Proceedings of
WInnComm 2019
Wireless Innovation Summit on Wireless Communications Technologies
*15-16 May 2019 * Berlin, Germany*
Hosted by Fraunhofer

Editors:
Lee Pucker, Wireless Innovation Forum
Stephanie Hamill, Wireless Innovation Forum

Sponsors:

Table of Contents

WInnComm Europe 2019 is a presentation only; paper optional event. Papers submitted for the Summit are included in this document. Presentations are available here: https://Europe.WirelessInnovation.Org

CERTIF: CONFORMITY TESTS ON SOFTWARE DEFINED RADIO PLATFORMS

Olivier Kirsch[1], Jean-Philippe Delahaye[2] and Alain Ribault[1]
[1]KEREVAL, Thorigné Fouillard, France
[2]Direction Général de l'Armement Maîtrise de l'Information, Bruz France

ABSTRACT

To ensure interoperability and portability of software defined radio components, the conformity to SDR (Software Defined Radio) standards (including APIs and behavior specifications) is mandatory. Either for the government agency or for the radio platform manufacturers and the waveforms developers, the conformity checking is a great challenge. Due to the huge number of requirements and to ensure reproducibility of the compliance assessment, we have designed a testing methodology and implemented it into the bench CERTIF (Conformance to ESSOR software defined Radio TestIng Framework). Firstly we summarize in this paper the test methodology applied to verify the conformity of a software radio platforms and applications to the SDR standards. Then we list all kind of non-conformity issues that can be detected by the bench CERTIF and we provide examples based on concrete use cases and coming from experience feedbacks on the bench. We will also highlight the importance of test results reproducibility.

1. INTRODUCTION

Since the introduction of the SCA [REF 2] (Software Communication Architecture) standard in the 2000s the rise in power of the Software Defined Radio in the field of military communications has brought to light needs for testing. In the early 2010s, the ESSOR Program Phase 1 has established the ESSOR Architecture that extends the SCA v2.2.2 OE particularly on DSP and FPGA resources and the associated JTRS APIs to fulfill the need of the European tactical radio systems. The paradigm of the ESSOR Architecture [REF 1], which has been recently released by OCCAR (Organisation Conjointe de Coopération en matière d'Armement / Organization for Joint Armament Co-operation), relies on several of the following concepts:

- a component-based architecture, a PIM/PSM approach for specification
- a Waveform/Platform separation of concerns,
- and on the use of the CORBA ORB for interactions between components running on GPP (as it is based on SCA 2.2.2).

It answers to a main goal to reach portability of the waveform onto SDR platform. This goal is common at least between the SDR standards publicly available and produced by JTNC, ESSOR and WinnF [Ref 3]. Last but not least statement is that since 2 decades of SDR standardization, standard evolution becomes an important dimension to take into account especially for stakeholders involved in SDR procurement program.

Consequently, requirements for a SDR conformance testing capability are among the following:

- Be able to address of SDR standard evolution.
- To take advantage from the PIM/PSM Standard approach by separating conformance analysis from the testing implementation.
- Define conformance testing on portability assessment meaning defining conformance checking in regards of the standard requirements specifying WF/PF interactions.

The test methodology presented in this paper answer to these high-level requirements. Especially the test design process is independent to any test tool implementation. It is based on the use of database and compliance test repository agnostic from any language. It is a valuable approach standard eco system with multiple actors as the test process is also based on standard like UML or OCL.

The rest of the paper is organized as follow. The next part details the test methodology and how is answer to the design requirement introduced before. The part 3 considers test the different test strategies to address the non-conformity detection.

The figure below lists the contributions of SCA concepts and shows the needs to assess the compliance to SCA concepts in order to take advantages of these contributions:

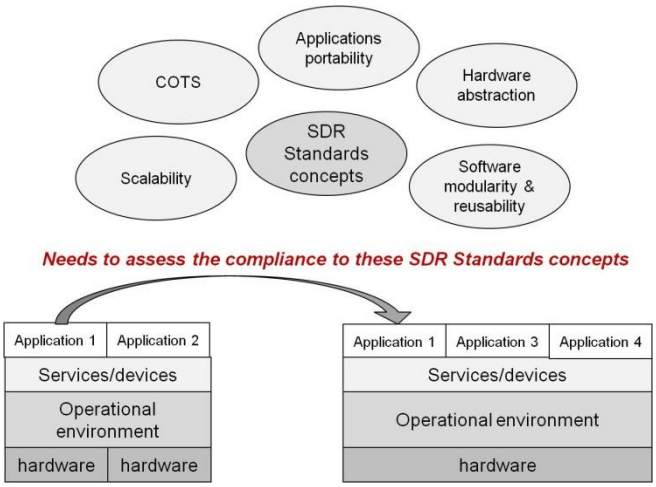


Figure 1 Concepts to assess

2. TESTING METHODOLOGY

The first step to design the testing methodology has been to define the nature of the system under test.

Firstly, we assumed that ESSOR software radio platform is a physical equipment with GPP, DSP and FPGA processing resources running a compliant ESSOR operating environment with a set of implemented Radio Device and Radio Service. Therefore, we chose to perform dynamic tests calling platform interfaces as compliance analysis method.

Secondly we assumed that a compliant application (or waveform) is a set of source code files that compile including IDL, C/C++VHDL and XML files in accordance to waveform design methodology defined by ESSOR [REF 1].

As the potential porting stage of this set of source code could change the content of the system under test, we chose to perform static analysis tests as compliance analysis method. This analysis is performed on the "golden source" which is the portable part of the application source code.

2.1. Test design process overview

The test design process follows the good practices promoted by ISTQB (International Software Testing Qualifications Board) [REF 4] particularly regarding the MBT (Model Based Testing) [REF 5] and automation design.

This process consists in four phases depicted in the following illustration:

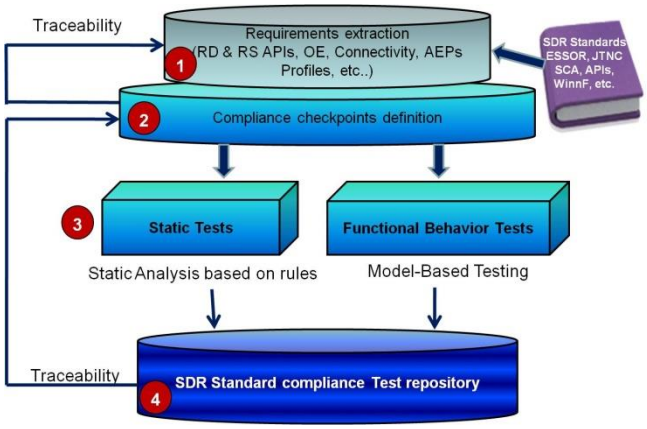


Figure 2 Test design process overview

The first phase is the extraction of the requirements from the specifications. Basically, each extracted requirement specifies either the nominal behavior or the processing error of one function of an interface. All of the extracted requirements are grouped by ESSOR device/service APIs and interfaces. It also includes JTRS APIs.

Requierement		RCC (Requirement Compliance Checkpoint)		
Requirement Identifier	Requirement Text	RCC Identifier	RCC Applicability	RCC Description
JTRS_AD_PROVIDE_CREATE_TONE	The createTone operation provides the capability of creating a tone or beep with the specified profile, for future use by the device user. - Synopsis: unsigned short createTone(in ToneProfileType toneProfile) raises(InvalidToneProfile); - Return Value: unsigned short An identification number associated with the tone/beep. 0 – 64k	-	-	-
JTRS_AD_PROVIDE_CREATE_TONE		JTRS_AD_PROVIDE_CREATE_TONE_SUCCESS_001	Platform	* Success case * Check the id is properly returned * Simple Tone creation * Check the tone presence
JTRS_AD_PROVIDE_CREATE_TONE		JTRS_AD_PROVIDE_CREATE_TONE_SUCCESS_002	Platform	* Success case * Check the id is properly returned * Complex Tone creation * Check the tone presence
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile	InvalidToneProfile (see A.5.3.1) A CORBA exception is raised when the tone/beep cannot be generated due to invalid attributes in toneProfileType structure.	-	-	-
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile		JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile_001	Platform	* ComplexToneProfile invalid * Check an exception: InvalidToneProfile is raised
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile		JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile_002	Platform	* SimpleToneProfile invalid * Check an exception: InvalidToneProfile is raised

Figure 3 Compliance checkpoint sample

The second phase is to define compliance checkpoints for each requirement. This step allows to define the conformance criteria independently from the test definition itself. This step is also a key point to ensure an optimum test coverage. The compliance checkpoints have to describe the test objectives to fulfill for each requirement and they have to cover each possible behavior.
Here is an example on the Figure 3 above based on the "createTone()" function member of the interface "AudibleAlertsandAlarms" from the JTRS and ESSOR Audio device.
In this example, two requirements are extracted from the specification. One specifies the nominal behavior and the other specifies the processing error behavior. For each of these requirements two compliance checkpoints are defined to cover the two kinds of tone specified (simple and complex).

Since the compliance checkpoints define the test objectives, the tests designed to check the conformity of the device or service has to match these checkpoints.
This is the goal of the third phase: the design of the tests. We will focus in this paper on the description of the design of the dynamic tests on radio platforms. Our approach to produce the test suite applied to SDR platform conformance testing is based on a MBT approach.
At last, the fourth phase is the generation and the publication of the tests, the compliance checkpoints and the requirements into a SDR Standard compliance test repository. This repository is a database linked to test manager software used to perform the test campaigns.
The following sections will detailed each of these steps.

2.2. Requirements extraction phase

The extraction of the requirements from the software defined radio standards follows the good practices promoted by IREB (International Requirements Engineering Board) [REF 6]. Following these recommendations a requirement shall be, among other things, exact, verifiable, unique and non-ambiguous.
The ESSOR RD and RS APIs define interfaces and associated behaviors of each device or service on a Software-Defined Radio platform. Applying IREB recommendations to these specifications, we have extracted requirements describing each nominal behavior of a function of an interface and each error management case of this function. In addition to the description, the attributes of a requirement are defined by the following elements:

- Nature: functional, non-functional (performance).
- Type: Mandatory or Optional (API Extensions).
- Applicability: platform, application or both
- Coverage of the requirement by another requirement.

All these requirements are grouped by device or service and by interface following the structure of the ESSOR specification [REF 1]. They form the basis of the test database. As the portability assessment is one of the goal of the conformity testing, most important requirement extraction is oriented on features that implies interaction between Platform and Waveform. It results that most part of the requirements extracted from the specification are split into two requirements. One of the two is applicable to platform and the other to the application.

2.3. Requirements analysis phase

After the extraction of all the requirements from the specification, the analysis phase consists in the definition of compliance checkpoints.
A compliance checkpoint describes a test objective in a precise and unambiguous manner with the following elements:

- The nature of the test objective: is this a success (or nominal) case, an error case or a case of exception rising?
- The description of the verification(s) to do for this test objective.
- The description of the considered conditions or alternatives for this test objective (if applicable).

Figure below describes the methodology for a compliance checkpoint definition

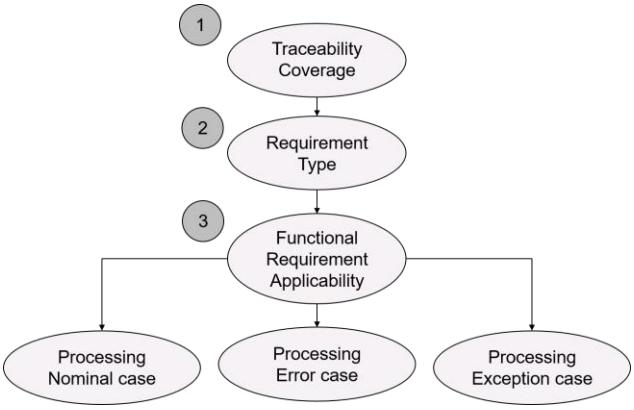


Figure 4 Methodology of a compliance checkpoint definition

1 If the requirement is already covered by other requirements, then there is no declination in compliance checkpoint for this targeted requirement.

2 According to its requirements type:
A requirement carries the testability information. A functional requirement will then be declined into compliance checkpoints based on the semantic analysis

3 A functional requirement is processed according to three identified general cases:

- **Nominal case** (success case): According to the considered alternative covering the success case one or more test objectives will be defined
- **Case of errors**: An operation can return several error codes (that is not an exception case). Each error case is a test objective, thus a compliance checkpoint by error case will be defined
- **Case of exceptions**: A requirement is about an exception type for a given operation. The exception can contain several cases. A compliance checkpoint will be created for each exception.

The test objectives described by the compliance checkpoints are specific to the application or to the platform and form the conformity criteria.

2.4. Behavior modeling phase

The MBT methodology adopted for this project is based on the domain artifacts illustrated in the specifications (the artifact concept is explained in the Model-Based Tester Extension Syllabus [REF 5]). The models represent a test view of the specification and the domain elements described in the specification.

The CERTIF approach relies on the use of a test model. This test model is written using a subset of the UML and OCL language, called UML4MBT and OCL4MBT. The design of the model with these languages ensures the independence of the model with the programming language used to implement the tests (e.g. C++, JAVA, etc.). More precisely, class diagrams describe the points of control (operation calls), the observations (checks) and the objects that constitute the system under test (SUT). The dynamic behavior of the system is expressed within OCL constraints, applied as pre and post conditions on the operations of the class diagram.

The figure 5 summarizes the different artefacts to be developed for the model. The test models are simply based on the Software Radio domain knowledge, using the specifications terminology, (e.g. ESSOR Architecture, JTNC SCA, WInnF, APIs, etc.).

A specific document format including the requirements and the compliance checkpoints can be imported into MBT software and result to the automatic creation of a class for each interface. Then, based on the domain knowledge the additional objects used for the test generation are conceived by the Test Designer to obtain the structural view of the system. Bear in mind that a test model is not a system model, although it could be inspired from it, as done in this example. Based on best practices in MBT, the test model should simply include a minimal set of objects sufficient to cover the tested perimeter and abstract enough to capture the equivalence classes of the test data.

2.5. Tests generation phase

The MBT software generates the tests pool thanks to the model designed on one side and to the tests objectives derived from the compliance checkpoints which are inserted into the model in the form of tags on the other side. It checks also that all the tests objectives are attainable in the model.

As the model describes the whole behavior of the device or service APIs the MBT software is able to generate tests fully independent from one to another. The self-sufficiency of the tests between each other is an important point for running the test campaign. Below is the example of a test for the “startTone” function of the Audio Device API.

