Design and Implementation of System Under Test for LTE Protocol Conformance Testing

Yin Chen, Zhizhong Ding, Dingliang Wang, and Teng Huang
Institute of Communications and Information Systems, Hefei University of Technology, Hefei 230009, China
Email: {hfut_cy, wdlwshcjf}@163.com; zzding@hfut.edu.cn; Huangteng66@gmail.com

Abstract —The test suites of protocol conformance released by European Telecommunications Standards Institute (ETSI) are used to examine whether user equipment (UE) conforms to the specified protocols. For developers and engineers, the workflow of test case is very helpful to comprehend the technical specifications. However, all the test suites developed in TTCN-3 language can not be run without building up a development environment which should connect with the System Under Test (SUT), the target to be tested. Unfortunately, it is quite time consuming and expensive to build up an actual standard test environment specified by the technical specifications at the start phase of development. To solve the problem, this paper proposes a feasible test model in which the SUT is simulated on a common computer. The main idea is that the software SUT implemented in TTCN level needs not to simulate all the details of the System Simulator and UE depicted in standard test environment, but only implements some necessary steps as long as the test suites could be run with correct test procedures and functions. What is more, the developed SUT can be reused for the development of many other test suites which have the similar procedures, due to its high flexibility and easy modification.

Index Terms—protocol conformance, test suites, SUT, TTCN

I. Introduction

Protocol Conformance testing is the testing to determine whether a product or system meets the specified protocols which are developed by authoritative organizations [1]. It is necessary for Long Term Evolution (LTE) UE to pass standardized tests to ensure its correct interaction with other LTE entities. Therefore, ETSI has been developing the TTCN-3 test suites and test system of protocol conformance testing which now are a part of 3rd Generation Partnership Project (3GPP) technical specifications named as TS 36.523-3 [2].

The standard test environment in TS 36.523-3 is shown in Fig. 1. Host-PC is a high performance computer needed for compiling and running the test suites. The test suite developed by Testing and Test Control Notation Version 3 (TTCN-3) only runs in the Host-PC. It sends signaling and configuration messages to system simulator (SS) to create real circumstance for UE. What's more, it is responsible for making verdict according to the message received from UE and SS. SS Hardware provides the adaption to UE to realize all the necessary lower protocol layers such as physical layer. UE, i.e. a mobile phone, is the target under test.

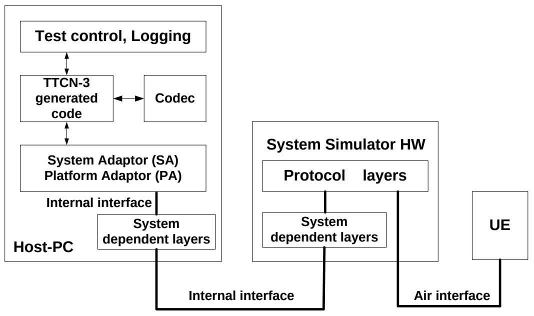

Fig. 1. General system architecture.

At the start of developing, it is helpful and useful for developer and engineer to run the test suites published by ETSI in order to understand the specifications correctly and deeply since the example codes give us much more details. In general, the actual test system is built up in [3] and [4] to do the Radio Resource Management (RRM) protocol conformance testing. In addition, in other protocol conformance testing fields, the test system is also built up [5]. However, the TTCN-3 test case can not be executed without SUT even it is in a commercial integrated development environment (IDE). It is expensive and time consuming to build up a standard test environment.

For the above reasons, this paper proposes a feasible test model and designs a SUT running on a common PC to replace the SS and UE in Fig. 1. With this approach, it is much easier to make the test suites run. Furthermore, the developed SUT could be a useful reference when developing SUT for other specifications. For example, the SUT of Radio Resource Control (RRC) conformance testing could be very useful for the development of Radio Resource Management (RRM) test suites since many procedures of both tests are quite similar or even the same. It should be emphasized here that the UE and SS are not totally simulated, but implemented only for the purpose that the test suites could be run with correct test procedures and functions.

The rest of this paper is organized as follows. Section II introduces TTCN-3 language and explains the reason to use TTCN-3. The reason why TTCN-3 test cases can not run independently is also explained. In section III, the LTE concepts which are closely related to SUT are introduced. In section IV, the test system proposed by

Manuscript received November 13, 2014; revised February 26, 2015.
doi:10.12720/jcm.10.2.145-153

ETSI will be analyzed from the view of LTE. On this basis, the test model of this paper and its mechanism will be educed. And the test suites are then analyzed to lead to the structure of SUT. In section V, the implementation process of SUT is shown. An experiment is given to verify the correction of SUT in section VI. At last, we conclude this paper with some future work.

II. TTCN-3 ANALYSIS

A. *The Advantages of TTCN*

Testing and Test Control Notation Version 3 (TTCN-3) developed by ETSI is an internationally standardized language for defining test specifications. It allows the concise description of test behavior. TTCN-3 provides all the constructs and features necessary for black box testing. It contains rich data types and very powerful matching mechanisms. TTCN-3 also supports message-based communication, procedure-based communication, timer handling, and dynamic test configuration. Above all, TTCN-3 is a language for testing.

To avoid misunderstanding the standards, ETSI uses TTCN-3 language to develop a large number of test cases according to the technical specifications. The UE should pass the entire tests before entering into the market. In 2000s, Global Certification Forum (GCF) confirmed that the TTCN test suite is the unique and standard test suite for protocol conformance testing.

