.
b. Reboot the machine to get boot prompt.
c. Set following environmental variables before any further action:
   - ***setenv ipaddr <IP address of Cavium board>***
   - ***setenv serverip <IP address of machine from where Cavium images/binaries are to loaded from>***
   - ***setenv netmask 255.255.0.0***
   - ***setenv ethact octeth0***
   - ***setenv gateway <IP address> say 172.26.0.254***
   - ***setenv bootue 'tftpboot 0x40000000 uesim ; bootoct 0x40000000 coremask=0x4'***
   - ***setenv bootlin 'tftpboot 0x40000000 vmlinux.64 ; bootoctlinux 0x40000000 coremask=0x2'***
   - ***setenv bootall 'run bootlin ; run bootue'***
   - ***saveenv – This helps to save the environment variables***
   - ***printenv – This command is to check the already set variables***
d. Reboot the machine to get boot prompt run the below command to run UE and get the linux prompt.
   - ***Run bootall***

8.3 eNodeB Execution Steps

8.3.1 Linux Platform

On Linux Machine 2, start the eNodeB:

a. eNodeB -> enodeb binary(in build/obj directory)

8.3.2 Cavium Platform

On Cavium Machine, use hyper-terminal for accessing Cavium board.

a. Reboot the machine to get boot prompt.
b. Set following environmental variables before any further action:
   - ***setenv ipaddr <IP address of Cavium board>***
   - ***setenv serverip <IP address of machine from where Cavium images/binaries are to loaded from>***
   - ***setenv netmask 255.255.0.0***
   - ***setenv ethact octeth0***
   - ***setenv gateway <IP address> say 172.26.0.254***
   - ***setenv bootlin 'tftpboot 0x40000000 vmlinux.64 ; bootoctlinux 0x40000000 coremask=0x2'***
   - ***setenv bootenb 'tftpboot 0x40000000 enb_se ; bootoct 0x40000000 coremask=0x9'***
   - ***setenv bootall 'run bootlin ; run bootenb'***
   - ***saveenv – This helps to save the environment variables***
   - ***printenv – This command is to check the already set variables***
c. Reboot the machine to get boot prompt run the below command to run enb_se and get the linux prompt.
   - ***Run bootall***
d. The Cavium board after last command must be running Linux. Run the following sequence on Linux prompt:
   - ***modprobe cavium-ethernet***
   - ***ifconfig eth0 <IP address of Cavium board> up***
   - ***ftpget -u <login> -p <password> <IP address of FTP server where enb_seum binaries are kept> seum /tftpboot/seum***
   - ***ftpget -u <login> -p <password> <IP address of FTP server where enb_seum binaries are kept> ve_cfg.txt /tftpboot/ve_cfg.txt***
   - ***ifconfig eth1 up***
   - ***./enb_seum***

8.3.3 ARM Platform

4GMX compatible board has 2 serial ports.

a. Connect the 4GMX compatible board to desktop/laptop considered as host machine on 1st serial port interface and necessary plugins.
b. Reset the board. Kernel images gets loaded. We get command prompt. This confirms that the U-ARM code loads successfully. Hence is the U-ARM command prompt.
c. Copy the ltearm.elf to /tftpboot/ of host machine
d. Issue ***bootarm ltearm.elf.*** This loads the necessary binaries to L-ARM.
e. Now move to eNodeB's build environment in U-ARM, from command prompt. That is, move to build/
f. Issue command ***./obj/enodeb*** This executes binary enodeb from U-ARM of the board.
g. if we observe packet loss in end to end setup before it reaches PDCP in DL or PGW in UL , execute following commands to increase the udp buffer size for avoiding the same:

   sysctl -w net.core.rmem_max=8388608

   sysctl -w net.core.wmem_max=8388608

   sysctl -w net.core.rmem_default=6553600

   sysctl -w net.core.wmem_default=6553600

8.4 VNC Server Execution Steps

8.4.1 Windows Platform

Running VLC media player on Windows:

a. Start VLC player on server machine
b. Click Media -> Streaming -> Add
c. Add the movie clip to play on the client
d. Click Stream -> Next
e. Select HTTP in “File” drop down menu in Destinations -> Add
f. Give <IP address of video server> in Address and Port <8080>
g. Uncheck “Active Transcoding” in Transcoding Options
h. Click Next -> Stream
i. Streaming starts.

8.4.2 Linux Platform

Running VLC media player on Linux machine using HTTP protocol:

a. vlc -vvv <movie clip> --loop --sout '#std{access=http, mux=ts, dst=<IP address of video-server>:8080}'

8.5 VNC Client Execution Steps

Running VLC player on Video Client Windows/Linux machine using HTTP protocol:

a. Start VLC player on client machine
b. Click Media -> Open Network Stream
c. Select Protocol as HTTP
d. Type address as <IP address of video-server>:8080
e. Click Play button.