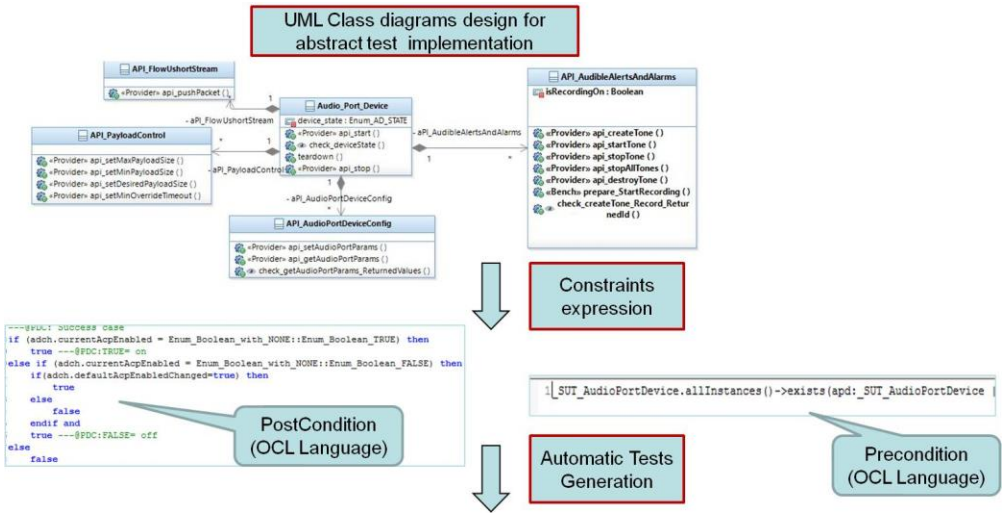


Figure 5 Class diagram and OCL constraints

As it is shown in the Figure 6 below, you can see the test generated consists in three parts.

- The set-up part in which the device is put in the appropriate state.
- The test body in which the test procedure is conducted.
- The Tear Down part in which the device is put to the initial state

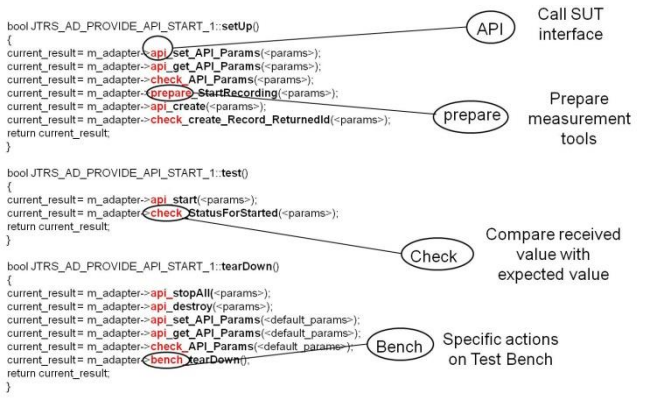


Figure 6 Test generation sample

The generation tool ensures that each compliance checkpoint integrated into the model is covered by at least one test and the completeness of the model is also verified during this process.

2.6. Tests publication phase

After the testing generation process phase, a pool of abstract tests is available to cover all the compliance checkpoints defined in the second phase. These tests can be published into several languages and an adaptation layer shall be developed to realize the different actions of the tests; they could be SUT calls, check function, measurement tool calls or specific actions on the bench. The example below is the publication in C++ language of one of the test of the "startTone" function of the Audio Device.

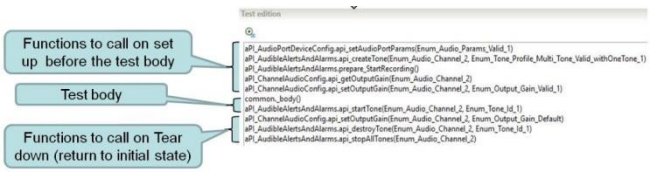


Figure 7 C++ test publication sample

The MBT software tool includes also a module to publish the test cases, the compliance checkpoints and the requirements into the database of a test management software. This feature allows to update automatically the test repository if a model evolves in case of fixing or specification update.

2.7. Test design process results phase

The figure below summarizes the design process cycle:

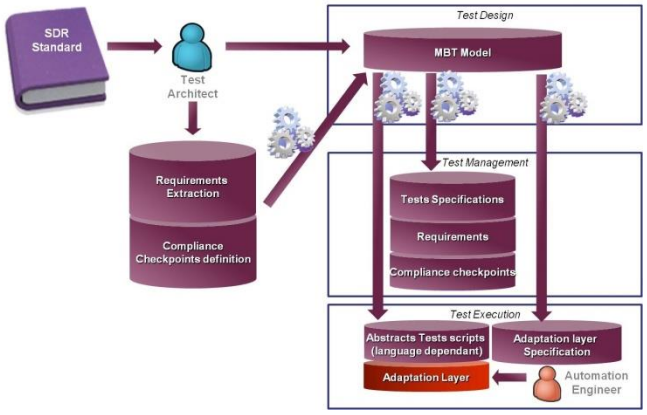


Figure 8 Design process cycle

The main advantages of this methodology are:

- Independence of the model from the target.
- Coverage of the requirements and completeness of the approach.
- Maintenance of the test repository is easier
- Definition of conformance criteria independently from the test definition itself.

The main drawbacks are:

- The initial cost of the modeling step.
- A Model Based testing expertise is required.

3. NON CONFORMITY DETECTION

The defects detected by the bench can take different forms. Indeed, even if the IDL interface of a device defined in the SDR related standards is strictly implemented on the radio platform under test, the behavior of the device when this interface is called can be non-compliant with the specification.

For example if the call to the "startTone" function of the API Audio Port Device defined by ESSOR Architecture RD API or JTRS API is successful (no exception returned) but the platform under test emits no sound, we consider that is a non-conformity although the signature of the function is properly implemented. An another example, on the IP Service API defined by ESSOR [REF 1], if the call to the "addRoute" function is successful, the route table returned by the "readRouteTable" function well contains the route added previously but the call to the "pushpacket" function with the new IP address added is not received by the recipient, this behavior is also a non conformity.

The advantage of the testing method implemented in the bench is to detect also these non-conformities. To illustrate the nature of the different non-conformities to be detected, we will take as example the API Audio Port Device defined by the JTRS standard. The role of this Device is to provide the ability to control alert and alarm tones and to notify the device user (e.g. the application) of a Push-To-Talk signal. The following sections detail some of the non-conformity cases taken into account in the bench.

3.1. Not implemented Interface

The concept of "not implemented Interface" covers two cases.

The first case is the lack of the interface. This case is easily detected by the bench at the connection step of the ports with the System under Test and the user will see a message like the following:

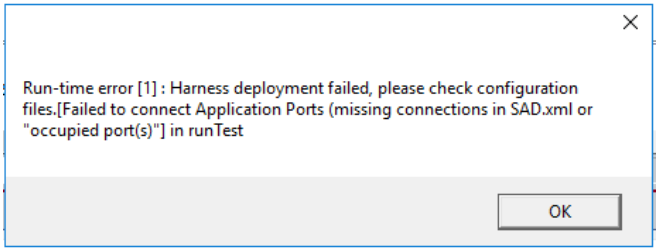


Figure 9 Error Message in case of lack of the port

and the test result is blocked.

Test: Test Name	Status
ESSOR_AD_PROVIDE_GET_ACP_DELAY_ENABLED_CH1...	Blocked

Figure 10 Test result in case of lack of the port

The second case is an empty implementation of the interface. This means that the port exists; it is derived from the valid IDL interface but there is no implementation of the expected processing inside the functions defined by the interface. In other word, the implementation of the interface is an empty shell. To check the capabilities of the bench to detect this defect we have created a "mutant" of an Audio Device that implements an empty version of the interface "AudibleAlertsAndAlarms".

When we launch the conformity test on this mutated Audio device, here are the results we obtain:

Test: Test Name	Status
JTRS_AD_PROVIDE_CREATE_TONE_1	Failed
JTRS_AD_PROVIDE_CREATE_TONE_2	Failed
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile_1	Failed
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile_2	Failed
JTRS_AD_PROVIDE_DESTROY_TONE_1	Failed
JTRS_AD_PROVIDE_DESTROY_TONE_EXCEPTION_InvalidToneId_1	Failed
JTRS_AD_PROVIDE_START_TONE_1	Failed
JTRS_AD_PROVIDE_START_TONE_EXCEPTION_InvalidToneId_1	Failed
JTRS_AD_PROVIDE_STOP_ALL_TONES_1	Failed
JTRS_AD_PROVIDE_STOP_TONE_1	Failed
JTRS_AD_PROVIDE_STOP_TONE_EXCEPTION_InvalidToneId_1	Failed

Figure 11 C++ Not Implemented interface results

All the tests on this interface report a result KO. By studying the test logs we can see that the tests on the exception management failed because no exception was raised by the empty implementation which is easy to check. However, the non-conformity of the nominal behavior of these functions is detected thanks to the test strategy chosen. For example, the test of the "createTone" failed because it calls also the startTone function and it captures and checks the audio signal. In the same way, the test of the "destroyTone" failed because it calls the "startTone" after and it checks that an exception "invalidToneId" raised. Indeed if we had made the choice to call the "destroyTone" function and simply check that no exception raised, the test result would be passed whereas the implementation of the interface was empty.

The lesson that we can learn from these results is that the bench is able to detect empty interface implementation.

3.2. Wrong interface cases

By "Wrong interface" cases, we talk about wrong APIs signatures. This means that the manufacturer does not comply with the interfaces defined by the specifications.

To check the capabilities of the bench to detect this king of non-conformity we have created another "mutant" of an Audio Device which implements a wrong version of the interface "AudibleAlertsAndAlarms". As you can see below we have added members to the "InvalidToneProfile" exception structure and to the "ComplexToneProfile" structure.

```
interface AudibleAlertsAndAlarms
  {
    exception InvalidToneProfile
    {
            boolean complexTone;
            boolean simpleTone;
            short MutantShortValue;
            boolean multiTone;
            string msg;
    };

    struct ComplexToneProfile
    {
            boolean MutantBoolValue;
            JTRS::ShortSequence toneSamples;
             Unsigned short numberOfRepeats;
    };
```

We have also changed the signature of the function StartTone below

```
void startTone( in unsigned short toneId )
        raises (InvalidToneId);
```

by adding a parameter as following:

```
void startTone( in unsigned short toneId, in unsigned long
MutantCharValue )
        raises (InvalidToneProfile);
```

and deleted the function destroyTone.

```
// void destroyTone( in unsigned short toneId )
//      raises (InvalidToneId);
```

Therefore, the IDL interface defined here is not compliant with either ESSOR Architecture RD API or JTRS API specifications.

When we launch the conformity test on this mutated Audio device here are the results we obtain:

Test: Test Name	Status
JTRS_AD_PROVIDE_CREATE_TONE_1	Failed
JTRS_AD_PROVIDE_CREATE_TONE_2	Failed
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile_1	Failed
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile_2	Failed
JTRS_AD_PROVIDE_DESTROY_TONE_1	Failed
JTRS_AD_PROVIDE_START_TONE_1	Failed
JTRS_AD_PROVIDE_STOP_ALL_TONES_1	Inconclusive
JTRS_AD_PROVIDE_STOP_TONE_1	Inconclusive

Figure 12 C++ Wrong interface results

All of the tests on this interface report a result KO or "Inconclusive" (inconclusive means that the Setup or the Teardown step has failed). By studying the test logs we can observe; as we can expect; that the tests of "StartTone" and "DestroyTone" functions failed because a CORBA exception is raised on the call. We detect also the wrong signature of the "InvalidToneProfile" exception by receiving a CORBA exception. However we can observe also that the tests on the "CreateTone" function failed because of the call of the "StartTone" function to check the behavior of CreateTone. In the same way the tests of "StopTone" and "StopAllTone" raised an inconclusive status because of the call of the "StartTone" during the Setup sequence.
This sample of wrong interface highlights the capacity of the bench to detect bad implementation of the interfaces defined in the standard.

3.3. Non conform behavior detection cases

To ensure conformity with the API specification we want to determine that the behavior of the radio platform under test respects the standard. To validate our test strategy and verify that the bench detects a potential improper behavior of the device we have created another mutated Audio Device. This mutant implements the right interfaces but it has two behavioral issues. It does not raised exception on the call of the "startTone" function if the Tone ID is unknown and it does not perform the deletion of the Tones on the call of the "destroyTone" function. The diagram below summarizes the behavior of the "startTone" function:

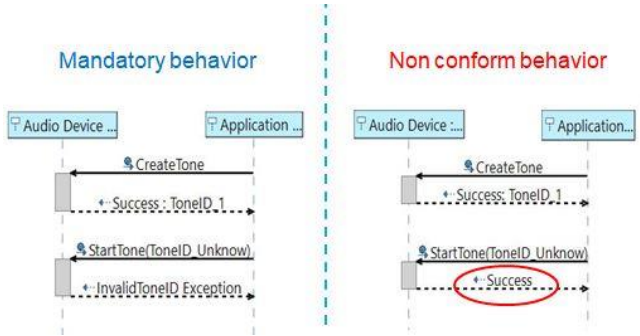
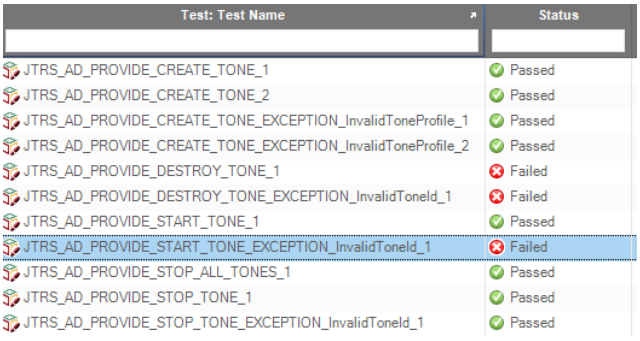


Figure 13 C++ Non conform behavior of startTone.

These two defects are interesting because the first one could hide the second one.
To check the capabilities of the bench we launch the tests covering the requirements of the "AudibleAlertsAndAlarms" interface and we obtain the following results:



Test: Test Name	Status
JTRS_AD_PROVIDE_CREATE_TONE_1	Passed
JTRS_AD_PROVIDE_CREATE_TONE_2	Passed
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile_1	Passed
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile_2	Passed
JTRS_AD_PROVIDE_DESTROY_TONE_1	Failed
JTRS_AD_PROVIDE_DESTROY_TONE_EXCEPTION_InvalidToneId_1	Failed
JTRS_AD_PROVIDE_START_TONE_1	Passed
JTRS_AD_PROVIDE_START_TONE_EXCEPTION_InvalidToneId_1	Failed
JTRS_AD_PROVIDE_STOP_ALL_TONES_1	Passed
JTRS_AD_PROVIDE_STOP_TONE_1	Passed
JTRS_AD_PROVIDE_STOP_TONE_EXCEPTION_InvalidToneId_1	Passed

Figure 14 C++ Non conform behavior detection results

Only three tests over eleven failed. By studying the test logs we see that the two behavioral defects are correctly detected. Indeed, we detect easily that the “StartTone” function does not return an exception when the tone ID is invalid but also that the “DestroyTone” function does not really delete the asked Tone ID. We capture this defect thanks to the strategy of test generation that adds a check after the call of the “DestroyTone” function. This check is a call to the “StartTone” function with the ID of the tone that should be deleted. As this call does not return the "InvalidToneID” exception the test failed.