Now TTCN-3 has more and more applications in many testing fields, such as car lights testing system [6], property testing [7] and web penetration testing [8]. What's more, the entire test behaviors can also be described briefly by it, so we designed and implemented the SUT by TTCN-3.

B. *The Structure of TTCN-3 Test System*

A TTCN-3 test system can be conceptually defined as a collection of different test system entities, which interact with each other during the execution of test suites. All the entities cooperate with each other to communicate with SUT. The entire components in the system are shown in Fig. 2.

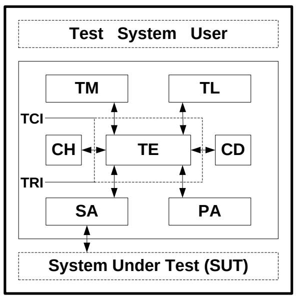

Fig. 2. The structure of TTCN-3 test system.

Test Management (TM) is responsible for the overall management of test system. Test Logging (TL) can generate and maintain test log. Component Handling (CH) is responsible for managing component, especially in parallel test system. The main task of Test Execution (TE) is to execute the abstract test suite. The Codec (CD) encodes data of TTCN types into bit-stream and vice versa. The role of SA is to provide the methods for communication between TE and SUT. It is also the bridge between the abstract TRI communication and real communication mechanisms employed by the SUT. Platform Adaptor (PA) implements TTCN-3 external functions and provides a TTCN-3 test system with a single notion of time. TTCN-3 Control Interface (TCI) provides means for TE to manage test execution, distributed execution of test components among different test devices, encoding and decoding operation and log information about test execution. TTCN-3 Runtime Interface (TRI) provides means for TE to send test data to the SUT, to manipulate timers, and to notify the TE of received test data and timeouts [9].

In general, most entities of the test system are completed by TTCN-3 Integrated Development Environment (IDE). Only the SA and CD should be developed by users. Because of the abstract nature of TTCN-3 communication mechanism, it will usually be up to the test system developers to implement these particular test system entities. And in different environments, different encode and decode rules may be required.

C. *The Importance of SUT in Test System*

In this section, the process of exchanging message between TE and SUT will be introduced, from which the function of SA, CD and TE will be shown in detail, and the importance of SUT will be explained and emphasized.

The whole process is shown in Fig. 3 [10]. At the start, TE begins executing the control part of test suite. When TTCN-3 execute statement is encountered, TE invokes the *triExecuteTestCase* operation to inform SA that a new test case is about to be started. This allows SA to prepare its communication facilities to be used. After mapping operation, TE invokes the *tciEncode* operation to encode the message to be sent. TE starts the timer soon after the message is sent out.

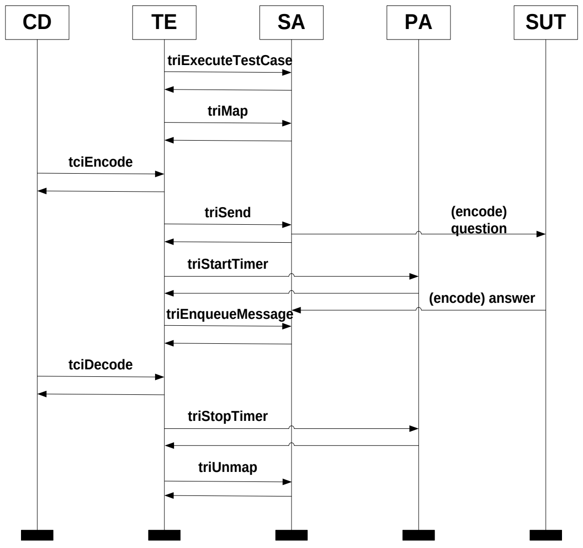

Fig. 3. General process of exchanging message.

As we can see, SUT is very important, it is responsible for receive and send message to continue communication.

Some companies try to develop a plugin to imitate SUT. However, currently the plugin is so simple that only the exact received message can be returned to TE, and obviously it can not be used in most situations.

III. LTE Introduction

A. LTE Definition

Long Term Evolution (LTE) is a standard for wireless communication of high data rate mobile phones and terminals. It is a new technique after the 3rd generation of mobile communication. LTE was first proposed in 2004. In May 2007, the LTE/SAE Trial Initiative alliance was founded with the goal of verifying and promoting the new standard in order to ensure the global introduction of the technology as quickly as possible. The LTE standard was finalized in December 2008, and the first available LTE service was launched on December 14, 2009 [11].

B. Structure of LTE System

As shown in Fig. 4, LTE system is consisted of Evolved Packet Core network (EPC) and evolved universal terrestrial radio access network (E-UTRAN). EPC splits into Mobility Management Entity (MME) and Serving Gateway (S-GW). E-UTRAN only includes evolved Node B (eNode B). EPC connects with eNode B by S1 interface. ENode B connects with each other by X2 interface [12].