9 Demonstration of LTE End-to-End Demo on Linux Platform

9.1 Demonstration of Video Streaming

Execute the following steps to start the demonstration:

1. On Linux Machine 1 start the Core Network Emulators (refer 7.1)
2. On Linux Machine 2, start enodeb (refer 7.3.1)
3. On Linux Machine 2, start the ueSim (refer 7.2.1)
4. To demonstrate data: VLC media server must start before the VLC media player in client machine.
    a. Running VLC media player on Windows or Linux (refer 7.4)
    b. Running VLC player on Video Client Linux machine using HTTP protocol (refer 7.5)
        i. Verify the streaming of the video now.

9.2 Demonstration of Mobile Termination Call

Execute the following steps to start the demonstration:

1. On Linux Machine 1 start the Core Network Emulators (refer 7.1)
2. On Linux Machine 2, start enodeb (refer 7.3.1)
3. On Linux Machine 2, start the ueSim (refer 7.2.1)
4. After the Attach of UE is complete, if the data is not sent; the Inactivity Timer at the eNodeB expires, and the UE reaches the ECM_IDLE state in which the RRC Connections are disconnected.
5. Send the video data to the video client using the steps in 7.4 after the UE reaches ECM_IDLE state (VLC media server must start before the VLC media player in client machine).
6. On receipt of the data packets, buffering is performed at the SGW, and the signaling path is re-established.
7. Once the signaling path is re-established, the data is transmitted to the Video Client.

9.3 Demonstration of Establishment of Dedicated Bearer and Transmitting Data on Dedicated Bearer

Execute the following steps to start the demonstration:

1. On Linux Machine 1 start the Core Network Emulators (refer 7.1)
2. On Linux Machine 2, start enodeb (refer 7.3.1)
3. On Linux Machine 2, start the ueSim (refer 7.2.1) [ueSim must be built by enabling the flag UESIM_TRIGGER_DRB_EST this triggers the enabling of the Dedicated Bearer].

2 To demonstrate data: VLC media server must start before the VLC media player in client machine.

a. Running VLC media player on Windows
    i. Start VLC player on server machine
    ii. Click Media -> Streaming -> Add (Add the movie clip to play on the client)
    iii. Click Stream -> Next
    iv. Select UDP in “File” drop down menu in Destinations -> Add
    v. Give <**IP address of video server**> in Address and Port <**1235**>

The data sent for ports ranging from 1235 to 1255 is sent on the Dedicated Bearer. The data sent for port 1234 or lesser is sent on the Default Bearer.

i. Uncheck “Active Transcoding” in Transcoding Options
ii. Click Next -> Stream
iii. Streaming starts.

b. Running VLC media player on Linux machine using UDP protocol:
   - ***vlc -vvv <movie clip> --loop --sout '#std{access=udp, mux=ts, dst=<IP address of video-server>:1235}'***

c. Running VLC player on Video Client Linux machine using UDP protocol:
   i. Start VLC player on client machine
   ii. Click Media -> Open Network Stream
   iii. Select Protocol as UDP
   iv. Type address as <**IP address of video-server**>:**1235**
   v. Click Play button
   vi. Verify the streaming of the video now
   vii. The data at the UE Simulator must be received on **lcId** = **4** (Dedicated Bearer).

10 Demonstration of LTE End-to-End Demo on EVM Platform

10.1 Demonstration of Video Streaming

This demonstration replicates the Linux End-to-End Demo setup except that the eNodeB runs on EVM board (refer 7.3) and the ueSim runs on Linux machine.

10.2 Demonstration of the Mobile Terminating Call

This demonstration replicates the Linux End-to-End Demo setup except that the eNodeB runs on EVM board (refer 7.3) and the ueSim runs on Linux machine.

10.3 Demonstration of Establishment of Dedicated Bearer and Transmitting Data on Dedicated Bearer

This demonstration replicates the Linux End-to-End Demo setup except that the eNodeB runs on EVM board (refer 7.3) and the ueSim runs on Linux machine.

11 Demonstration of LTE End-to-End Demo on Cavium Platform

11.1 Demonstration of video streaming

Execute the following steps to start the demonstration:

1. On Linux Machine 1start the Core Network Emulators (refer 7.1)
2. On Cavium Machine start enodeb (refer 7.3.2)
3. On Cavium Machine start ueSim (refer 7.2.2)
4. Try to check whether client machine is able to ping server machine and vice versa.
5. To demonstrate data:
   a. Start VLC player on Video Server Linux machine.
      - ***vlc -vvv Avatar\ -\ Trailer\ \(Medium\).mov --loop --sout '#std{access=http, mux=ts, dst=172.26.0.19:8080}'***

        Where:

        "**Avatar\ -\ Trailer\ \ (Medium\).mov**": movie name playing on video server.