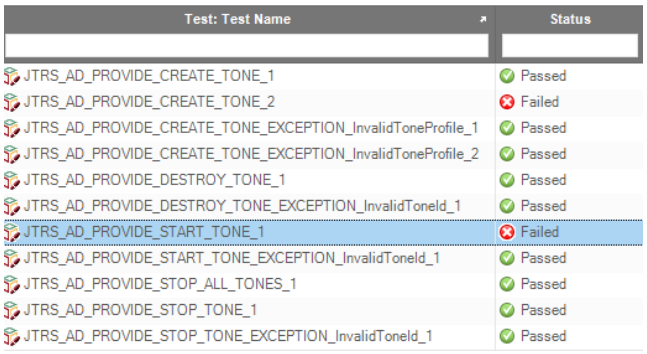
This example shows the advantage of the behavior modeling applied to the tests to detect more tricky defects than failure or wrong interface implementation

3.4. Non conform data processing detection cases

Another scope of the behavior analysis is the data processing. In the previous example, we have checked that the tests are able to detect improper logic behavior. Now we want to verify that the data processing features of the API are compliant with the specifications.

To simulate this kind of defect we have created another mutated Audio Device with the following defect. On the call of the “CreateTone” function for a complex tone, this defect device adds the value 5 to each element of the tone samples sequence sent. The effect of this behavior will be the emission of an audio signal that does not match with the audio sequence configured.

One more time we launch the tests covering the requirements of the “AudibleAlertsAndAlarms” interface and we obtain the following results:



Test: Test Name	Status
JTRS_AD_PROVIDE_CREATE_TONE_1	Passed
JTRS_AD_PROVIDE_CREATE_TONE_2	Failed
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile_1	Passed
JTRS_AD_PROVIDE_CREATE_TONE_EXCEPTION_InvalidToneProfile_2	Passed
JTRS_AD_PROVIDE_DESTROY_TONE_1	Passed
JTRS_AD_PROVIDE_DESTROY_TONE_EXCEPTION_InvalidToneId_1	Passed
JTRS_AD_PROVIDE_START_TONE_1	Failed
JTRS_AD_PROVIDE_START_TONE_EXCEPTION_InvalidToneId_1	Passed
JTRS_AD_PROVIDE_STOP_ALL_TONES_1	Passed
JTRS_AD_PROVIDE_STOP_TONE_1	Passed
JTRS_AD_PROVIDE_STOP_TONE_EXCEPTION_InvalidToneId_1	Passed

Figure 15 C++ Non conform data processing detection results

The two failed tests are related to the “StartTone” and “CreateTone” functions. The tests of these functions capture the audio signal emitted by the radio platform and compare it with the tone samples sequence configured. We can see in this example that one of the two tests of the “createTone” is failed. That is the one which checks the creation of a complex tone and the signal comparison failed because it is outside the tolerance range. The other one checks the creation of a simple tone and in that case the comparison succeeds because no offset is applied to the audio structure configured.

This kind of non-conformity is more difficult to detect and could be interpreted as a performance test instead of functional test if the measuring tolerance is too low. However the use of measurement tools and the check of data processing is clearly a good way for detecting functional defects.

3.5. Tests of boundaries values cases

The last case we explore is the check of the boundaries values. Indeed several parameters are defined in the SDR standards such as ESSOR and JTRS and they are often associated to a valid range. The check of these ranges shall be part of the conformity verification. To validate the detection of non-compliant range with the specification we have created a mutated Audio Device which do not respect the range values of the parameters “MaxPayloadSize” and “MinPayloadSize”. These parameters belong to the Audio Sample Stream Extension of the Audio Device and they define the minimum and maximum size of a packet received or transmitted through the “pushpacket” function.

In the JTRS specification the range of these parameters is defined respectively from 0 to 512 for the “MinPayloadSize” and from 1 to 16383 for the “MaxPayloadSize”“. The mutated Audio device we created is configured to accept respectively a range from 50 to 512 for the “MinPayloadSize” and from 1 to 12500 for the “MaxPayloadSize”. We launch the tests covering the requirements of the Audio Sample Stream Extension and we obtain the following results:

Name	Status
[1]JTRS_AD_PK_PROVIDE_GET_MAX_PAYLOAD_SIZE_1_GPP	Passed
[1]JTRS_AD_PK_PROVIDE_SET_MIN_PAYLOAD_SIZE_1_MIN_GPP	Failed
[1]JTRS_AD_PK_PROVIDE_SET_MIN_PAYLOAD_SIZE_1_MEDIAN_GPP	Passed
[1]JTRS_AD_PK_PROVIDE_SET_MIN_PAYLOAD_SIZE_1_MAX_GPP	Passed
[1]JTRS_AD_PK_PROVIDE_PUSH_PACKET_1_GPP	Passed
[1]JTRS_AD_PK_PROVIDE_SET_MIN_PAYLOAD_SIZE_EXCEPTION_InvalidParameter_1_GPP	Passed
[1]JTRS_AD_PK_PROVIDE_SET_MIN_OVERRIDE_TIMEOUT_EXCEPTION_InvalidParameter_1_...	Passed
[1]JTRS_AD_PK_PROVIDE_SET_MAX_PAYLOAD_SIZE_1_MIN_GPP	Passed
[1]JTRS_AD_PK_PROVIDE_SET_MAX_PAYLOAD_SIZE_1_MEDIAN_GPP	Passed
[1]JTRS_AD_PK_PROVIDE_SET_MAX_PAYLOAD_SIZE_1_MAX_GPP	Failed
[1]JTRS_AD_PK_PROVIDE_PUSH_PACKET_EXCEPTION_UnableToComplete_1_GPP	Passed
[1]JTRS_AD_PK_PROVIDE_GET_MIN_PAYLOAD_SIZE_1_GPP	Passed
[1]JTRS_AD_PK_PROVIDE_SET_MAX_PAYLOAD_SIZE_EXCEPTION_InvalidParameter_1_GPP	Passed

Figure 16 C++ Boundaries values tests results

As shown in figure 16, such test cases are duplicated and suffixed by “Min”, “Median” or “Max”. These test cases are generated by the test bench for each parameterized test in order to check the lower bound, the upper bound and a median value. This feature allows us to verify the range supported by the device and also to test the return of exception for values out of bound. In our example, the two failed tests allow to point out to the tester that the lower bound of the “MinPayloadSize” and the upper bound of the “MaxPayloadSize” are not compliant with the ESSOR Architecture specification.
The boundaries values tests ensure us both the conformity to the SDR standard and the validity of the values provided by the manufacturer.

4. EFFICIENCY OF THE TEST BENCH

By applying the test strategy designed for the test bench, we are able to cover directly 86 % of the requirements extracted from the ESSOR Architecture. The remaining 14 % relates to internal behaviors, to hardware exceptions that could be tested only by too invasive methods or that have no impact on waveform portability.
Thanks to the test bench architecture, 96 % of the tests are fully automated and the results as well as the logs for the analysis are available in a centralized database.

5. CONCLUSION AND PERSPECTIVES

Using the test strategy and design described here, we are able to cover a wide range of non-conformities. Indeed, we detect interface implementation defects; device or service behavior defects and we check also conformity of the technical bounds declared by the manufacturer. Moreover, the software architecture of the test bench and the modeling of the APIs facilitate the evolution of the tests with the SDR standard evolution. The maintenance and/or the rework of the test database are also simplified.

The definition and design of a testing methodology and the development of corresponding tools set based on testing practices well recognized by the test industry, by using standard technologies like MBT, UML, OCL languages, this work represents a significant step in the emergence of solutions in conformity assessment in the standardized SDR domain. In particular, by addressing the compliancy testing of radio devices APIs and Radio Services APIs on SDR Platform, it fills a gap for portability evaluation. This work will help to evaluate standard conformity of the Radio product for example in the French National Military SDR program CONTACT and leverage the French investment in the ESSOR technologies.

The main subject not covered at this time by the test bench is related to the performance measurements. Indeed the SDR standards such as ESSOR or JTRS define performance criteria (e.g. “Worst Case Command Execution Time” for each Audio Device features). There are no limits defined in the specifications for these criteria but the measurement of these one can be important to evaluate the conformity of the radio platform under test. Therefore, we are carrying studies on this subject.

6. ACKNOWLEDGMENT

This work was performed under DGA contract with the participation of the SMARTESTING (Besançon, France) company on the MBT part, the DIGIDIA (La Chapelle des Fougeretz, France) company on the test device implementation part and the support of Frédéric Leroy (ENSTA Bretagne, Brest, France) for his SCA Expertise. The author would like to acknowledge these contributions and the support of DGA-Mi.

7. REFERENCES

[1] ESSOR Architecture Specification:
http://www.occar.int/programmes/essor#expert-area

[2] JTNC SCA and APIs standards:
https://www.public.navy.mil/jtnc/

[3] Wireless Innovation Forum, SDS Committee Public Files, Reports and Specifications:
https://sds.wirelessinnovation.org/specifications-and-recommendations

[4] ISTQB homepage:
https://www.istqb.org/

[5] ISTQB Model Based Testing extension:
https://www.istqb.org/certification-path-root/model-based-tester-extension/model-tester-extension-contents.html

[6] IREB Homepage: https://www.ireb.org/en

An Approach for solving Real-time and Synchronization Issues in heterogeneous Multi-Processor Software Defined Systems

Peter Troll, Dr. Boyd Buchin, Dr. Khaled Fazel, Rohde & Schwarz, and Dr. Marc Adrat, Fraunhofer FKIE

ABSTRACT

Real-time and synchronization issues have been subject to deliberation – and a source of potential confusion – since the invention of computers and their application in technical systems. They also are core issues of Software Defined Systems (SDS) and Software Defined Radio (SDR) as they are distributed real-time systems requiring a precise measurement of time and time-exact execution of commands.

Taking into consideration that waveform portability is one of the primary objectives of SDR, the use of universal, simple and easy-to-use concepts is paramount which allow the provision of waveform agnostic Application Programming Interfaces (APIs) by the host environment. Waveform specific solutions can then be designed employing the APIs and the universal concepts they encapsulate.

Due to further objectives like scalability and broad applicability in various fields, numerous standards have evolved over the years, that address real-time and synchronization issues in the SDR-ecosystem, e.g. IEEE/OMG POSIX, SCA, the JTRS/JTNC standards and the WINNF specifications. They allow for diverse approaches to synchronization and in some cases orthogonal solutions, e.g. "Absolute Time" vs. "Relative Time" within the WINNF Transceiver Facility PIM specification.

In this contribution, we will sketch how through the systematic combination of well-established concepts from these standards a comprehensive – but nevertheless simple – strategy to support real-time and synchronization issues in SDS is possible. The strategy is applicable both to SCA as well as to non-SCA host environments.

To give an example that relates to practice, we will exemplify the application of the strategy to a common hardware architecture that includes an FPGA and a DSP or GPP as computational elements (CE) and we will look into the specific real-time aspects of the different types of CEs.

The central ideas the strategy is based on are:

- Consequent application of the concept of a "system-wide monotonic clock".
- Utilisation of the real-time capabilities of FPGAs in combination with APIs that allow real-time capable implementations, like the JTRS Modem Hardware Abstraction Layer (MHAL) on Chip Bus (MOCB) API.
- Fostering waveform portability by provision of a "lean platform" that features a clear separation between universal functionality of the host environment and waveform specific functionality in the applications.

This paper is about synchronization in SDS in general, with focus on SDR system's core challenge of how the host environment can enable an application to synchronize on the air interface.

1. INTRODUCTION

The challenges arising with distributed real-time SDR systems have been addressed by the various specifications and standards from their respective point of view.

The **Software Communications Architecture** (SCA) [1a, 1b] provides an architecture framework for SDR technology distributed-computing communication systems. The SCA integrates real-time support by including POSIX [3a, 3b], particularly its clock and timer interfaces. The Application Environment Profiles (AEP) [2a, 2b] specify the respective subset of the POSIX specification that also includes the real-time operating system functionality.

The SCA specifies what a *logical device* or *service* shall look like, but it does not specify their concrete functionalities. The latter is in the scope of a variety of **JTRS API**s.

A standard that is key for addressing aspects required for time synchronization is the JTRS **Timing Service** API [4]. The base API introduces the so-called *Terminal Time* concept that – on the one hand – adopts the POSIX monotonic clock approach and that – on the other hand – extends its scope to distributed systems synchronization: "*The Timing Service synchronizes the Terminal Time between distributed components within the terminal*". For completion, the standard introduces a quality indicator on time accuracy, i.e. Time Figure of Merit (TFOM), known from and commonly used with Global Navigation Satellite Systems (GNSS) like NAVSTAR GPS. The *Terminal Time* TFOM describes the estimated time error (ETE) of a particular *Terminal Time* clock instance within the distributed system.

The JTRS Timing Service base API also defines *System Time* as the terminal's estimate of UTC.

As stated above, the Timing Service API heavily relies on the existing, well-established POSIX concepts. Statements such as "*Terminal Time is … monotonic increasing"* and "*A waveform retrieves Terminal Time via the POSIX time interfaces*" are a clear expression of this dependency.

The scope of the JTRS Timing Service API is synchronization in a distributed system in general. The standard does not specifically address a waveform's air-interface synchronization. Nevertheless, the JTRS Timing Service provides some basic premises for this with its *Waveform Time* Extension. The extension allows maintaining, store and recovering *Waveform Time*. The standard does not restrict *Waveform Time* to air-interface synchronization, nor does it explicitly give an answer to the question on how a waveform application can achieve synchronization.

The **Transceiver Facility PIM Specification** (TFSv2) of the **Wireless Innovation Forum** (WInnF) [6a] intended to address the issue of a waveforms air-interface synchronization.

Claiming to address a wide range of transceiver types, grades and variations, the TFSv2 allows for different air-interface synchronization strategies. In particular, the TFSv2

does not presuppose the POSIX or JTRS Timing Service synchronization concepts.

That notwithstanding, there is a subset of services and interfaces of the TFSv2 that allow to build the bridge to the well-established POSIX and JTRS Timing Service concepts. A more detailed view on that subset of the TFSv2 is available within an addendum [6b] that comes with the TFSv2 and specifies a TFSv2 compliant model for Monotonic Clock Absolute Time Controlled Transceivers.

This paper outlines how the concepts already included in the SDR standards today, can be applied to solve the issue of waveform air-interface synchronization.

2. Architectural and Conceptual Considerations

The considerations presented hereafter are based upon a sample SDR system. Figure 1 illustrates its blueprint.

Platform Architecture

The platform part of the system assumes a hardware architecture with computational elements (CE) as shown. An FPGA for the real-time signal processing, e.g. of the physical layer of the waveform, and a processor (GPP or DSP) for the procedural-oriented processing of the higher layers.