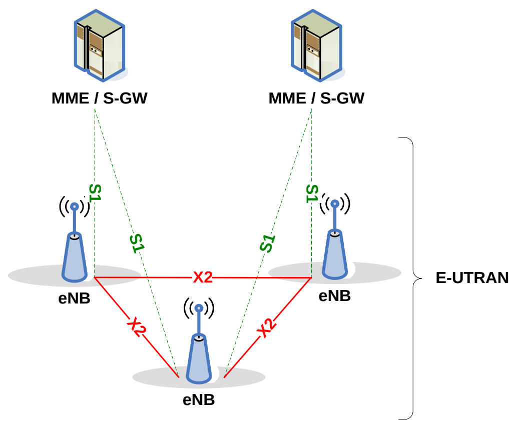

Fig. 4. LTE system architecture.

In LTE system, eNode B provides a wireless interface for UE. The UE which conforms to the interface can communicate with each other even though they produced by different factories. The test suites developed by ETSI are used to check whether the UE conforms to the specification or not.

C. LTE Protocol Stack

The wireless interface protocol stack is divided into user plane and control plane protocol stack which are shown in Fig. 5 and Fig. 6. Physical layer is located on the bottom of protocol stack. It provides all the functions of the bit-stream transporting in the physical medium. Data Link Layer (DLL) includes threes sublayers, Medium Access Control (MAC), Radio Link Control (RLC) and Packet Data Convergence Protocol (PDCP) layer. DLL layer is responsible for transporting, ciphering and so on. The Radio Resource Control (RRC) layer in the eNode B control plane is in charge of dealing with all the interactive signaling between UE and eNode B. Non-access stratum (NAS) is a functional layer in the LTE wireless telecom protocol stacks between the EPC and UE. This layer is used to manage the establishment of communication sessions and maintain continuous communication with UE [13].

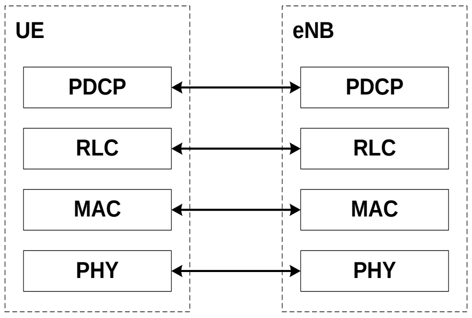

Fig. 5. User plane protocol stack.

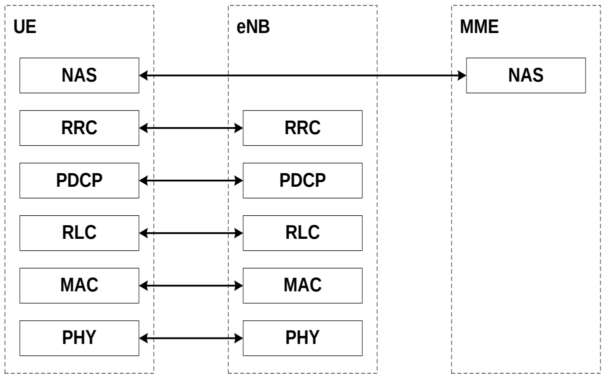

Fig. 6. Control plane protocol stack.

In conclusion, RRC layer is the core part of E-UTRAN. All the signaling sent from UE is handled by RRC layer. It also sends signaling to UE. The physical layer and DLL give service to the RRC layer. Their main task is to ensure the correction of signaling.

IV. The Design of SUT

In this section, the feasible test model will be proposed and stated firstly. Then the conformance test suite which decides the structure of SUT is analyzed in detail. After that, the standard test system shown in Fig. 1 is analyzed from the view of LTE protocol stack. We will try to find out how the SUT can be implemented.

The entire test suites include several parts of UE conformance testing, such as RLC conformance test, MAC conformance test, and the ability test of cell selection in pure E-UTRAN environment. The process to be shown next takes the test case of cell selection as an example. The others are similar to it or even easier.

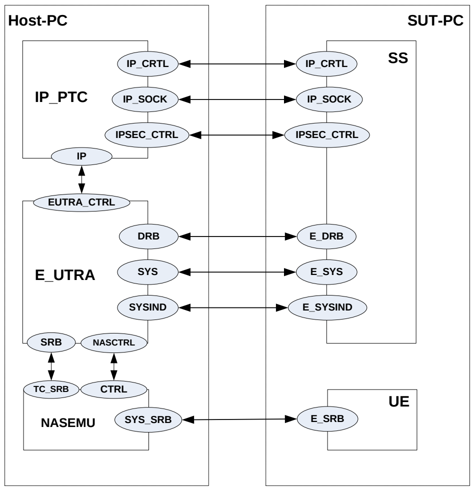

Fig. 7. Test model proposed by this paper.

A. Test Model

The feasible test model proposed by this paper is shown in Fig. 7. Two computers are directly connected by ethernet cable. The test suites run in the Host-PC and SUT code runs in SUT-PC. The SUT is logically consisted of two parts: SS and UE. When Host-PC communicates with SS, some functions of SS are omitted to make the SUT more concise. When Host-PC communicates with UE, the functions of low layers are omitted.

B. The Analyzing of Test suites

The test suites of cell selection in pure E-UTRAN environment can be divided into four parts. The relationships between each part are shown in Fig. 8.

- Main Test Control (MTC): MTC controls the whole process of testing, which includes exception handling, Parallel Test Control (PTC) creating, port mapping and making the final verdict.
- IP_PTC: The IP_PTC configures for UE and SS, including IP configuration, port number configuration and so on.
- E-UTRA: E-UTRA is the core part of the code structure. It is responsible for sending signaling and configuration information to IP_PTC, NASEMU and SS. It also needs to make verdict according to the received message. It mainly communicates with the SS in practice.
- NASEMU: NASEMU is responsible for Handling of ASPs from test cases and RRC PDU from system adaptor. It is responsible for encoding, decoding, ciphered and deciphered for the NAS PDU.