        172.26.0.19: Server IP addresses running video server.

   b. Start VLC player on Video Client Linux machine.
      i. Start VLC player on client machine
      ii. Click Media -> Open Network Stream
      iii. Select Protocol as HTTP
      iv. Type address as 172.26.0.19:8080
      v. Click Play button and the movie is started.

11.2 Demonstration of Mobile Termination Call

Execute the following steps to start the demonstration:

1. On Linux Machine 1start the Core Network Emulators (refer 7.1)
2. On Cavium Machine start enodeb (refer 7.3.2)
3. On Cavium Machine start uesim (refer 7.2.2)
4. Try to check whether client machine is able to ping server machine and vice versa.
5. To demonstrate data:
   a. Start VLC player on Video Server Linux machine.
      - ***vlc -vvv Avatar\ -\ Trailer\ \(Medium\).mov --loop --sout '#std{access=http, mux=ts, dst=172.26.0.19:8080}'***

        Where:

        "**Avatar\ -\ Trailer\ \(Medium\).mov**": movie name playing on video server.

        172.26.0.19: Server IP addresses running video server.

   b. Start VLC player on Video Client Linux machine.
      i. Start VLC player on client machine
      ii. Click Media -> Open Network Stream
      iii. Select Protocol as HTTP
      iv. Type address as 172.26.0.19:8080
      v. Click Play button and the movie is started.

11.3 Demonstration of Establishment of Dedicated Bearer and Transmitting Data on Dedicated Bearer

Execute the following steps to start the demonstration:

1. On Linux Machine start the Core Network Emulators (refer 7.1)
2. On Cavium Machine start enodeb (refer 7.3.2)
3. On Cavium Machine, start the ueSim (refer 7.2.2) [ueSim must be built by enabling the flag UESIM_TRIGGER_DRB_EST). This triggers the enabling of the Dedicated Bearer].
4. Try to check whether client machine is able to ping server machine and vice versa.
5. To demonstrate data:
   a. Start VLC player on Video Server Linux machine.
      - ***vlc -vvv Avatar\ -\ Trailer\ \(Medium\).mov --loop --sout '#std{access=udp, mux=ts, dst=172.26.0.19:1235}'***

        Where:

        "**Avatar\ -\ Trailer\ \ (Medium\).mov**": movie name playing on video server.

        172.26.0.19: Server IP addresses running video server.

   b. Start VLC player on Video Client Linux machine.
      i. Start VLC player on client machine
      ii. Click Media -> Open Network Stream
      iii. Select Protocol as HTTP
      iv. Type address as 172.26.0.19:1235
      v. Click Play button and finally the movie is started
      vi. Ensure that the data is received on lcId=4 (Dedicated Bearer).