We are going to focus on a simple system with two CEs only. It is obvious that the strategy also works in hardware architectures that are more complex.

Thus, the FPGA – at least if the system is designed for computationally challenging waveforms – will host most of the interfaces to the Transceiver Subsystem. The GPP and/or DSP will host the more decision-oriented interfaces. We further assume the capability of obtaining UTC time information from GNSS.

Hence, the operating environment of this system with its *logical devices* und *services* [1] is considered to provide the following capabilities:

- A Transceiver Subsystem (shaded in green) providing a TFSv2 compliant abstraction with its *Transceiver Time* (absolute time) synchronized with *Terminal Time*.
- A JTRS-compliant Timing Service (shaded in blue), which among other things administrates the synchronization functionality.
- A *Transceiver Time*, respectively *Terminal Time* access available on the FPGA. Most probably, the baseband signal interface is also deployed on the FPGA, but this neither is shown in Figure 1 nor is it a precondition.
- A JTRS MOCB API [5b] compliant interconnect (shaded in brown) between FPGA and DSP/GPP.

Waveform Application Architecture

An application architecture is supposed with components (shaded in yellow) deployed on both CEs. We again focus on a simple representation, being aware that both CEs usually host further components performing other duties.

Any application implementing a waveform that depends on a precise air-interface synchronization will exhibit a component that provides the capability to manage and control its specific *waveform time* representation. Let us call that component Waveform Time Controller (WFTC). The WFTC will at least comprise the capability to initialize, set and adjust the current *waveform time*. For duties like sending a wake-up call to another application component at a required point in time with regard to *waveform time*, the WFTC component will be the right place to implement.

The functionality needed will finally be provided to the adjacent component (exemplarily deployed on GPP/DSP CE #2) by means of operations of a waveform internal API.

Figure 1 identifies that specific WFTC application component, showing the likely case that the component is distributed across the FPGA and the processor, i.e. it has sub-components on both of the CEs. These sub-components will use the JTRS MHAL on Chip Bus (MOCB) for communication.

Air-interface Synchronization Principles

The crucial point of the approach from waveform application point of view is that the platform provides a mechanism that allows getting *Transceiver/Terminal Time* awareness into the WFTC. No issue at all since a TFSv2 compliant transceiver provides us with the respective *TimeAccess* Interface. An FPGA is predestined for implementing the component that keeps synchronicity to *Terminal Time* as that CE's technology provides hard real-time capabilities.

The task of initially establishing and then maintaining the relationship between *Terminal Time* and the specific *waveform time* is up to the application and will be performed at a proper location within the WFTC.

By now, we did not make any assumptions on the *waveform time* format. Commonly different representations are in use within an application depending on the particular scope. In order to store and retrieve *waveform time* using the JTRS-compliant Timing Service there is a need of a representation in seconds and nanoseconds. For humans it is often more comfortable to use a calendar time representation. In this case, commonly UTC is chosen to be the *waveform time*. Retrieving UTC is done by using the Timing Service's *SystemTimeAccess* interface operations that provide us with the relationship between UTC and *Terminal Time,* and an accuracy indicator (TFOM) for UTC.

UTC calendar time can be translated then to sec/nsec *waveform time* format considering the epoch (e.g. 1970-01-01 with POSIX). Beyond that, there might be further structure alternatives. For popular TDMA systems, for instance, it is likely that *waveform time* also has an equivalent representation that allows identifying individual TDMA frames specified within that particular waveform.

An application implementing such a waveform consequently will also translate the Transceiver API real-time operations to the convenient *waveform time* format.

More commonly expressed it is about establishing and maintaining the relationship between *waveform time* and the monotonically increasing *Terminal/Transceiver Time* and about mutually mapping one to the other. Finally controlling the Transceiver is done based on its absolute monotonic time base.

[1] We use the terms *device* and *service* as understood with the SCA.

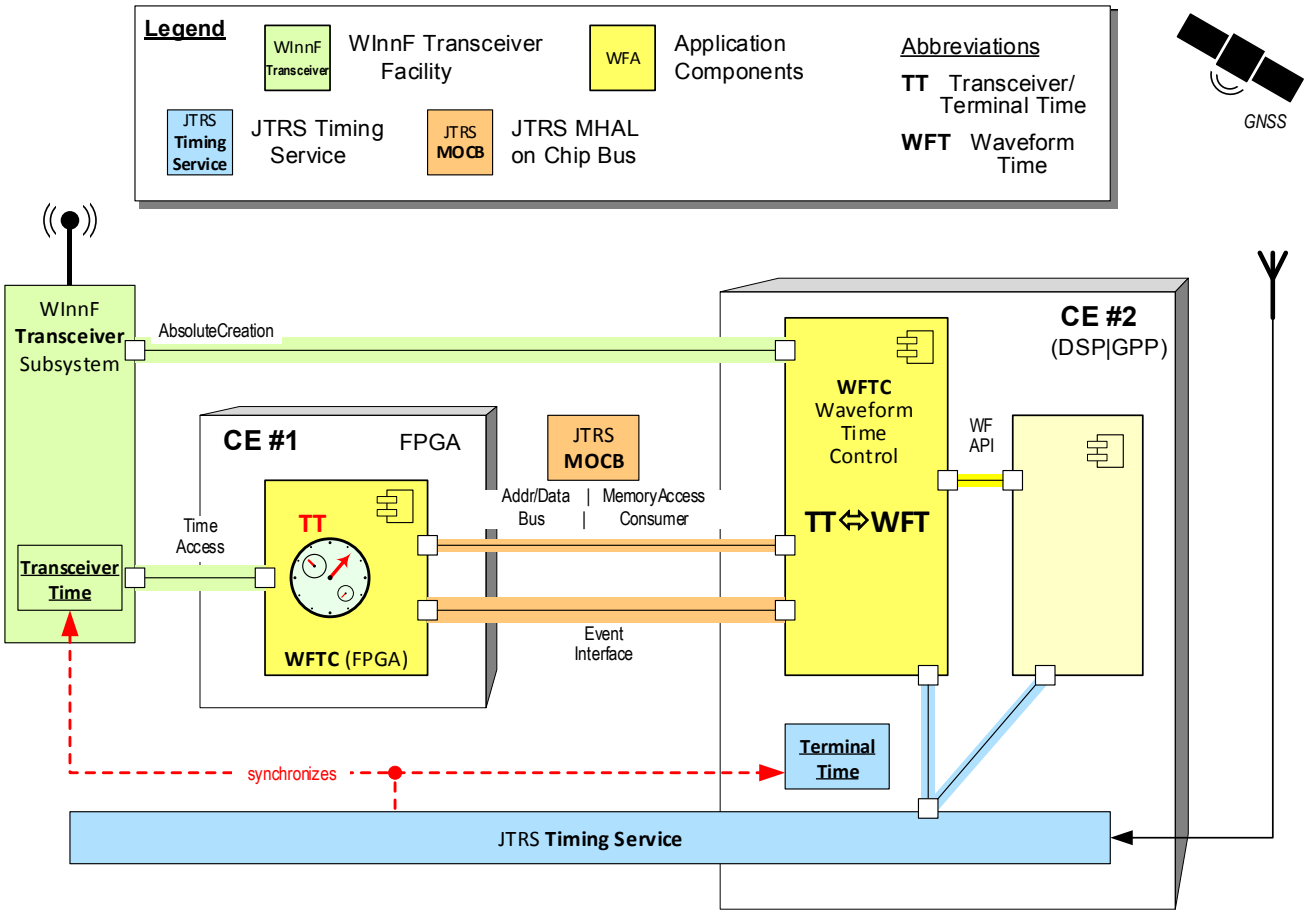


Figure 1: Exemplary model of an SDR system

This is indicated in figure 1 by identifying the mutual *Terminal Time* (TT) to *Waveform Time* (WFT) translation as the WFTC's main task. While the mechanisms in principle (e.g. setting and adjusting *waveform time*) will be similar with any waveform, the *waveform time* format will often be specific. That will express in the particular signatures of the waveform internal API provided by the WFTC.

At that point, it is worth noting that mapping any event that occurs in the transceiver to the monotonic time scale makes it easier to check for validity of a respective sequence of operation calls than in systems, where the time may make jumps. This would cause ambiguity in certain constellations on whether an operation call applies to time before or after adjustment.

Now we are close to the final question: how to synchronize application components hosted on CEs different from the FPGA. One of the questions usually arising is about receiving a trigger at the right point in time with respect to *waveform time*. It is obvious that the proper component to control and generate theses triggers is the WFTC.

As already mentioned above, these capabilities will be provided by a waveform internal API. Comprising operations that allow for requests like the following: "Give me a wakeup call at that particular *waveform time*". With the JTRS MHAL on Chip Bus, particularly with its *GPP|DSPEvent* interfaces, we have everything at hand in order to implement the waveform specific functionalities required, while properly handling the real-time constraints.

3. SCA VS. NON-SCA ENVIRONMENTS

Let us come back to a statement made previously in this paper that the approach can be applied to both SCA as well as non-SCA environments.

In the first place, the approach does not require any of the functionality an SCA Core Framework would provide. In the second place, – and this is the decisive aspect of the design – the approach does not require the availability of POSIX real-time support. Thus, it is independent of both. In the third place, – and this is an important benefit – the approach can also be applied to environments, that provide POSIX real-time support and where potentially no FPGA CE is available. Figure 2 illustrates the solution.

Implementing the Waveform Time Controller component in such a case might be assembled out of the POSIX respectively SCA AEP toolbox. This is smoothly feasible since the approach presented is based upon a set of generic concepts (like monotonic clock) common with, respectively adopted from POSIX Real-time support.

Note that motivation is not about just doing things differently. It is a tribute to different CE technology and the objective to take maximum advantage of its real-time capabilities.

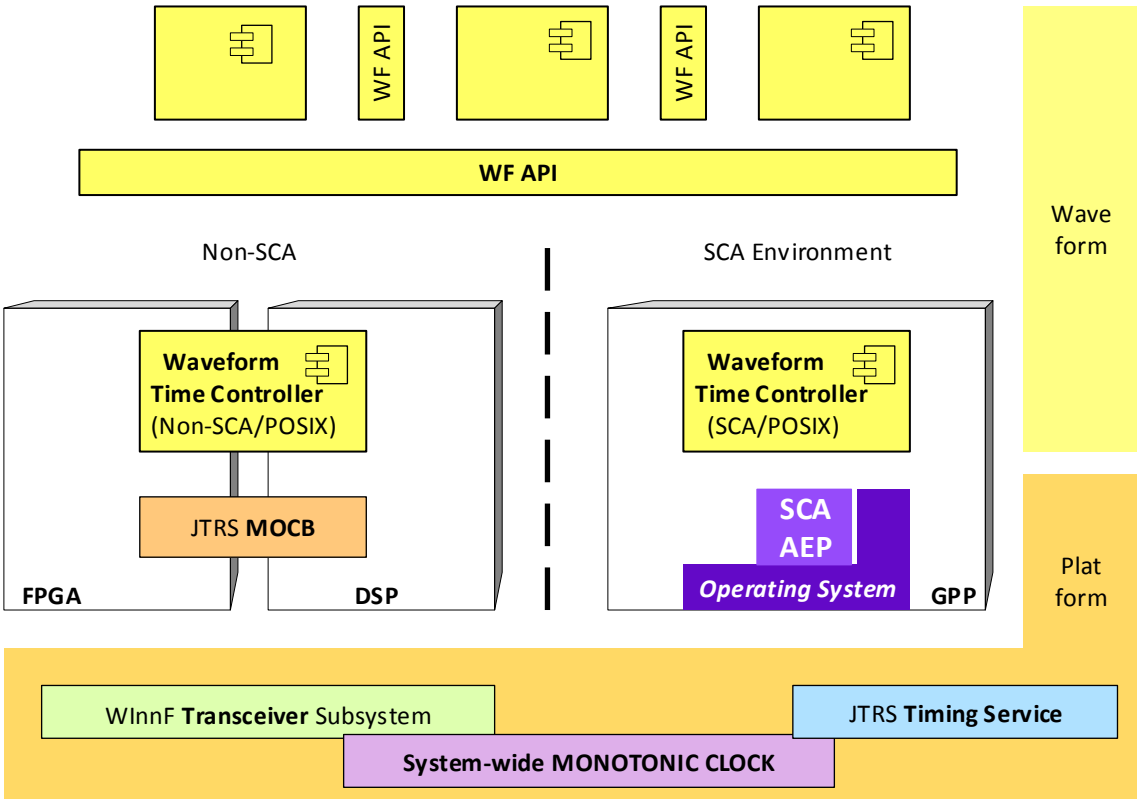


Figure 2: SCA vs. non-SCA Environments

4. Waveform Application Portability

In the SDS/SDR context, application portability is an issue independent of the synchronization approach chosen. Thus, a general objective for application architecture and design will be that major portions of an application may port at little expense to different host environments. What is directly associated to finding proper waveform internal API, separating components where adaptation effort is expected.

Respective considerations will include concepts and API available. In the first place, of course the concepts and API of the host environment for which the application initially is designed.

Particularly in the context of transceiver subsystem abstraction and air-interface synchronization a single, commonly accepted and widely spread standard has not been established yet. Over the last years, several contributions have been provided addressing particular parts of RF fronted synchronization. Exemplarily, and without any claim to completeness, they are MHAL RF Chain Coordinator [5a], Transceiver Facility Specification V1 [7] and V2 [6a, 6b], and the Open Baseband Interface Specification for SDR (OBISS) [8]. Certainly composite or even further proprietary solutions may come up with application porting tasks.

How far different approaches may be transformed into each other by proper adaptation is hard to estimate. For the synchronization approach presented in this paper, we demonstrated that it works at least with heterogeneous systems, i.e. when finding different CE technology (FPGA, DSP, or GPP). Additionally monotonic clock represents a generic concept that allows adaptation of other approaches. For instance, mapping an approach where events are scheduled relative to previous ones to an absolute time base is easy to realize. The other way round is more complicated, particularly if there is a need an initial event to happen at a particular time.

So from host environment point of view we consider the approach appropriate for efficient porting of applications initially designed against synchronization concepts different from the one presented with this paper.

5. Summary and Conclusions

The approach presented within this paper is a result of comprehensive considerations on SDR real-time and synchronization capabilities. Taking into account the various aspects from high-level overall system strategy and general objectives down to detailed application architecture and design. Key spots are on granting responsibilities between host environment and application in front of waveform portability, as well as on how to accommodate heterogeneity with regard to computational element technology.

The answers found in the respective area and reflecting with the approach are summarized hereafter. The fundamental ideas from general strategic point of view are:

- Design a platform as lean as possible. Avoid redundancy (do one thing and do it well).
- Provide waveform agnostic abstraction of functionalities rather than making assumptions on what a waveform will

need, and particularly how an application is going to implement.
- Rely on existing and established concepts and standards rather than reinventing the wheel.
- Apply concepts and standards best suitable for real-time capable implementation on the respective CE technology.