Every part of the test suite as well as SS communicates with each other through ports. All the Ports are responsible for sending and receiving messages. As we can see from the Fig. 8, the E-UTRA has eight ports to communicate with outside.

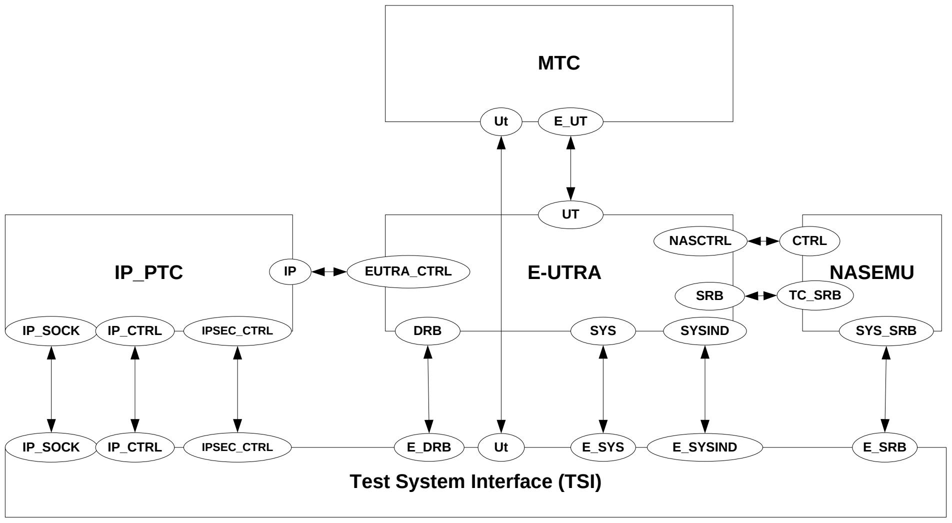

Fig. 8. The structure of 6-1 test group

We need to send the message to the correct port at the right time. If something unexpected happened, the MTC will suspend the testing. For example, before UE selects a cell to camp on, the RRC layer of UE will use the E_SRB port to send RRC connection request message to the NASEMU through SYS_SRB port. NASEMU decodes and deciphers it and then sends it to the SRB port of E-UTRA, E-UTRA calls a function to check whether the message fits to the expected one. If it passes the checking, the test will continue. On the other hand, if the message of RRC connection request is send to the wrong ports, MTC will stop the whole test. For example, once SYS ports received the message, the exception handling starts. E-UTRA tells MTC through the UT port, and MTC will stop the test case.

C. The Analyzing of Test System

The test system shown in Fig. 1 can be described more clearly from the view of layers. The relationship between each layer is shown in Fig. 9 [2]. The TTCN Code and E-UTRAN PTC is a part of the test suite running in the Host-PC. The PDCP, RLC, MAC and PHY are a part of SS. SS can create cell circumstance for UE. E-UTRAN do configuration to low layers through SYS and SYSIND ports. After receiving the signal, SS will respond to E-UTRAN. Only when the response is received by E-UTRAN correctly, the test suite is able to continue.

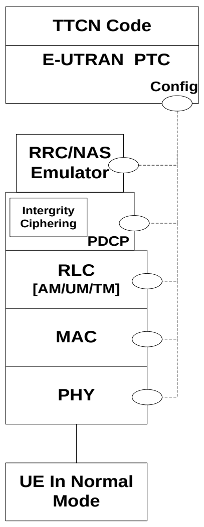

Fig. 9. Test mode for pure environment.

When SS communicates with E-UTRAN, from the view of E-UTRAN, SS is a part of SUT. But the SUT we design is not to implement the real SS. For the purpose to make the test suite run with correct test procedures and functions, it responds the message expected by E-UTRAN. Similarly, when the IP_PTC sends a message to SS, we need to make the SUT return the expected message to IP_PTC.

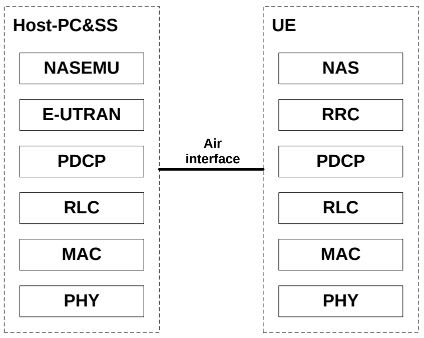

Fig. 10. Protocol stack of test model.

When UE communicates with E-UTRAN, the relationship between Host-PC and SS from the view of protocol stack is shown in the Fig. 10. If UE tries to send RRC connection request to E-UTRAN, after encoded and ciphered by NAS layer, the signaling will be sent to lower layers. Then lower layers package it to a specific frame format. At last, the frame format is send out by the physical layer to the Host-PC. When SS receives the frame format, it is unpackaged by the lower layers. After these steps finish, the message to be decoded and deciphered is received by NASEMU. Then it will be sent to E-UTRAN. If it fits to the expected message, the test suite is able to continue.