12 End-to-End Call Flow (Message Sequence Chart)

S1 Setup and Valid UE Attach with Default Bearer Establishment

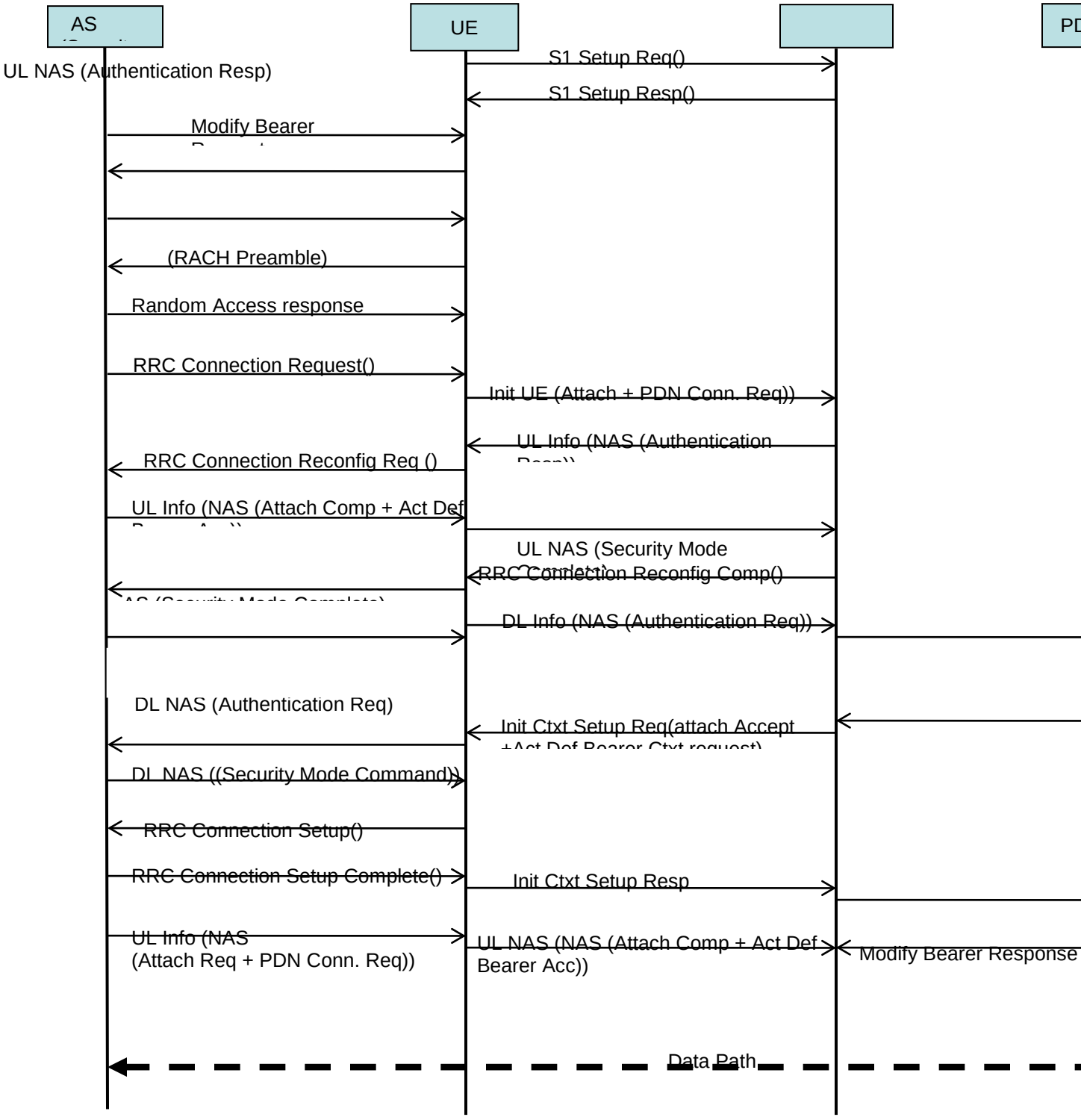


Figure 9: LTE Control and Data Call flow

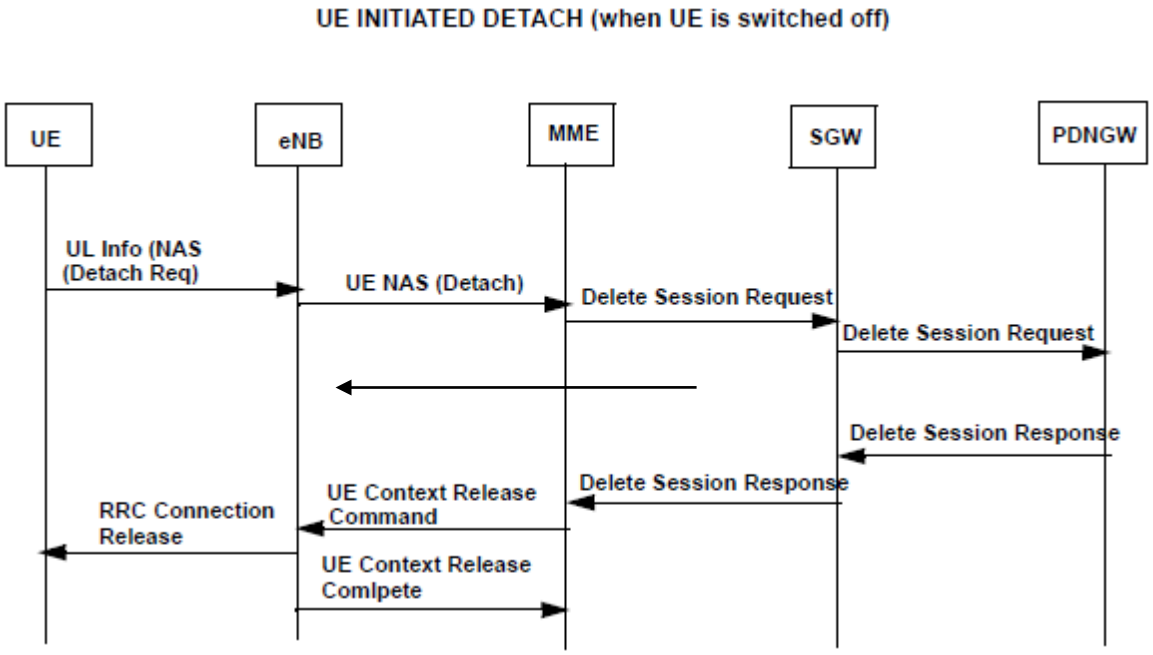


Figure 10: Message Sequence Flow between UE and CNE through eNodeB

S1 Setup and Valid UE Attach with Default and Dedicated Bearer Establishment

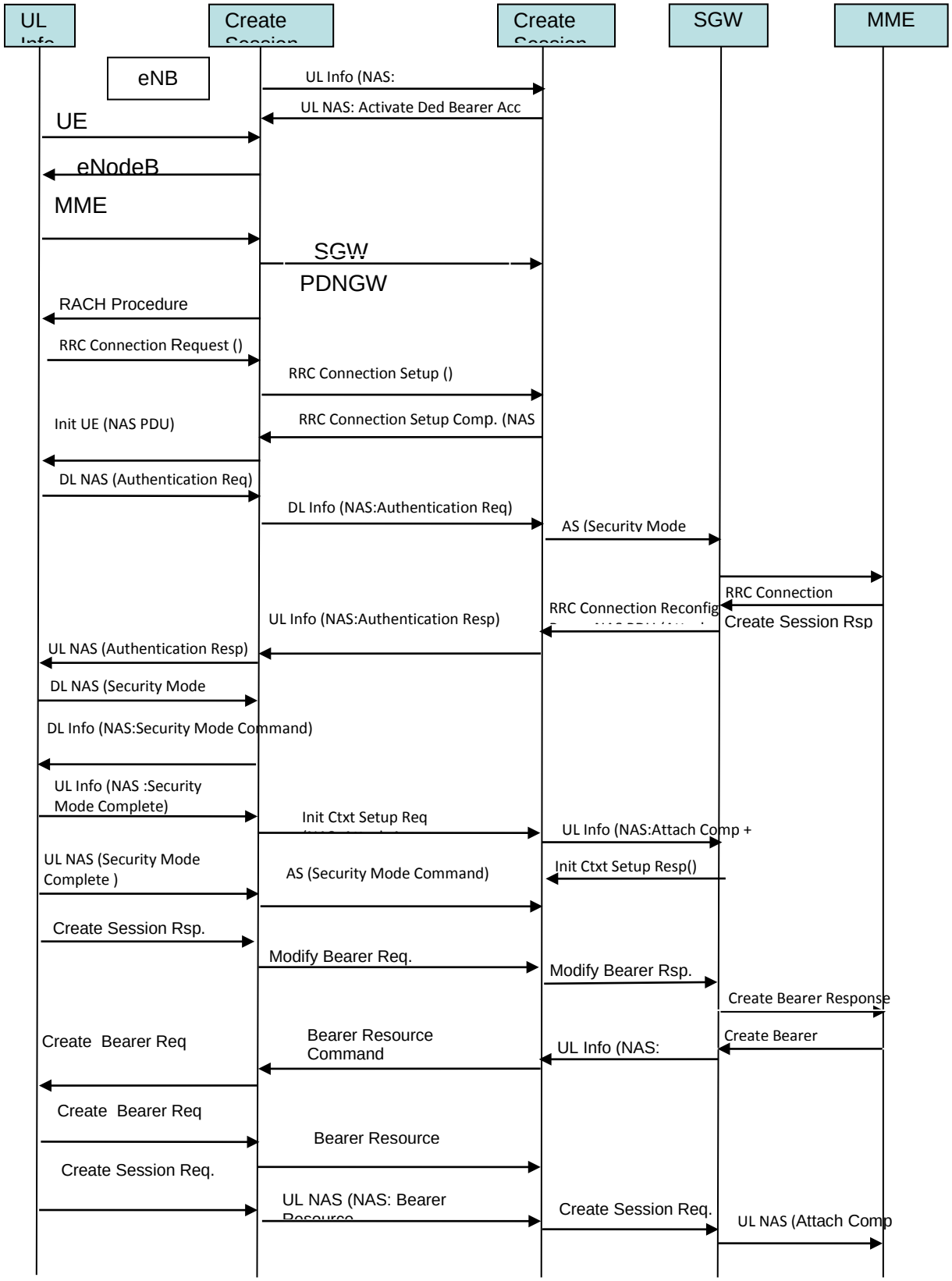


Figure 11: LTE Control and Data Call flow using Dedicated Bearer

Mobile Terminating Call Sequence

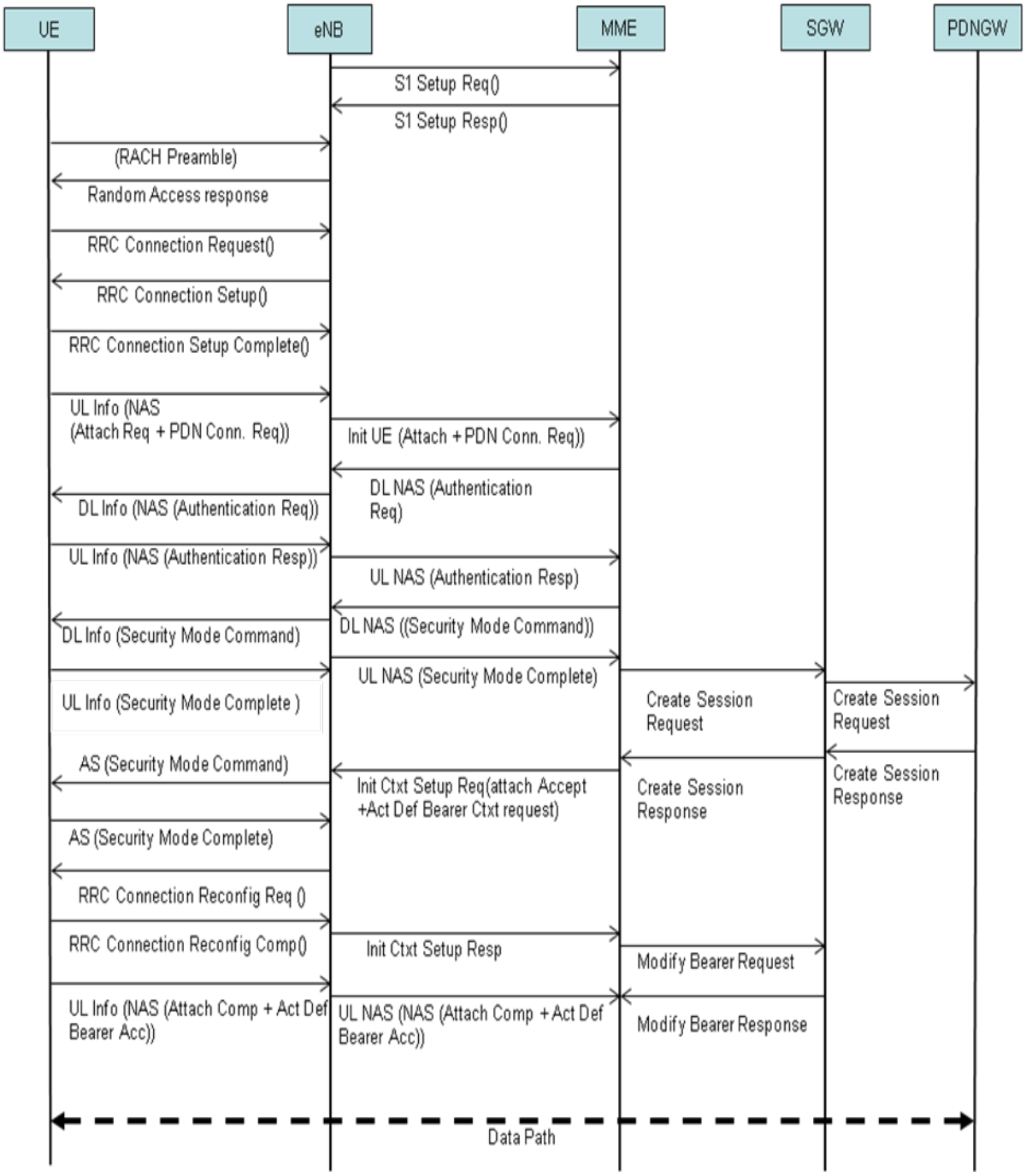


Figure 12: Mobile Terminating Call Sequence - Flow 1

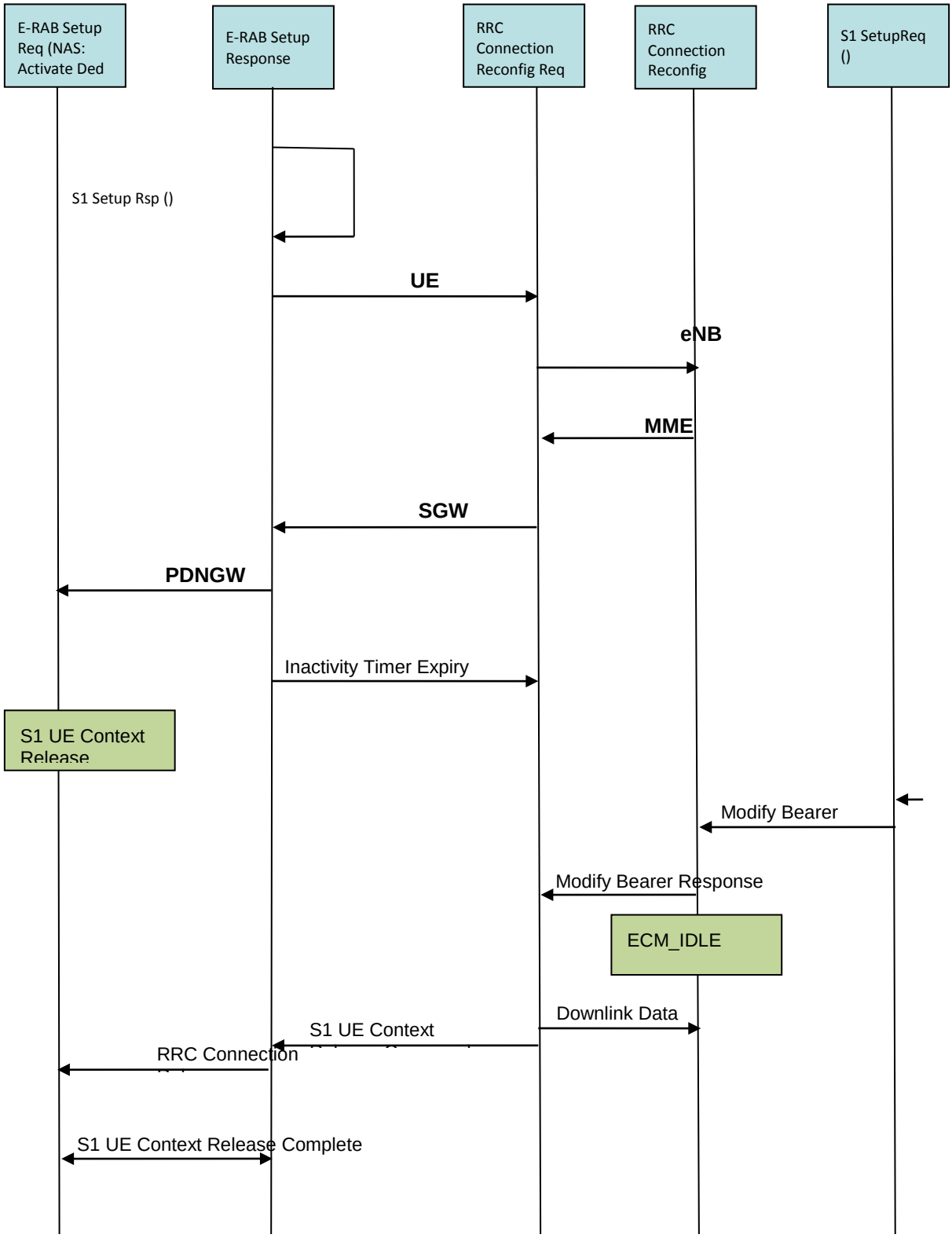


Figure 13: Mobile Terminating Call Sequence - Flow 2