Technological solutions to achieve these strategic goals were presented:

- A system wide monotonic clock represents a simple but effective concept. Particularly comprising the transceiver subsystem is an essential step.
- With FPGA and inter processor communication JTRS Standard MHAL on Chip Bus has been identified as ultimately simple but effective solution.

All of the previously said lead to appropriate objectives with regard to applications architecture, design and implementation:

- Exploit hard real-time capabilities of FPGA technology if available.
- Look for solutions that allow minimizing porting effort to dedicated application components. In other words, maximize the percentage of an application that is likely to be ported with little or no expense, what is finally a question of defining proper waveform internal API.

The achievements described by this paper are of course a result of a couple of lessons learned over the last years. From experience and many discussions, the authors believe that technical solutions need to be simple, universal, but complete in order to see wide acceptance. Commonly finding them takes its time even though it seems apparent.

6. REFERENCES

[1a] Software Communications Architecture Specification Version 2.2.2, FINAL / 15 May 2006

[1b] Software Communications Architecture Specification Version: 4.1, 20 August 2015

[2a] Software Communications Architecture Specification Appendix B, SCA Application Environment Profile Version: 2.2.2, FINAL / 15 May 2006

[2b] Software Communications Architecture Specification Appendix B, SCA Application Environment Profiles Version: 4.1, 20 August 2015

[3a] IEEE Standard for Information Technology– Portable Operating System Interface (POSIX®) Base Specifications, Issue 7, IEEE Std 1003.1™-2017

[3b] IEEE Standard for Information Technology– Standardized Application Environment Profile (AEP) – POSIX® Realtime and Embedded Application Support IEEE Std 1003.13™-20003

[4] Joint Tactical Radio System Standard Timing Service API Version: 1.4.4, 26 June 2013

[5a] Joint Tactical Networking Center Standard Modem Hardware Abstraction Layer API Version: 3.0, 02 Oct 2013

[5b] Joint Tactical Radio System Standard MHAL on Chip Bus API, Version: 1.1.5, 26 June 2013

[6a] Transceiver Facility PIM Specification WINNF-08-S-0008, Version V2.0.0, 7 June 2017

[6b] Transceiver Facility Use Case Monotonic Clock Absolute Time Controlled Transceivers WINNF-TS-0008, Version V2.0.0-A1.0.0, 9 November 2017

[7] Transceiver Facility Specification (deprecated) SDRF-08-S-0008-V1.0.0, Approved 28 January 2009

[8] Open Baseband Interface Specification for SDR (OBISS) WINNF-15-I-0094, 20 November 2015

A COMPARATIVE STUDY OF EIGHT TRANSFER MECHANISMS WITH FM3TR

Jin Lian (lianjin.me@qq.com)[2,1,3], Qi Tang* (q.tang.andy@qq.com)[1,3], Lihua Zhu (zlh@hnu.edu.com)[2,1,3], Li Zhou (zhouli2035@nudt.edu.cn)[1,3], Shan Wang (chinafir@nudt.edu.cn)[1,3], Jun Xiong (xj8765@nudt.edu.cn)[1,3], Haitao Zhao (haitaozhao@nudt.edu.cn)[1,3], Shengchun Huang (huangsc@nudt.edu.cn)[1,3], and Jibo Wei (wjbhw@sina.com)[1,3]

[1]National University of Defense Technology, Changsha, Hunan, P. R. China
[2]Hunan University, Changsha, Hunan, P. R. China
[3]Hunan Engineering Research Center of Software Radio, Changsha, P. R. China

ABSTRACT

The transfer mechanism is an indispensable constituent of the SCA-compliant SDR system. It leverages standardized client-server innovating mechanism to the SDR system that is composed of a set of different kinds of components. With the transfer mechanism, the client and server may be located in the same or different address spaces, making them apparent to each other. Many different transfer mechanisms can be used in implementing the SCA-based SDR system, including CORBA, RPC, RPC over DDS, IPC, etc. However, different techniques render various overheads to the system. Though some literature studies the performance of different techniques, these works either only focus on one or two transfer mechanisms or lack of considering a realistic waveform. For this reason, this paper conducts an in-depth analysis of the performance of eight different transfer mechanisms, i.e., ACE TAO, omniORB, RPCexpress, e*ORB, ORBit2, ICE, RPC over DDS and TCP/IP, by integrating them with the Future Multiband Multiwaveform Modular Tactical Radio (FM3TR). Various performance metrics, including latency, predictability, static and dynamic memory consumption are taken into account, thus providing a full profile for each transfer mechanism. In order to perform these measurements and result in fair results, the FM3TR is ported to different transfer mechanisms with the same function. The experimental result shows the overall performance of different transfer mechanisms, thus paving the way for selecting the transfer mechanism for the SDR system.

1. INTRODUCTION

Software Defined Radio (SDR) features of decomposing of software and hardware, modularization of communication modules, covering multiple frequency bands and supporting a large number of waveform applications, thus providing advantages of scalability, rapid iteration and wide bandwidth. In the military and civil fields, there are two typical software radio architectures. The first one is Software Communications Architecture (SCA) which is planned by the Joint Tactical Radio System (JTRS) of the United States. The other one is composed of GNU Radio and general hardware platforms, such as USRP, which is proposed by the open source organizations.

The software radio architecture composed of GNU Radio and general hardware platform realizes the function of up and down-conversion, A/D and D/A conversion by its general hardware platform; And then it transfers the baseband I/Q data to GPP; Finally, GPP processes the received data using GNU Radio. However, the main goal of GNU Radio is to quick verify waveforms and algorithms on GPP, the standardization, stability, real-time and other vital issues of the SDR architecture are not considered.

SCA is an implementation-independent real-time software architecture, proposing a set of standards for hardware, software, security architecture and application programming interface (API). SCA reduces the system development time and cost by adopting commercial standards (POSIX) and technology frameworks (XML, CORBA, UML, etc.). The reuse of software is vital for shortening the development cycle of waveform applications and supporting the portability of waveform components between different SCA implementations. Compared with GNU Radio, SCA proposes a complete set of specifications to restrict waveform design. At the same time, the Operating Environment (OE) and middleware are used to shield the differences of the communication protocol among external hardware devices. This makes SCA-compliant SDR more suitable for engineering practice, and also makes it a de-facto SDR industry standard [1].

Middleware is one kind of system software, which is above the operating system and under the application software, providing the service that the operating system can't provide for the software application, shielding the heterogeneous differences, and realizing the communication between different platforms. It enables the software developer to focus on developing his own application without concerning the platform-related communication and input/output details. It also supports the distributed application. The middleware includes Enterprise Service Bus (ESB), Transaction Processing (TP), Distributed Computing Environment (DCE), Message-Oriented Middleware (MOM), etc.

*The corresponding author.

The communication middleware can support SCA-compliant waveform running on heterogeneous multi-processor system to achieve better performance. When it comes to upgrading SCA-compliant SDR hardware system, middleware is able to realize communication between old and new systems, which reduces the iteration cost of SDR platform. Although the middleware brings additional resource overhead to the system, this sacrifice barters for more convenience, such as reusability of waveform and components, cross-platform communication, convenience of waveform transplantation [2] and development, etc.

Traditional SCA-compliant SDR uses the CORBA middleware. The implement of CORBA includes ACE TAO, omniORB, e*ORB, etc. SCA 4.0 improves the architecture and defines an independent transfer mechanism API. As long as the data transmitted conforms to the standard API, it can flexibly choose the transfer mechanism to implement SCA, which enables to select the appropriate transfer mechanism according to the specific needs.

At present, CORBA, ICE, gRPC, COM+, RMI, DDS, SOAP are widely used in software development. These middlewares are mainly divided into two groups, i.e., RPC and MOM. In engineering practice, RPC is more in line with the requirement of SCA-compliant SDR for real-time, resource-constrained embedded environment. DDS can be used as a communication engine to provide RPC functions, i.e. RPC over DDS. Beyond that, RPCexpress, TCP sockets are alternatives.

The actual development environment of SCA-compliant SDR is complex. For example, [3]points out that the number of components and the size of data packets in waveform will affect the performance of SDR. Therefore, the transfer mechanism technology is one of the key factors for transplantation, development cycle and system overhead. However, the performance test results provided by middleware vendors are usually based on very simple models, without considering the actual waveform development environment. While the literature about the analysis of transfer mechanisms combining a waveform lacks or just takes one or two transfer mechanisms into account [4].

Therefore, this paper makes a comparative study of eight transfer mechanisms that can be applied to SCA-compliant SDR with FM3TR. The advantages and disadvantages of different transfer mechanisms in scalability, usability and performance are discussed. In order to evaluate these eight transmission mechanisms from the perspective of Engineering development, we combine them with FM3TR waveform. The results are used as a reference for the selection criteria of transmission mechanism.

2. INTRODUCTION OF EIGHT MECHANISMS

The purpose of this paper is to evaluate the performance of different transfer mechanisms in SCA-compliant SDR. This section will clarify the reasons for choosing these transfer mechanisms and the performance test model combining transfer mechanism with waveform.

From the point of view of practical engineering application environment and existing literature, this paper considers both the commonly used Commercial Off-The-Shelf (COTS) software and the transfer mechanism which is less used but can be potentially applied to develop SDR systems. Here come the eight choices.

Before SCA 2.2.2, SCA specification clearly stipulated that the CORBA middleware technology must be used as soft bus. Although SCA 4.0 version no longer stipulated this, the CORBA technology still has certain advantages in modeling architecture and maturity in the application of SCA-compliant SDR. So the following four kinds of typical CORBA middleware are selected as the transfer mechanism tested in this paper.

2.1. TAO

The Adaptive Communication Environment (ACE) is a C++ communication framework for cross-platform concurrent communication. It provides many framework components and reusable C++ wrappers. TAO is an open source CORBA implementation under the ACE framework. It supports multiple platforms, including Windows, Linux, Unix, Mac, VxWorks, and many other platforms. It has been applied in a lot of software such as Software Communications Architecture Reference Implementation (SCARI).

2.2. omniORB

The omniORB is an ORB product developed by AT&T Cambridge Laboratory. It is mainly applicable to C++ and python. Open-Source SCA Implementation::Embedded (OSSIE) using omniORB as communication middleware.

2.3. e*ORB

The e*ORB is a middleware developed by Distributed Software Architecture Provider PrismTech. It complies with JTRS SCA standard and OMG minimum CORBA standard, and supports interoperability among GPP, DSP and FPGA platforms. In 2006, Virginia Tech transplanted OSSIE to TI TMS320C6416 platform and used e*ORB to adapt to resource-constrained DSP platform.

2.4. ORBit2

The ORBit2 is CORBA-compliant ORB. ORBit2 was originally designed for GNU Network Object Model Environment (GNOME) projects. It is mainly applicable to C, C++, python. It also supports Perl, Lisp, Pascal, Ruby, and TCL. Its ORB core is written in C and can run on Linux, UNIX and Windows platforms.

2.5. ICE

In addition to the traditional CORBA, Internet Communications Engine (ICE) will also be tested. ICE is an object-oriented RPC framework for supporting distributed applications. ICE aims to free developers from the trivialities of underlying network programming, such as network connection, serialization and deserialization. It allows developers to focus on business logic. There are a lot of discussions about CORBA and ICE. ICE supporters have the following points of view [5]:

1.CORBA standard is too much and complex, but has no manufacturers truly implement all features of CORBA. The standard of CORBA is meaningless in the situation that CORBA implementation by different manufacturers is incompatible with each other. Beyond that, the huge and complex characteristics make CORBA itself difficult to use.

2.CORBA C++ mapping has many shortcomings and pitfalls in memory management and exception safety. In contrast, ICE C++ mapping is simple and intuitive and it will not leak memory due to errors. ICE C++ mapping is based on the Standard Template Library (STL) of industrial standards, and the C++ mapping rules need to be remembered are much less than CORBA.

3.CORBA's inefficient alignment rules lead to redundant data copies. Data encoding is complex but does not lead to corresponding performance improvements. IIOP's complexity leads to interoperability and performance problems. ICE's protocol is simple and more efficient, providing some features that IIOP does not provide, such as data compression and batch request batching.

Although a part of the view is improved by the experimental results, lots of transfer performance advantages of ICE have not been confirmed in previous studies. Therefore, further experiments are needed to evaluate whether ICE can perform well as SCA transmission mechanism.

2.6. RPC over DDS

The Data Distribution Service (DDS) is a network middleware used to simplify complex network programming. It is also a standard proposed by OMG to provide scalable, real-time, reliable, high-performance and interoperable data distribution services. It implements a "publish-subscribe" mode for sending and receiving data, events, and commands across languages and platforms between nodes.

The main advantage of DDS is that it need not to pay attention to the information receiver, the location of the recipient and whether the message is sent or not. It configures DDS communication behavior through the QoS parameters. This advantage is more obvious when there are more data providers and receivers.

However, DDS only provides a way for message publishers to communicate with message subscribers in real time. It does not provide the operation of remote object request proxy, so the DDS alone can't match the SCA-compliant Radio platform. RPC over DDS of eProsima company implements remote process calls using DDS as the underlying transfer engine. Therefore, it may be used as a transfer mechanism under SCA 4.0 standard.

2.7. RPCexpress

The RPCexpress is designed to support object-oriented component development and provide remote procedure call functions. The key function of RPCexpress is to provide standardized method call semantics between client and server in the same or different address spaces. RPCexpress was originally designed for embedded software definition system, implemented by C++, and currently supports Linux and Windows platforms. It holds most IDL semantics including basic data types, struct, sequence, array and any. It occupies less resources and displays high transfer performance.

2.8. TCP sockets

The last transfer mechanism considered in this paper is TCP sockets, which is a naive inter-process communication mechanism. TCP sockets encapsulate TCP/IP. Compared with other transfer mechanisms, the socket lacks many kinds of services that are often provided the available middlewares, such as naming service, event service, query service, concurrency control service, etc. Besides, to use it in the software, the developer also needs to encode and decode the transmitted data, making it more difficult to be applied in developing softwares. However, the advantages of TCP sockets are remarkable, e.g., fewer layers of encapsulation, lower transfer latency, convenient for real-time data interaction.

3. FM3TR WAVEFORM

This paper uses FM3TR to reveal the performance of different transfer mechanisms. The FM3TR waveform defines voice mode and data mode, implementing frequency hopping over both VHF and UHF military bands. The FM3TR voice mode is based on 16kHz PCM coding. CVSD component compresses the PCM stream and support semi-duplex communication between different PTT terminals. The data mode is different from voice mode, whose data source is processed through the RS component. Channel coding is adopted to improve the reliability of transmission. Fig.1 shows the structure of the FM3TR waveform.