In fact, as shown in Fig. 11, the protocol structure consists of two main layers, Radio Network Layer, and Transport Network Layer [14]. E-UTRAN functions are realized in the Radio Network Layer. The Transport Network Layer represents standard transport technology which is used for E-UTRAN.

As mentioned above, the Transport Network Layer is mainly used to transport data and signal. According to the test model of this paper, the function of transporting can be realized by socket. For these reasons, when the communication is between UE and E-UTRAN, the SUT can be designed to replace UE.

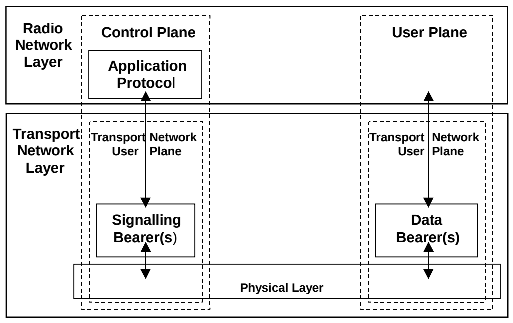

Fig. 11. General protocol model for E-UTRAN interfaces.

According to the test system in Fig. 1, The TTCN-3 code runs on the host system only and no TTCN-3 components are downloaded to system simulator HW. So SS together with UE can be regarded as a black box from the view of Host-PC. What the Host-PC does is to compare received message with expected message. For this reason, the black box can be replaced by a PC which is able to send the expected message to Host-PC for the purpose of making the test suite run with correct test procedures and functions. So the test model can be designed as Fig. 12.

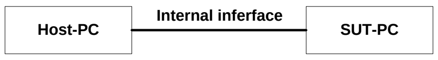

Fig. 12. Test system proposed by this paper.

Two computers are connected by cable. The SUT sends the correct message to Host-PC, all the internal processing in the “black box” is omitted. Above all, the concrete test model proposed by this paper can be designed in detail as Fig. 7.

If the SUT is used to simulate more than one user, every user has an individual port to communicate with Host-PC. Some settings can be done in Adaptor to make Host-PC sent the message to specific port. So each user can communicate with Host-PC independently.

V. The Implementation of SUT

As mentioned in section II, the behaviors of SS and UE are described by TTCN-3 language. In order to achieve the flow of test case, developer and engineer need to develop the TTCN-3 SUT code, Codec and Adaptor. In this section, the implementation of SUT and the develop process of Codec and Adaptor will be stated in detail.

A. TTCN-3 SUT

The main task of SUT is to send and receive messages. As a preparation work, several ports are defined. Every port of SUT is mapped to a specific port of test suite as shown in Fig. 7. Timer is always necessary in order to avoid program deadlock. Some exception conditions, such as unexpected message is received, should be taken into consideration. *Alt* statement is used to deal with these situations. At last, the connection between SUT and test suites should be released.

The brief code of SUT is shown next. The function of “NasEmu” is mentioned. The port of “NasEmu” and some relevant operations are defined in this “NasEmu” header file. Other header files are the definition of the messages. What needs to concern is that if message exchanging happened more than once in the whole process, the relevant part of the code needs to add in order.

```
module SUT{

        import from NasEmu all
        //import from other header files

        type port myPort message {
                inout all;
        }

        type component mtcType {
                port myPort IP_SOCK;
                port myPort IP_CTRL;
                port myPort IPSEC_CTRL;
                port myPort E_DRB;
                port myPort Ut;
                port myPort E_SYS;
                port myPort E_SYSIND;

                timer localtimer := 15.0;
        }

        type component systemType {
                port myPort IP_SOCK;
                port myPort IP_CTRL;
                port myPort IPSEC_CTRL;
                port myPort DRB;
                port myPort Ut;
                port myPort SYS;
                port myPort SYSIND;
        }

        testcase tc() runs on mtcType system systemType {
                map(mtc:IP_SOCK, system:IP_SOCK);
                map(mtc:IP_CTRL, system:IP_CTRL);
                map(mtc:IPSEC_CTRL, system:IPSEC_CTRL);
                map(mtc:E_DRB, system:DRB);
                map(mtc:Ut, system:Ut);
                map(mtc:E_SYS, system:SYS);
                map(mtc:E_SYSIND, system:SYSIND);

    localtimer.start;
alt {
[]    receive(correct message) // correct message is received
{
    localtimer.stop;
    send (expected message);
 }

[]    receive()          //Unexpected message received
{
    localtimer.stop;
    log(“Unexpected message received”);
}

[] localtimer.timeout{ }
}
                unmap(mtc:IP_SOCK, system:IP_SOCK);
                unmap(mtc:IP_CTRL, system:IP_CTRL);
                unmap(mtc:IPSEC_CTRL, system:IPSEC_CTRL);
                unmap(mtc:E_DRB, system:DRB);
                unmap(mtc:Ut, system:Ut);
                unmap(mtc:E_SYS, system:SYS);
                unmap(mtc:E_SYSIND, system:SYSIND);
        }
}
```

B. Codec

Technical specification has some Codec regulations, that is, specific data types need to be encoded and decoded by specific Codec. But in this paper, for the purpose to show the flow path of test suites, common Codec is a proper choice.