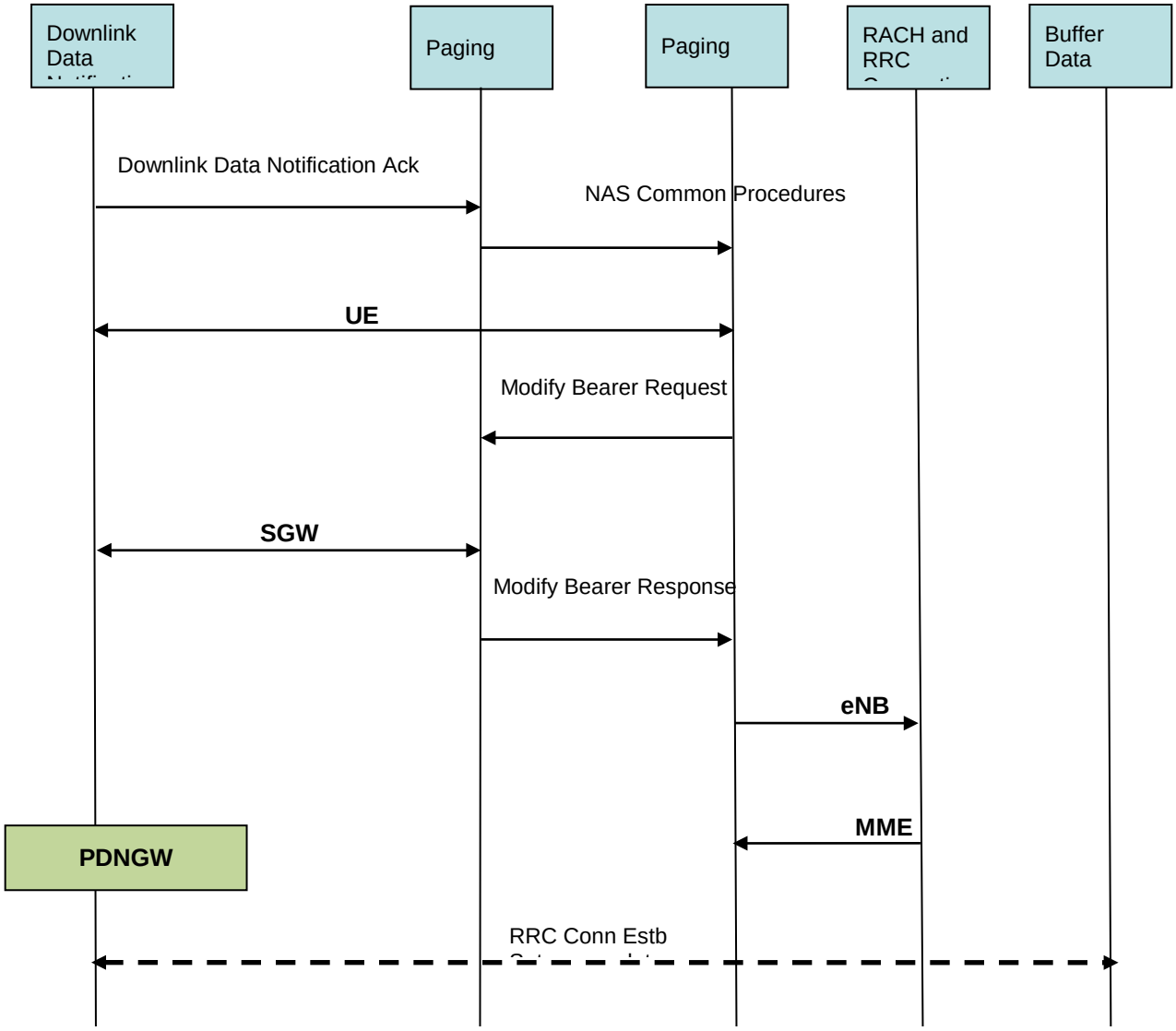


Figure 14: Mobile Terminating Call Sequence - Flow 3

13 Troubleshooting

This section describes the solution for the common problems that are seen while executing the demo:

1. Messages exchange between CNE or eNodeB fails:
   a. MME and eNodeB must run from root mode, since SCTP uses RAW sockets. If root access is not available then run SCTP over UDP sockets. For information on how to configure UDP sockets, contact support@trillium.com.
   b. PDN-GW must run with root privilege, since it uses PCAP library to communicate with external network [Video Server].
   c. Verify that the right IP addresses are provided in **ve_cfg.txt** and **vbsm_cfg.txt** files.