The conduct the experiment, the communication interface between components of FM3TR is unified. The communication interface are implemented by eight communication mechanisms mentioned above. Data interaction is carried out by the pushPacket function. The following is part of the definition of interface:

```
module Packet
{
        interface OctetStream : PayloadStatus
        {
                void pushPacket(
                    in StreamControlType control, in JTRS::OctetSequence payload);
                raises(UnableToComplete);
        };
};
```

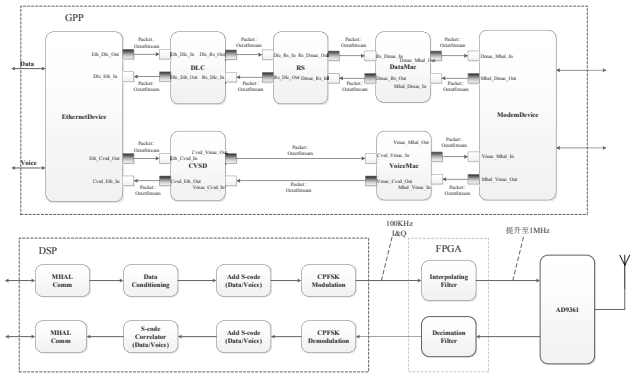


Figure 1: FM3TR waveform

Devices & Components:

- The EthernetDevice communicates with the host computer through TCP sockets, on the other hand, it interacts with other components through the selected transmission mechanism.
- The ModemDevice is essentially a Modem Hardware Abstract Layer (MHAL), which interacts with subsequent components after adding data type (voice/data) information to the original data.
- The Data Link Control (DLC) component replaces the header of the data packet for the data packet from the EthernetDevice, automatically fills in less than 111 bytes of payload and sends it to the RS component, and provides a simple automatic repeat-reQuest (ARQ) mechanism. For the data packet from the RS component, the way to process the message is decided according to the header information: if the destination address is 0 (broadcast), it will be sent directly to EthernetDevice. If the destination address is local, detecting data and sending response ACK, then send it to EthernetDevice. In other cases, the packet is discarded.
- The Reed-Solomon (RS) adopts RS code to achieve the channel coding and decoding,since it can improve the reliability of the data transmission.
- The Data Media Access Control (MAC) component mainly completes the function of data packet framing. For RS component's data packet, it is sent to ModemDevice after framing operation. Each frame includes four data hops and one synchronous hop. If the received data is ACK reply packet, no processing is done and sent directly to ModemDevice.
- The Continuous Variable Slope Delta (CVSD) is a voice compression coding method. CVSD component mainly realize the function of voice data compression coding and decoding.
- The voice Mac component has the same function as data Mac component. But voiceMac only processes one frame at a time, and processes data directly from ModemDevcie.

4. PERFORMANCE MEASURING

This section describes methods used for conducting the experiment and metrics for performance evaluation of different transfer mechanisms. Further, performance results of eight transfer mechanisms are analyzed. The FM3TR combined with each transfer mechanism is developed on a virtual machine running Ubuntu 14.04 LTS using VMware that is deployed on a host computer with an Intel Core i5-3470 processor (3.20GHz and 4.00 GB RAM). In the experiment, the function code of each FM3TR component is exactly the same, thus avoiding differences in time and space performance caused by different implementation methods, which ensures the correctness and fairness of the result.

4.1. Performance Metrics

We select some key indicators to measure the performance of the transfer mechanism from different perspectives [6].

It should be noted that although we try to ensure the permanent hardware and software environment, it is impossible to guarantee that no other process occupies the CPU or other computing resources except for the test waveform due to the complexity of the operating environment. This will inevitably lead to jitters in the executing environment of the experiment, and then affect the experiment results. The easiest way to solve this problem is to enlarge the number of conducted tests and then use the average value to eliminate the impact of abnormal data caused by the jitter of the test environment. However, in the actual experimental process, this method cannot achieve the expected goal. Even if the sample size is expanded, the abnormal result will appear randomly. This indicates that the interference that occurs in the experiment is the background interference that appears randomly throughout the whole experiment. To deal with this interference, a very simple "denoising" method is used to preprocess the raw data [7]. That is, firstly, the mean μ and standard deviation σ of the experimental results are calculated, then the data whose value is greater than μ+3σ is discarded. Statistics show that the amount of abandoned data is less than 2% of the original data, but most of the interference is effectively eliminated. Compared with the original test results, the processed test results can better reflect the general performance of the transfer mechanism tested, so it has more reference value.

A. Latency

In SCA-compliant SDR, system latency is an important indicator to evaluate the transfer mechanism [8]. Besides, low latency is also a necessity of real-time systems. As Fig.2 shows, the system latency is divided into four parts, i.e., propagation latency, transmission latency, queuing latency and processing latency. The transmission latency refers to the time cost of data transmission from PC to waveform application. The propagation latency refers to the time of data transmission between components. The

queuing latency refers to the time interval between the time when the data packet arrives the packet queue of the component and the time when the component starts processing this packet. The processing latency refers to the time costed by the component for processing a data packet.

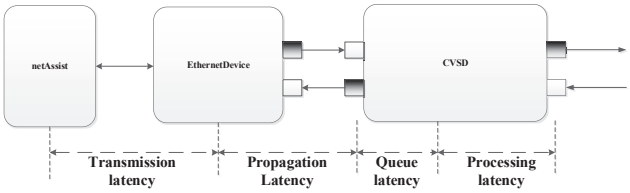


Figure 2: Latency profiling

Because the implementation code of each component in the waveform are identical, the processing latency are identical when measuring different transfer mechanisms. In order to carry out the loop test, we developed a small program called netAssist which simulates the sending and receiving of data packets in specified format from the host computer. After the FM3TR waveform runs, netAssist sends the data packet to the EthernetDevice component, and then the data packet returns to the original path after each waveform component in turn. Finally, the data packet is sent to netAssist by the EthernetDevice component. NetAssist records the amount of data it sends and receives to detect whether data packets are lost during testing.

To measure the propagation latency accurately, the *timingLog* [9] tool is used to record the time when the EthernetDevice receives the data packet and the time when the EthernetDevice starts sending the processed data to the PC, so as to eliminate the effect of transmission latency. The latency of a single loop is $t_{loop} = (t_{end} - t_{start})/N$.

B. Throughput

The above single round-trip time is an important indicator of latency when using different transfer mechanisms. However, the overall performance is also related to the throughput. Throughput not only reflects the ability of processing queries per unit time, but also reflects the performance of transmission mechanism considering the influence of waveform. The calculation of throughput is obtained by the latency between nodes and the corresponding packet size.

C. Predictability

Due to the complexity of the entire operating environment and the volatility of the network environment, jitter of propagation latency is inevitable. Predictability depends on jitter of latency. The jitter of latency can lead to a series of problems. For example, when voice data does not arrive at the receiver end evenly, the receiver end must make up and try to correct, otherwise it will cause the user's voice distortion problem. In addition, the jitter of latency will also lead to network congestion. Therefore, delay jitter is one of the important performance indicators of the transfer mechanism. We use the standard deviation of the latency indicates the degree of the jitter of latency for different transfer mechanisms.

D. Static Footprint Size

SDR systems often need to run on the resource-constrained platforms, such as the popular embedded mobile platform. On these platforms, the battery life of the device is an important factor affecting the user's actual experience. And memory size is one of the factors affecting device power consumption. On the other hand, if you want to use the same transfer mechanism on a heterogeneous processor, you also need to consider the memory requirements. Static memory is mainly composed of library files, IDL compiler-generated or handwritten frameworks and pile files, and functional implementation code. Common shell commands, such as ps, top, etc., can be used to measure memory size. In addition, the size of static memory can only partly explain the memory occupancy of the transfer mechanism, which needs to be combined with dynamic memory in order to conduct a comprehensive analysis.

E. Dynamic Memory Size

The overhead of dynamic memory is not necessarily related to the size of its static memory. The dynamic memory is composed of a stack opened by the process, data segment memory, and various service logs. Using only the aforementioned shell commands, we cannot examine the dynamic memory usage. Therefore, we use the open source tool Valgrind to analyze the memory usage of the process in real time through time slicing.

4.2. Experiment Results

The reasons for choosing these indicators to measure the performance of the transfer mechanism have been explained in the previous paper. This section will analyze the experimental results using above methods.

A. Latency

Fig.3 shows the round-trip latency when eight transfer mechanisms combine the same waveform. Among them, TCP sockets, RPCexpress, omniORB and TAO have significant advantages in round-trip latency. It is noteworthy that we find the time cost for TAO to establish connections between components is significantly longer than that of other transfer mechanisms in experiment, but we do not consider this factor in this experiment.

B. Throughput

Fig.4 shows the average Queries per second of the eight transfer mechanisms to describe the throughput of them in

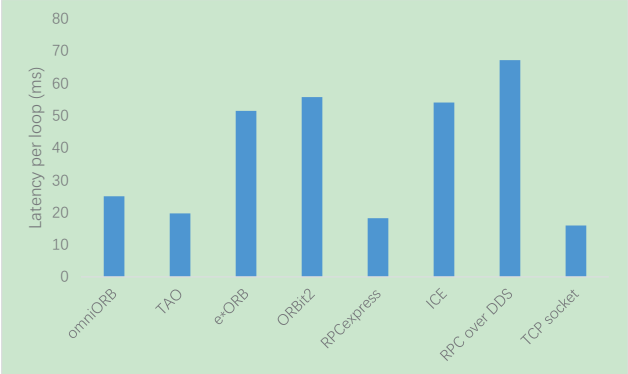


Figure 3: latency per loop

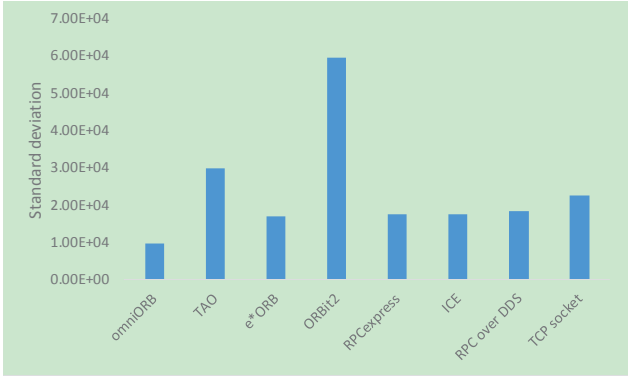


Figure 5: stand deviation

this experiment. The QPS of TCP socket is significantly higher than that of all other transfer mechanisms, while RPCexpress, omniORB and TAO are significantly higher than the remaining four transfer mechanisms. Although TCP socket has a faster transmission rate due to fewer encapsulating, but in multi-component waveform, it adds to difficulties of waveform development.

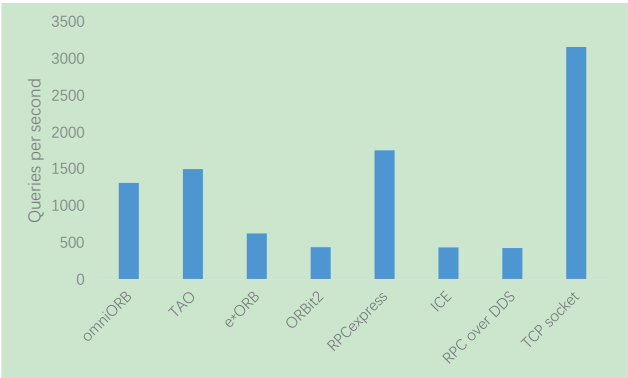


Figure 4: queries per second

largest, followed by TAO and ICE. Compared with framework code, the size of library file is the most important factor which affects static footprint memory. Among them, RPC over DDS and ICE have larger library files. When choosing the transfer mechanism, we should take full account of the memory resources. Therefore, we will analyze the dynamic memory occupancy of each transfer mechanism in the next subsection.

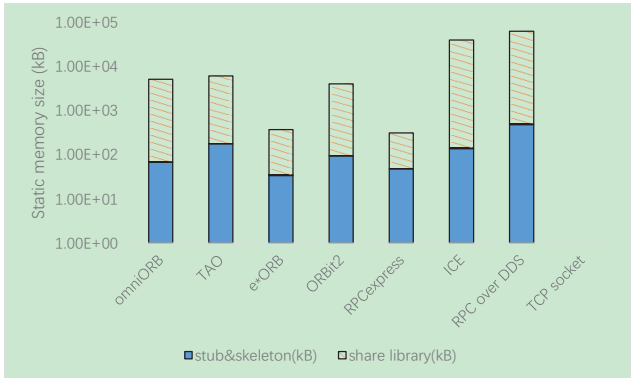


Figure 6: static memory size

C. Predictability

The latency jitter of eight transfer mechanisms is shown in Fig.5. We describe the degree of lantency jitter by the standard deviation of latency. From Fig.5, we can see that the latency jitter of ORBit2 is high. Considering Fig.3 and Fig.4, ORBit2 has higher latency and lower throughput than others. This shows that the jitter in transmission will have some impact on the performance.

D. Static Footprint Size

Fig.6 shows the size of static footprint memory of eight transfer mechanisms. Static footprint memory consists of implementation code, framework code and shared library file. The implementation code in this experiment is basically the same, so it is not considered. From Fig.6, we can see that the size of RPC over DDS framework code is the

E. Dynamic Memory Size

In Fig.7, ICE occupy significantly larger dynamic memory, while omniORB and RPCexpress occupies less dynamic memory than others. Combining Fig.6, when waveform runs, RPCxpress and ICE require the largest memory resources. Among them, RPCexpress requires the least total memory except TCP socket.

Acknowledgments:
We would like to thank Hao Liu and Shi-Li Zhu for their assistance in this work.

REFERENCES

[1] G. Abgrall, F. Le Roy, J.-P. Delahaye, J.-P. Diguet, and G. Gogniat, "A comparative study of two software defined radio platforms,"

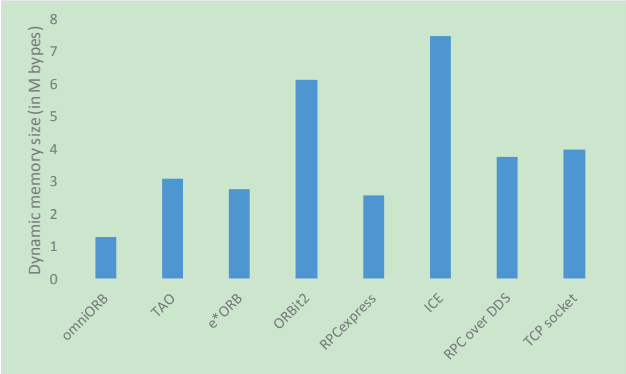


Figure 7: dynamic memory size

SDR '08 Technical Conference and Product Exposition, 2008.

[2] F. Le Roy, L. Rakotondrainibe, J. P. Delahaye, and A. Mansour, "Insights into portability issues of FM3TR waveform," *Analog Integrated Circuits and Signal Processing*, vol. 3456789, pp. 1–13, 2017.