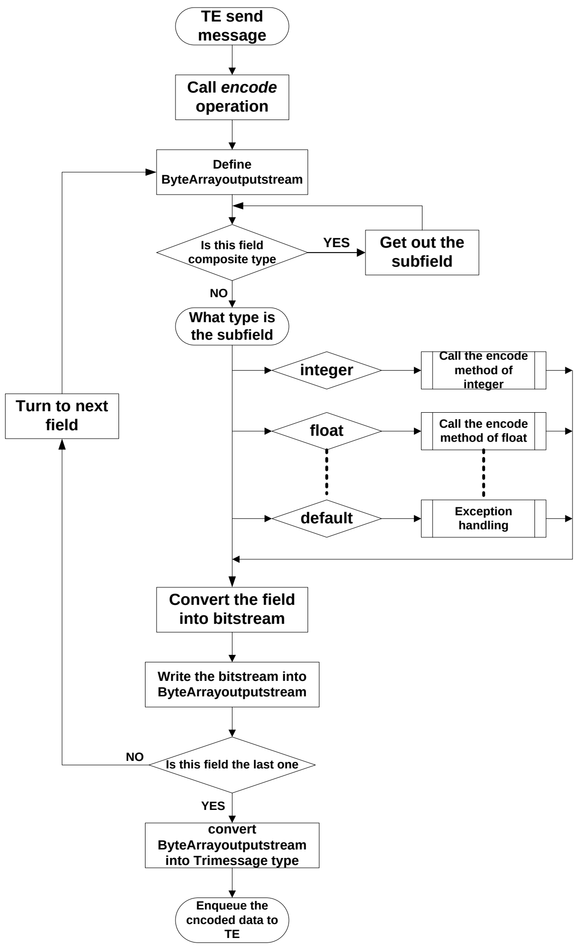
TE send message
Call *encode* operation
Define ByteArrayoutputstream
Is this field composite type
YES
Get out the subfield
NO
What type is the subfield
integer
Call the encode method of integer
float
Call the encode method of float
default
Exception handling
Turn to next field
Convert the field into bitstream
Write the bitstream into ByteArrayoutputstream
NO
Is this field the last one
YES
convert ByteArrayoutputstream into Trimessage type
Enqueue the cncoded data to TE

Fig. 13. The flowchart of encoding.

Common Codec is a Codec which can encode and decode all the TTCN-3 data types including composite type. The programme flowchart of common Codec is shown in Fig. 13. After TE starts to send message, it calls the encode operation. At the beginning of the operation, the *ByteArrayoutstream* is defined which is used to store the result of encoding. Then, the data is verdicted whether it is a composited type or not. If not, the exact type is found out, and the data will be encoded by the relevant method. If it is actually a composite type, the subfield of the data will be get out to be encoded. Of course, if the subfield is still composite type, its subfield must also be get out untill the type is one of the basic types. The encode methods of basic types can be developed freely by developers. But what needs to concern is that all the methods must be based on the framework provided by the technical specification. After the data is converted into bitstream, the bitstream is copied to the *ByteArrayoutstream*. If all the fields of the data have already been encoded, the result needs to be converted into *Trimessage* type which can be recognized by TE. At last, the encoding result will be returned to TE. Once TE receives the result, the encode result will be sent to SUT.

On the other hand, the methods of decoding must highly conform to the encode methods. The whole process of decoding is quite similar to the encode process.

C. Adaptor

All operations used in test case and transport protocol must be implemented in adaptor. Because messages need to be received by special port, the ports defined in test case should be mapped one-to-one with the ports of SUT.

The brief code of Adaptor is shown next. The main task of *triExecuteTestcase* is to get the parameters. Usually, the IDE provides an interface for users to get the configurations. In the part of *trimap*, the socket is used to receive the message. Then the message will be converted into *Trimessge* type before sending to TE. The interface of *trisend* is responsible for getting the encoded message and packet it into a datagram before sending out. At the end of the test suite, *triunmap* will be called to stop listening to ports and close the sender soceket.

```
    triExecuteTestcase(final TriTestCaseId testcase,

final TriPortIdList tsiList){
     //get the remote IP address, port number and local address, port
    //number from the configuration of IDE
     }

triMap(final TriPortId compPortId,
                                        final TriPortId tsiPortId) {
// prepare to be ready to communicate
             rxSocket = new DatagramSocket(localPortNumber);
              txSocket = new DatagramSocket();

// listen on the receiver socket
                        while (mylock){
                        udpReceiverThread = new Thread() {
                        rxSocket.setSoTimeout(100);
                        rxSocket.receive(packet);

final byte[] trimmedMsg = new byte[packet.getLength()];
System.arraycopy(msg, 0, trimmedMsg, 0, packet.getLength());

// convert the received message to Trimessage
final TriMessage rcvMessage = new TriMessageImpl(trimmedMsg);

// enqueue the received message to TE
triEnqueueMsg(tsiPortId, new TriAddressImpl(
               new byte[] {}), compPortId.getComponent(), rcvMessage);

// close the receive socket
        rxSocket.close();}
        }
        }

triunmap(final TriPortId compPortId,
                                        final TriPortId tsiPortId) ){
// stop listening
        Mylock=false;

//close the sender socket
        txSocket.close();
        }

triSend(final TriComponentId componentId,
        final TriPortId tsiPortId, final TriAddress address,
        final TriMessage sendMessage) {
// get the encoded message
        final byte[] mesg = sendMessage.getEncodedMessage();

//get the remote IPAddress
final InetAddress addr = InetAddress.getByName(remoteIPAddress);

// packet the message
final DatagramPacket packet = new DatagramPacket(mesg,
mesg.length, addr, remotePortNumber);
        }
```

In this example, UDP is adopted. Other transport layer protocols such as TCP can also be implemented in the adaptor if necessary.