   d. Verify platform and Server running CNEs are reachable on Ethernet interface. Try PING command from both the sides to receive reply.

2. Failed to bring up binaries.
   a. Verify vbsm_cfg.txt configuration file and vb_hss_ue.db database file present with MME binary.
   b. Verify qosm_cfg.txt configuration file present with SGW binary.
   c. Verify avsm_cfg.txt configuration file present with PGW binary.
   d. Verify ve_cfg.txt and ys_cfg.txt configuration files present with eNodeB binary.
   e. Verify uesim_cfg.txt and ys_cfg.txt configuration files present with UeSim binary.

3. Video Client is not able to run Video:
   a. Run **route** command and verify proper routes are created on servers running video client and ueSim.
   b. Verify Server running VC is reachable on Ethernet interface from Server running ueSim and vice-versa. Try PING command from both the sides to receive reply.
   c. Verify that same protocol and address is given as VLC running on Video Server machine.
   d. Verify ueSim is running on Server whose IP address is same as it is configured in **uesim_cfg.txt** file.

4. ueSim binary fails:
   a. ueSim must run using root permissions, since it uses PCAP library.
   b. Verify the different IP address parameters given in **uesim_cfg.txt** file are as per the test setup.

Note: Verify that **ys_cfg.txt** file has proper eNodeB and ueSim IP addresses. The variables to be checked are:

YS_IP_ADDR_ENB for **eNodeB IP** and

YS_IP_ADDR_UE for **ueSim IP**.

14 References

Refer to the following documents for additional information.

- *LTE MME Reference Application Functional Specification*, Continuous Computing.
- *LTE SGW Reference Application Functional Specification*, Continuous Computing.
- *LTE PGW Reference Application Functional Specification*, Continuous Computing.
- *LTE eNodeB Reference Application Functional Specification*, Continuous Computing.
- *LTE UE Simulator Functional Specification*, Continuous Computing.

Notes

9450 Carroll Park Drive

San Diego, CA 92121-2256

858-882-8800

www.radisys.com