[3] P. Putthapipat, "Lightweight Middleware for Software Defined Radio (SDR) Inter-Components Communication," 2013. [Online]. Available: http://digitalcommons.fiu.edu/etd/867

[4] A. Gokhale and D. C. Schmidt, "Measuring the performance of communication middleware on high-speed networks," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 4, pp. 306–317, 2004.

[5] S. Ravindra Babu, "The Rise and Fall of CORBA," *Adarsh Journal of Management Research*, vol. 2, no. 2, p. 63, 2016.

[6] I. Gomez, V. Marojevic, J. Bracke, and A. Gelonch, "Performance and overhead analysis of the ALOE middleware for SDR," *Proceedings - IEEE Military Communications Conference MILCOM*, pp. 1134–1139, 2010.

[7] G. Abgrall, F. Le Roy, J. P. Diguet, G. Gogniat, and J. P. Delahaye, "Predictibility of inter-component latency in a software communications architecture operating environment," *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum, IPDPSW 2010*, pp. 1–8, 2010.

[8] B. Wireless and V. Tech, "Latency Profiling for SCA Software Radio," *Forum American Bar Association*, pp. 2–7, 2007.

[9] Q. Tang, J. Lian, L. Zhou, S. Wang, J. Xiong, H. Zhao, S. Huang, and J. Wei, "RPCexpress : a try to implement an efficient middleware from the ground up based on requirements of embedded software defined systems," *Proceedings of WInnComm 2018*, no. December, pp. 1–5, 2018.

EXPERIMENTAL EVALUATION OF LSPR ROUTING PROTOCOL

Khalid Hussain Mohammadani
School of EE, Beijing University of Posts and Telecommunications
Beijing, China
khalid.mohammadani@gmail.com

Safiullah Faizullah
Dept.of computer science
Islamic Madina University
Madina, Saudi Arabia
safi.research@gmail.com

Kamran Ali Memon
School of EE, Beijing University of Posts and Telecommunications
Beijing, China
Ali.kamran77@gmail.com

Ali Alzahrani
Dept.of computer science
Islamic Madina University
Madina, Saudi Arabia
alnaashi@hotmail.com

Turki Alghamdi
Dept.of computer science
Islamic Madina University
Madina, Saudi Arabia
dr.turki.iu@gmail.com

Arshad Shaikh
Department of Computer Science
Isra University
Hyderabad, Pakistan
amshaikh@hotmail.com

***Abstract*—Mobile ad-hoc network (MANET) routing protocols can be classified as either topology based (which uses the inter-node connectivity information to create routes) or position based (that uses geographical positions of the nodes for routing and forwarding decisions). Topology-based routing often runs into problems due to its inability of identifying week links (links that are about to break due to mobility), resulting in broken routes and hence cost performance penalty. Position based routing often uses a greedy approach that stuck in a local-maxima situation resulting in a stalled communication. In our previous work** [1] **we had introduced LSPR (Location Server based Proactive Routing) protocol that offers a unique hybrid of both topology-based and position-based routing strategies. LSPR constructs routes from topology information extracted from geographical position data of the nodes. It ignores any weak links by confirming that each pair of communicating nodes are located at least two-third transmission-range apart. In this study, we attempt to provide more support and evidence that our LSPR protocol is indeed a better choice for routing in MANETs. We hereby compared the performance of LSPR protocol with AODV, DSDV, LAR, and LSAR protocols under varying mobility and network conditions.**

Keywords—MANETS, GPS, Topology-based routing protocols and Position based routing protocols.

I. INTRODUCTION

A transient network is a set of portable nodes (i.e. mobile phones and laptops etc.) known as a mobile ad-hoc network (MANET). MANET is established by wireless connections without any access point, infrastructure or centralized management[2]. In MANET, all portable nodes act as both routers and hosts. Physical topology is affected and changed from time to time due to joining and leaving the portable nodes[3]. The most important consciousness of the MANET format tradition is to understand the right and useful direction between some nodes in order to make timely and reliable delivery of messages possible. Route detection would be found with minimal overload[4]. Each portable node has topology information according to different routing techniques in routing tables. By using a proactive manner of routes nodes exchange of routing information periodically[5]. Proactive routing technique requires that each node maintains and updates routing tables, according to changing in the network topology[5]. Paths can be made in a reactive manner, only when they need the original node. The position-based or location-based routing protocols are excellent for aggregation due to ad hoc networks: that is not necessary for accordance with preserve the routing tables updated or in accordance with having a huge view regarding network topology and adjustments, which translates into reducing overhead routing. Bandwidth optimization, dynamic topology, link failure scalability, and routing are the main challenges of MANETs. The link failure is the biggest issue due to nodes movement independently in any direction[6]. Researchers developed several routing protocols to overcome these main problems. Topology-based and position based routings are broad categories of routing protocols for MANET.

Topology-based routing protocols have all information based on network structure. These are not suitable for MANET when the network nodes do not have a constant position and always change their position in the network. These routing protocols are also effective and non-stable in a high-density network with where high traffic may generate[7].

On another side, position-based routing protocols are suitable for the high dense network to maintain the network topology[8]. These type of routing can easily handle the data forwarding among the nodes. They depend on local data to redirect the data packets instead of maintaining entire network information. It is the plus point of position based routing protocols.

This paper presents the experimental evaluation of our previously developed position based routing protocol known as LSPR[1] that is compared with other position based and topology based routing protocols

The rest of the paper is prepared as the following section: section II describes some review of routing protocols of MANET and types of routing protocols. Section III presents the methodology of this paper. Results and discussion are explained in section IV while section V concludes the paper followed by references.

II. LITERATURE REVIEW

Conventional routing protocols rely primarily on information in the routing table that corresponds to the movement of the paths with many possible addresses. To ensure that the routing tables are updated and reflect the actual network topology, it often exchanges route updates and route descriptions. Link State Algorithms (LS) and Distance vector algorithms (DV) are two different conventional algorithms for network routing.

A. Link State (LS) Routing Algorithm

Each node maintains the root of the entire topology for each individual link in this type of routing[9]. As a result, all network nodes quickly show partnership charges for each connection to the different nodes using flooding. These types of flooding floods make it possible to predict costs for each other. Each host in the system has an address table that is used to save all the connection costs that the node gets. As a result of receiving the surplus message, each node updates its routing table and chooses the most specific format for each target node. These communication costs can give erroneous data to the communication costs in any node due to delayed deployment, distributed systems and etc., leading to the evolution of the direction cycle. These cycles are fleeting, as they disappear when the lifetime of packet expires. On the other hand, these cycles increase the overhead in the system.

B. Distance Vector (DV) Routing Algorithm

In this type of routing, each node does not indicate the cost of its friendly links, but, unlike it sends it to all hosts, it shows an estimate of the shortest distance for each node on each of its neighbors [9]. Then the node that is available then uses this information to recalculate the routing tables with the shorter method of calculation. In contrast to LS routing protocols, the DV routing protocols are more efficient, less operational and have much less storage space. Then, again, distance vectors can lead to the development of short and large-scale routing rings. The key factor is that the nodes determine their confidence in the home in a completely scattered way that is centered on data that can be rigid. DV routing protocols of MANET are divided into two categories: Topology-based Routing and Position-based Routing. Fig.1 shows their classifications.

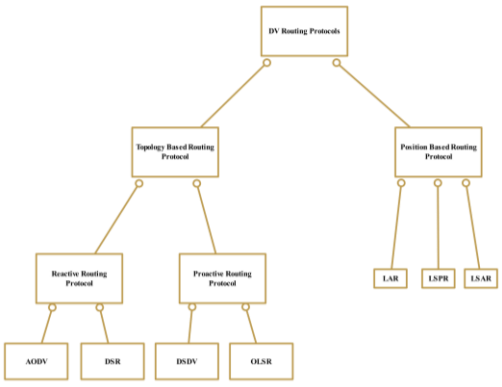


Fig. 1. Distance vector routing protocols of MANET

C. Topology-based Routing Protocols

Topology-based routine protocols already provide information about network connections to perform packet redirection. The pre-defined routing protocol was based on the topology information that consisted of establishing the path and maintaining the path. They use the link information in the network to forward packets.

D. Proactive Routing Protocols:

Proactive routing protocols observe a similar approach to wired routing protocols. By constantly comparing the recognized routes and attempting to discover new routes, they try to hold an up-to-date map online[10]. This allows them to send packets efficiently because the path is recognized while the packet reaches the node. Examples of proactive routing protocols are Destination Sequenced Distance Vector (DSDV) and Optimized Link State Routing (OLSR) protocols.

E. Destination-Sequenced Distance Vector (DSDV)

DSDV is a type of proactive routing protocol. Each node sustains routing tables. The routing table is updated continuously. Nodes may send and receive the packets in the network with the assist over routing information. Sequence numbers originate primarily from the same node of the receiver, ensuring continuity of the loop. The installation time removes the false entries from the table. Original data is a pointer to a table that has path validation information and is also used to evaluate network variations[11].

F. Optimized Link State Routing (OLSR) Protocol

OLSR is an optimization of the pure hyperlink state protocol by reducing the range of knowledge distributed in messages and reducing the amount of retransmission to transmit these messages in the entire community. For this reason, the OLSR protocol uses multicast retransmission technology to successfully and economically flood its messages. It displays the more specific methods in the jump number phrases, which are immediately available at will. OLSR is better suited for dense and important mobile networks[12], [13].

G. Reactive Routing Protocols

Unlike proactive routing protocols, reactive routing protocols do not attempt to establish a network connection over time. Rather, the routing development is desired on demand by any node which has to send packets. The method is predicated proceeding the requests which are inundated the entire MANET[14]. Ad-hoc on-demand distance vector (AODV) and Dynamic Source Routing (DSR) routing protocols are two types of reactive routing protocols.

H. Ad-hoc On-demand Distance Vector (AODV)

AODV uses a procedure for detecting courses in order to gradually build new courses based on need. AODV is a diffuse account that uses vector-separator calculations. As soon as the session becomes a dark destination, AODV causes a cycle to request a packet and deliver it to its neighbors. The preferred point of view of this agreement is that the courses depend on the interest and the succession numbers are used in the destinations to determine the most recent route to the destination. In this sense, the postponement of the composition of the Assembly shall be less. However, since courses are only held during use, it is generally required that the session is detected before packets

are exchanged. This prompts to postpone the main package to be transferred[15].

I. Dynamic Source Directive (DSR)

DSR is an address convention for remote poetic systems. It is like AODV where you set a frame of interest when the transmission distributor requires it. In any case, use the source address rather than relying on the address table in each intermediate device[16].

J. Position-Based Routing Protocols

The position-based address agreement uses location data to find the exact areas in the destination center, as well as its adjacent location. It uses location data to provide a more reliable and efficient address for specific applications and this information is mostly obtained through the Global Positioning System and regional administrations. Because of the use of district administrations and cargo procedures, their implementation has been much improved than the structure-based management agreement. It shows a better diversity, a force against continuous topological changes. These management agreements aim to improve efficiency and implement the system. The address is executed in a bounce style to redirect packets of information. Its purpose is to deal with regulators who have many centers. The preferred position for this type of address is that the data next to the information packet is fully displayed, instead of retaining the entire system data. This will reduce the overload on the address and increase packet transfer speed. Location data for each hub is determined using location services and the use of forwarding routines to forward information packets[17]. At the point where the source distributor needs to move a packet to the destination, it must obtain the area (x, y) of the destination through the site service.

K. Location-Aided Routing (LAR) Protocol

The purpose of Location-Aided Routing (LAR) described in [11] is to reduce overhead. LAR uses data that can limit floods to a specific region, known as the request zone area. As a result, several application packets is decreased in a manner. Rather than the entire flood into the network, which includes a routing packet, LAR sends packets to nodes with a very high probability of finding a route. The LAR estimated zone is defined as the area that is estimated to be the target's recent position point. Throughout the process of finding the route, the flood system asks for an operating area where the potential strap and the location of the wired node are.

L. Location Server Assisted Routing Protocol (LSAR)

LSAR is one of the reactive protocol[18]. It uses geographical data to locate the shortest routing path among the nodes. Instead of straight flooding, this convention sends bundles of information through links that then culminate in the topology-based routing protocol. The LSAR is responsive, rooted and makes the course only when needed. In LSAR, the root node is responsible for supporting the route. All nodes update their tables as they get immediate root announce message. LSAR includes complex functionality for data forwarding process. It takes time to route discovery because of complex functionality and routing overhead is high. The complex functionality includes on some set of messages that are: Send Rout request (SRREQ), Receive Route Request (RREQ), Send Route Reply (SRREP), Receive route reply (RRERP), Send Route set (SRS), Receive route set (RRS). Still, it was compared with LAR and AODV but gave better performance in sense of PDR and throughput.

M. Location server based proactive routing (LSPR)

LSPR is one of mixed routing protocol that uses proactive based and position based routing protocol approaches. In our previous work[1], we proposed the LSPR routing protocol and compared with DSDV and LSAR. In this paper, LSPR is compared with AODV, DSDV, LSAR and LAR routing protocols. Further, LSPR is explained in the next section.

III. METHODOLOGY

This part explains the functionality of the LSPR. The functionality of LSPR includes main three parts. i. Root Announce (RA) ii. Announce to Root (AtR) iii. Data Forwarding (DF).

A. Overview of LSPR

Initially, LSPR uses the location registrar also known as the root node to maintain the routing information in the network. Root node sends Root Announce (RA) message included on the adjacency matrix to every node about their neighbor nodes. Each node updates the root node about its GPS coordinates, in reply to the root announce message. Unlike LSAR, Root node does not help every node to find the shortage path but every node runs shortage path algorithm (i.e Dijkstra algorithm) itself on available adjacency matrix for shortage path in the network. It will help to reduce the routing overhead. LSPR include three primary functions to forward the data.

At the start, Root Announce (RA) is the first step of LSPR in which node 0 is chosen to make the Root Node or location registrar. Root node transmits an RA message in the network. Initially, this RA message has the empty adjacency matrix which is filled as other nodes send their GPS coordinates to the root node via announce to root packet. All nodes receive and send the RA message in the network as shown in Fig. 2.

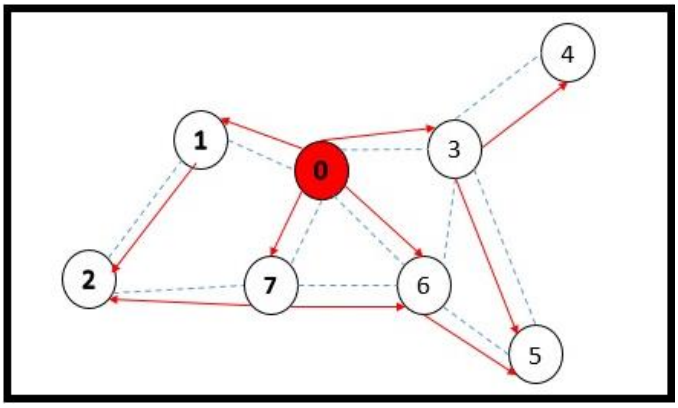


Fig. 2. Distance Root Announce (RA)

Next, Announce to Root (AtR) is the second step as network nodes received root announce message as they reply the AtR message with their own locations to the root node and root node records them and makes an adjacency matrix. The root node executes distance formulas regarding the GPS

coordinates yet fills the adjacency matrix with binary information. This is a procedure to store the location of all nodes. As a result, all nodes also know that the root node and next hop as shown in Fig.3. In this way, all nodes familiar own neighboring nodes in the network.