In conclusion, the message is encoded by Codec and then sent to special port of SUT by adaptor. Once the message arrives, it is sent to TE by adaptor. After the message decoded, other expected message is sent out by SUT. Similarly, Host-PC receives it and makes verdict. If everything goes well, the test case continues.

VI. An Example of Test Procedure

TABLE I: MAIN BEHAVIORS

Step	**Procedure**	**Message sequence**	
		S-U	***Message type***
1	Cause the UE to be switched off at the beginning of a test case	→	UT_SYSTEM_REQ
2		←	UT_COMMON_CNF
3	Cell Configuration	→	p_SYSTEM_CTRL_REQ
4		←	car_CellConfig_CNF
5	Radio Bearer configuration	→	CommonRadioBearerConfig_REQ
6		←	CommonRadioBearerConfig_CNF
7	DRB configuration	→	CommonRadioBearerConfig_REQ
8		←	CommonRadioBearerConfig_CNF

In this section, the process from the beginning of test case to the cell creation is chosen to be the example. Nearly all the test cases in the pure environment and multi-mode environment start with this operation. The main behaviors of the process are shown in Table I.

Before executing the test case, the code of Codec and adaptor should be compiled and loaded to the IDE. It is an important step to make test suites run successfully.

As mentioned in section V, several ports and timer need to be defined. According to the main behavior of the process, the core part of SUT code which shows the first message exchanging is shown below.

```
localtimer.start;

    alt {
```

```
    //received the right messsage
    [] Ut.receive(cs_ UT_SYSTEM_REQ)
       {
          localtimer.stop;
         Ut.send(car_UT_COMMON_CNF);
       }

    //received other messge
    [] Ut.receive()
       {
          localtimer.stop;
          log("Unexpected message received");
       }

    //nothing is received until the ending of the timer
    [] localtimer.timeout{ }
   }
```

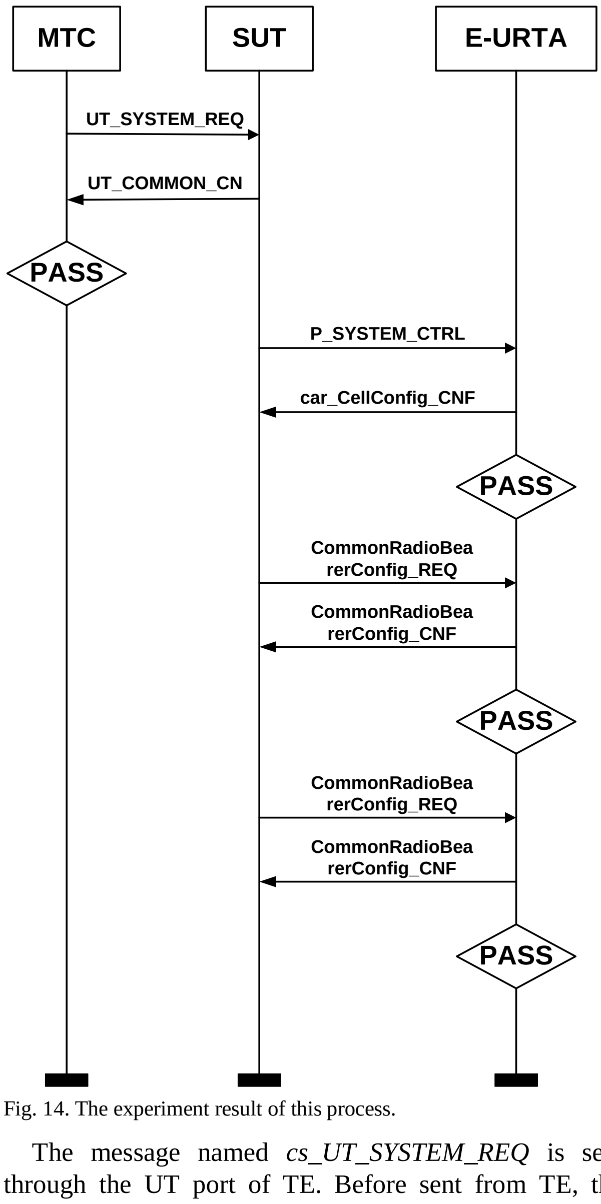
MTC
SUT
E-URTA
UT_SYSTEM_REQ
UT_COMMON_CN
PASS
P_SYSTEM_CTRL
car_CellConfig_CNF
PASS
CommonRadioBearerConfig_REQ
CommonRadioBearerConfig_CNF
PASS
CommonRadioBearerConfig_REQ
CommonRadioBearerConfig_CNF
PASS

Fig. 14. The experiment result of this process.

The message named *cs_UT_SYSTEM_REQ* is sent through the UT port of TE. Before sent from TE, the message is encoded by the codec and TE calls the *trisend* method to send it out. After sending it, TE is waiting for the message *car_UT_COMMON_CNF*. After SUT receives *cs_UT_SYSTEM_REQ*, the message expected by test case is sent from SUT. When test case receives the message named *car_UT_COMMON_CNF*, TE calls the method of *Trienqueue* to enqueue the message to TE. Then the message will be sent to decode. The decoded message will be sent to TE again. If everything goes well, the test case passes. So the test suite is able to go on. The rest part of SUT is similar with above.

The experiment result of this process is illustrated in Fig. 14. The experiment result indicates that the first several steps in this test suite passed. The right message is sent to the right port. In reality, the rest flow of the test suite is similar to it. If the received message matches with the expected one, the step of test will pass and next step continues. If all the steps of the test case pass, the equipment is in accord with the relevant standards.

The experiment result shown in Fig. 14 implies the correction of the method proposed by this paper. Currently, the protocol conformance test suites can only been run in actual test system [3]-[5]. But to build an actual test system is expensive and time consuming. So the cost of development rises. With the method proposed by this paper, the test system is more concise and convenient to be realized.

VII. Conclusions

The main contribution of this paper is that a feasible test model is designed and the SUT of the test model is implemented. In this test model, a computer is used to simulate the SUT to instead UE and SS. Test suites could be run with correct test procedures and functions. In this paper, a typical process of all the test suites is chosen as an example to verify its correction.

In the near future, the RRM conformance test suite may be also published. The method proposed by this paper is also suitable for using. With the widely used of TTCN-3, more and more fields such as Intelligent Transportation System (ITS) and Wimax (802.16) have chosen TTCN as a standard language to develop test suites. Authoritative organizations release more and more TTCN-3 test suites for manufacturers to test their products. So the method of this paper may be more and more useful.

As going on work, the structure of SUT can be smarter. The code efficiency of the common codec can be improved by trying some other encode technique.

References

[1] H. Dong, L. Liu and X. Li, "The Application of TTCN-3 in The Conformance Testing of TD-LTE System," presented at 2nd International Conference on Business Computing and Global Informatization, Shanghai, China, Oct. 12-14, 2012.

[2] 3GPP TS 36.523-3, User Equipment (UE) conformance specification; Part 3: Test suits, Sep. 2012.

[3] F. Chen and G. Zhou, "RRM Conformance testing in TD-LTE system based on TTCN-3," presented at IEEE International Conference on Information and Automation, Yinchuan, China, Aug. 2013.

[4] Q. Li and G. Deng, "Research and implementation of 3G/4G inter-RAT terminal RRM conformance testing system (in Chinese)," *Computer Engineering and Design*, vol. 34, no. 12, pp. 4081-4088, Dec. 2013.

[5] Y. Liang, B. Zhao, C. Gao and J. Sun, "Research on conformance testing for the safety communicaion protocol of high-speed railway control system based on TTCN-3," presented at International Conference on Transportation, Mechanical and Electrical Engineering, Changchun, China, Dec 16-18, 2011.

[6] D. Wang, J. Kuang and W. Tan, "Conformance testing for the car lights system based on AUTOSAR standard," presented at IEEE International Conference on communication Software and Networks, Xi'an, China, May 27-29. 2011.

[7] S. Zhang, H. Li, Y. Xue and X. Wang, "Using TTCN-3 to test SPDY Protocol Interaction Property," presented at IEEE International Computers, Software and Applications Conference Workshops, Västerås, Sweden, July 27-29. 2014.

[8] B. Stepien, L. Peyton, and P. Xiong, "Using TTCN-3 as a Modeling Language for Web Penetration Testing," presented at IEEE International Conference on Industrial Technology, Athens, Greece, March 19-21, 2012.

[9] ETSI, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3, Part 1: TTCN-3 Core notation, ETSI ES 201 873-1, June 2014.

[10] C. Willcock, *et al.*, *An Introduction to TTCN-3*, Chichester, U.K.: Wiley, 2005, ch 12, pp. 204-208.

[11] S. Sesia, I. Toufik and M. Baker, *The UMTS Long Term Evolution From Theory to Parctice*, Beijing, China: Posts and Telecom Press, 2009, pp. 1-3.

[12] 3GPP TS 36.300, E-UTRA and E-UTRAN Overall Description, June 2013.

[13] Y. Wang and Z. Sun, *TD-LTE Principles and System Design*, Beijing, China: Posts and Telecom Press, 2010, pp. 33-35.

[14] 3GPP TS 36.401, E-UTRA Architecture Description, June 2010.

Yin Chen received his B.S. degree in Information and Computing Science from Hefei University of Technology, Anhui province, China. He is currently working toward Master's degree at Hefei University of Technology. His research interests include protocol conformance testing and wireless networks.

Zhizhong Ding received his B.E. degree in Radio Communications from Nanjing University of Aeronautics and Astronautics, Nanjing, China, Master's degree in Circuit and System from Hefei University of Technoloy, Hefei, China, and Ph.D. in Information and Communication Engineering from University of Science and Technology of China. He currently is a Professor with the Department of Communication Engineering and with the Institute of Communications and Information Systems, Hefei University of Technology. His research intersests include wireless communications, network communications and information theory.

Dingliang Wang received his B.E degree in Communication Engineering from Hefei University of Technology, Anhui province, China. He is currently working toward Master's degree at Hefei University of Technology. His research interests include protocol conformance testing and wireless communication.

Teng Huang received his B.E degree in Communication Engineering from Anqing Normal University, Anhui province, China. He is currently working toward Master's degree at Hefei University of Technology. His research interests include protocol conformance testing and OFDM system.