At the last, Nodes can forward the data after receiving the adjacency matrix known as Data Forwarding (DF). Assume, Node#7 needs a route to transfer data packets to node#4 (as Fig.4 shows), node#7 executes the Dijkstra set of rules at the given adjacency matrix to check the shortest path of the target node, which is among 7,6,3 up to 4 and so forward the information.

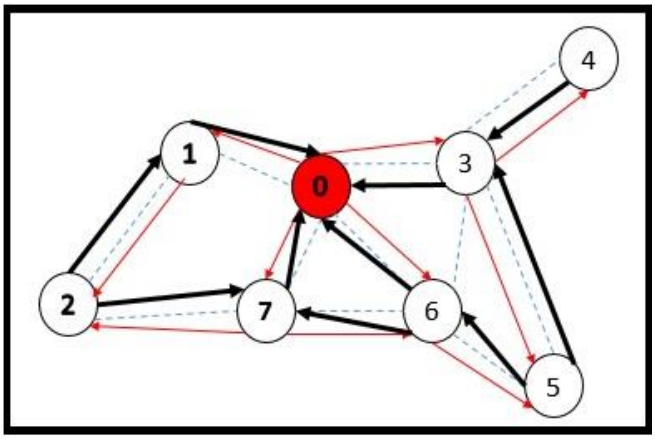


Fig. 3. Announce to Root (AtR)

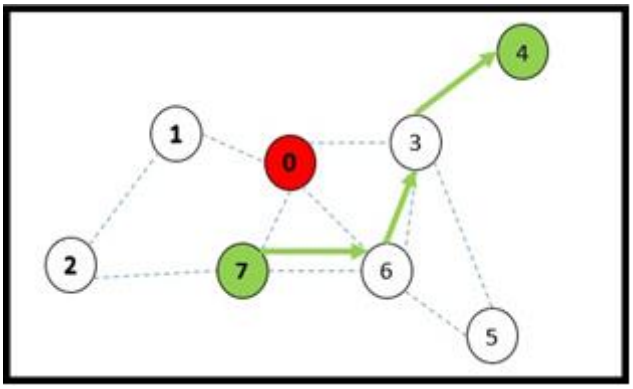


Fig. 4. Data Forwarding (DF)

B. Performance Evaluation

LSPR has been developed for Network Simulator NS-2 using C++ by our research team. NS-2 is one of the famous open source network simulators which includes basic MANET routing protocols like AODV, DSDV, and OLSR. We have also run the patch of LAR and LSAR routing protocols in NS-2. We have simulated five routing protocols to examine the performance of LSPR as well as its competing routing protocols AODV, DSDV, OLSR, and LSAR. Three different quality of service parameters. In addition, three mobility speed (e.g. 5, 10 and 20) m/s are also simulated to compare the performance of five routing protocols. Table I shows the simulation parameters used in this paper.

TABLE I. SIMULATION PARAMETERS

Parameters	Values
Simulation Time	600 sec
Topology Size	500m x 500m
Number of Mobile Nodes	100
Mobility Model	Random Waypoint
Traffic Type	CBR (160 bytes packet)
Routing Protocols	LSPR, AODV, DSDV, LSAR, and LAR
Mobility Speed	5, 10, and 20 m/s
Ns-2 Version	NS-2.33

We have calculated throughput, PDR and NRL as the quality of service parameters to analyze the performance of LSPR routing protocols accordance with multiple node mobility speed and compare with AODV, DSDV, LSAR, and LAR routing protocols

Throughput is measured in bits per second (bps), kilobits per second (kbps) and so on. Eq. (1) defines the throughput in kbps. Where Pkts is packet, PktSize is packet size in bytes, 8 is multiply factor to calculate the bits. We can say the total amount of data packets are received in time (τ) in seconds.

$$\text{Throughput [kbps]} = (\sum (\text{Pkts} \times \text{PktSize}) \times 8) / (\tau \times 1024) \quad (1)$$

Packet delivery ratio (PDR) is a ratio of the total received data packets RDPkts over the total sent data packets SDPkts, expressed in Eq. (2). PDR and throughput are directly proportional to each other therefore if one increases others also increase and one decreases other will too decrease.

$$\text{PDR [\%]} = (\sum \text{RDPkts}) / (\sum \text{SDPkts}) \times 100 \quad (2)$$

Generally, all routing packets of MANET transmit tiny size of packets known as the routing packets to collect the routing information of all nodes in the MANET. These routing packets do not contain any application information like data packets. The routing packet also utilizes the same bandwidth which is consumed by data packets due to shred medium in MANET. Normalized routing load (NRL) is a percentage of total data packets received to the total routing packets by destination as shown in Eq. (3).

$$\text{NRL [\%]} = (\sum \text{RDPs}) / (\sum \text{RRPs}) \times 100 \quad (3)$$

IV. RESULTS AND DISCUSSION

In this section, LSPR is compared with AODV, DSDV, LSAR, and LAR to determine the protocol that performs better under different mobility speed for the above mentioned QoS metrics.

A. *Packet Delivery Ratio % Vs Mobility Speed (m/s)*

PDR% is a ratio of the received data packet over generated data packets as it is discussed in the previous section. Any routing protocol gives better performance whose PDR is high as compared to other routing protocols. Figure 5 shows the PDR vs mobility speed of all routing protocols. LSPR routing protocol has high PDR than other four routing protocols. However, mobility speed is increased in the second and third scenario but LSPR maintains its stability and can deliver more data packets. As compared to AODV and DSDV the LSAR and LAR routing protocols have also good performance yet their performance is lower than LSPR in all cases of mobility speed.

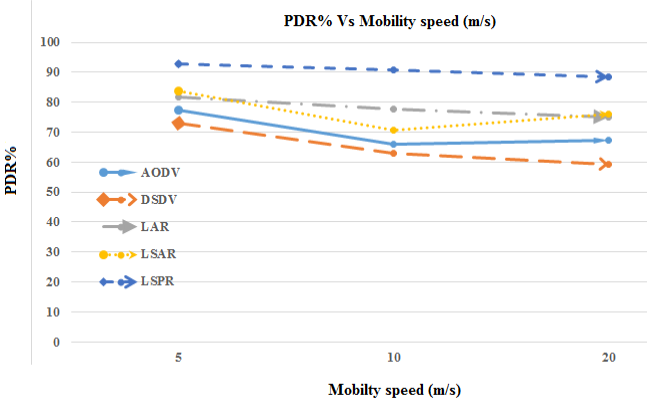


Fig. 5. PDR% Vs Mobility speed (m/s)

B. *Throughput[kbps] Vs Mobility Speed (m/s)*

Throughput was explained in the third section. It is the total amount of data received in unit time. Bits per second is the measuring unit of throughput. The throughput of DSDV and AODV similar but lower than other routing protocols in high-speed mobility scenario. LAR and LSAR defeat one another and their throughout also neck to neck. The throughput of LSPR is also similar to LSAR and LAR but still, it is performance is better as mobility increases and remains higher than others and figure 6shows the highest throughput of LSPR. As we have discussed earlier that PDR and Throughput are directly proportional to each other. Therefore the results of throughput and PDR are like similar.

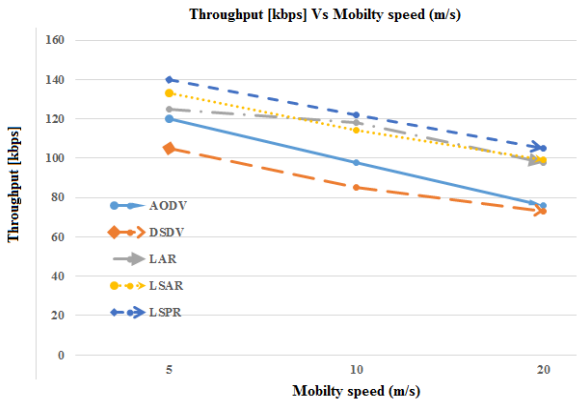


Fig. 6. DThroughput [kbps] Vs Mobility speed (m/s)

C. *Normalized Routing Load Vs Mobility Speed (m/s)*

As discussed in the third section about NRL. It is a percentage of total data packets received to the total routing packets by destination. NRL must be lowest for best performance of routing protocols. Figure 7 shows the comparative results of all routing protocols where the NRL of DSDV is high due to high mobility the DSDV cannot maintain the routing load. Even DSDV always give better performance without mobility than the other proactive and reactive routing protocol. The second highest NRL of AODV is shown in the figure. LSAR and LAR initially work similar but when mobility increase the LAR cannot maintain the routing load and it generates more routing packets than LSAR. The NRL of LSRP is better than others, its NRL is almost constant and slightly increases as mobility increases.

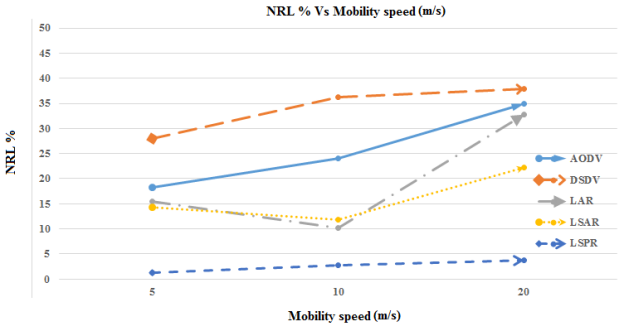


Fig. 7. NRL % Vs Mobility speed(m/s)

V. CONCLUSION

Location Server based Proactive Routing (LSPR) protocol provides a unique combination of both topology-based and position-based routing strategies. The LSPR forms paths of topology data separated the geographical location data of the nodes. It ignores all the weak connections by confirming that each pair of communicating nodes is at least two-thirds transmission-range apart. In this study, we have attempted to provide more support and evidence that our LSPR protocol is indeed a better choice for routing in MANETs. We hereby compared the performance of LSPR protocol with AODV, DSDV, LAR, and LSAR routing protocols under varying mobility. The mobility does not affect the performance of LSPR, LSPR performances best in all quality of service parameters. This is a helpful sign or we are able to address so LSPR beats AODV, DSDV LAR, and LSAR into nearly every aspect.

REFERENCES

[1] M. U. Butt, A. Shaikh, and H. Kazi, “LOCATION SERVER BASED PROACTIVE ROUTING PROTOCOL,” 2016.

[2] T. Wang *et al.*, “Propagation modeling and defending of a mobile sensor worm in wireless sensor and actuator networks,” *Sensors*, vol. 17, no. 1, p. 139, 2017.

[3] R. Suraj, S. Tapaswi, S. Yousef, K. K. Pattanaik, and M. Cole, “Mobility prediction in mobile ad hoc

networks using a lightweight genetic algorithm," *Wirel. Networks*, vol. 22, no. 6, pp. 1797–1806, 2016.

[4] L. Mejaele and E. O. Ochola, "Effect of varying node mobility in the analysis of black hole attack on MANET reactive routing protocols," in *2016 Information Security for South Africa (ISSA)*, 2016, pp. 62–68.

[5] W. A. Jabbar, M. Ismail, R. Nordin, and S. Arif, "Power-efficient routing schemes for MANETs: a survey and open issues," *Wirel. Networks*, vol. 23, no. 6, pp. 1917–1952, 2017.

[6] G. A. Walikar and R. C. Biradar, "A survey on hybrid routing mechanisms in mobile ad hoc networks," *J. Netw. Comput. Appl.*, vol. 77, pp. 48–63, 2017.

[7] J. Shen, C. Wang, A. Wang, X. Sun, S. Moh, and P. C. K. Hung, "Organized topology based routing protocol in incompletely predictable ad-hoc networks," *Comput. Commun.*, vol. 99, pp. 107–118, 2017.

[8] O. S. Oubbati, A. Lakas, F. Zhou, M. Güneş, and M. B. Yagoubi, "A survey on position-based routing protocols for Flying Ad hoc Networks (FANETs)," *Veh. Commun.*, vol. 10, pp. 29–56, 2017.

[9] S. Rosati, K. Krużelecki, G. Heitz, D. Floreano, and B. Rimoldi, "Dynamic routing for flying ad hoc networks," *IEEE Trans. Veh. Technol.*, vol. 65, no. 3, pp. 1690–1700, 2016.

[10] M. Safdar, I. A. Khan, F. Ullah, F. Khan, and S. R. Jan, "Comparative Study of Routing Protocols in Mobile Adhoc Networks," *Int. J. Comput. Sci. Trends Technol. ISSN*, pp. 2347–8578, 2016.

[11] R. F. S. Pearlin and G. Rekha, "Performance comparison of AODV, DSDV and DSR protocols in mobile networks using NS-2," *Indian J. Sci. Technol.*, vol. 9, no. 8, 2016.

[12] S. Sharma and M. A. Kumar, "Performance Analysis of OLSR, AODV, DSR MANETs Routing Protocols," *Int. J. Eng. Sci.*, vol. 7993, 2016.

[13] N. Harrag, A. Refoufi, and A. Harrag, "New NSGA-II-based OLSR self-organized routing protocol for mobile ad hoc networks," *J. Ambient Intell. Humaniz. Comput.*, pp. 1–21, 2018.

[14] H. Singh, H. Kaur, A. Sharma, and R. Malhotra, "Performance investigation of reactive AODV and hybrid GRP routing protocols under influence of IEEE 802.11 n MANET," in *2015 Fifth International Conference on Advanced Computing & Communication Technologies (ACCT)*, 2015, pp. 325–328.

[15] K. H. Mohammadani, H. Kazi, A. Shaikh, I. Channa, and S. Faizullah, "A Comparison of Homogeneous vs Heterogeneous Choice of Routing Protocols in Integrated Wireless Networks," *Eng. Sci. Technol. Int. Res. J.*, vol. 1, no. 3, pp. 44–50, 2017.

[16] A. Shaikh, D. Vasan, and H. Mohammadani, Khalid, "Performance Analysis of MANET Routing Protocols – A Comparative Study," vol. 83, no. 7, pp. 1–29, 2013.

[17] O. Almomani, M. Al-Shugran, J. A. Alzubi, and O. A. Alzubi, "Performance evaluation of position-based routing protocols using different mobility models in manet," *Int. J. Comput. Appl.*, vol. 119, no. 3, 2015.

[18] A. A. Chhachhar, "Location Server Assisted Routing Protocol," Isra University, Hyderabad, Pakistan, 2015.