Functional Test Case Generator for Contiki Operating System and Cooja Simulator

Abhinandan H. Patil, *BITS Pilani, Goa Campus*, *BITS Pilani, Goa Campus*, and Krishnan R, *DCE, Bengaluru*

***Abstract*— Evolving multi-parameter, multi-configuration systems require regression test suite that can be customized. This is in terms of run time. Run time can be customized by generating the combinations using combinatorial techniques. For systems like Contiki operating system, the test cases need to be executed in its simulator Cooja. Executing test cases in a simulator requires functional test cases to be generated from the combinatorial parameter combinations obtained. In this work we present a methodology to generate the functional test cases. We present Functional Test Case Generator for Contiki and Cooja (FTCGCC), which is a tool developed using our methodology. We demonstrate use of our tool by generating customizable regression test suite for Contiki and Cooja using code coverage as criteria. FTCGCC is developed for the test case generation when target System Under Test is IoT operating system Contiki and its simulator Cooja. We find that our tool generates all the test cases. FTCGCC generates the cases which are readily executable in the Contiki and Cooja environment. Further, the approach mentioned can be used in other cases where the simulators are involved which accept the XML based test cases. The design of the FTCGCC can be reused for other simulators. The FTCGCC test case generator engine is generic in nature.**

***Index Terms*— ACTS, CONTIKI, COOJA, CT, CT-RTS, NIST**

I. Introduction

COMBINATORIAL testing (CT) is field of testing which is in practice both in the industry and research [1]. When the System Under Test (SUT) has combination of configurations or input parameters, CT can be used to reduce the number of regression test cases needed [2] [3]. National Institute of Standards and Technology (NIST) gives the tools which aid in performing combinatorial testing. Advanced Combinatorial Testing for Software (ACTS) is one such tool [4]. ACTS tool takes the input parameter and input parameter modeling and generates the test design document which is in the form of rows. Each row is independent test case. For the cases when the test cases are readily executable as in the case of Graphical User Interface (GUI) applications, no intermediate step is required. However, when the test environment expects the test cases in a particular format, intermediate processing is required. For the System Under Test (SUT) as Contiki operating system and its simulator Cooja, the test design contains the test cases which are not readily executable in the SUT test environment. Manual generation of functional test cases is a tedious and error prone task. In this work we present our tool, Functional Test Case Generator which can be used for auto-generation of functional test cases. We demonstrate the successful application of our tool to generate the functional test cases for the Contiki operating system. The novelty of the FTCGCC is that it is a tool developed for the specific requirements of the test case generation for Contiki and Cooja. We evaluated few generic purpose test case generators such as IBM ATG, TCGTool, Randoop, Automatic Testing Platform, Conformiq. These are generic purpose tools. These tools generate the test cases either on the basis of code or requirement. However, they do not meet the requirements of functional test case generation for Contiki and Cooja. We therefore had to develop the FTCGCC from scratch for this work. The FTCGCC takes the text file as input and generates the test cases which are readily executable in Contiki and Cooja based test environment as explained in the subsequent sections of this work. In this work we give the details of the high level design and software implementation of the tool in further sections. The usage of the tool and final results of the test case execution are also documented.

Table 1. Existing test case generation tools

Test case generator	Owner	Remarks
IBM ATG	IBM	Model Based Testing tool
TCGTool	SourceForge	Test Case generation from finite state machines.
Randoop	University of Washinton	Junit test case generator
Automatic Testing	SourceForge	Useful for the web

This paragraph of the first footnote will contain the date on which you submitted your paper for review. It will als o contain support information, including sponsor and financial support acknowledgment. For example, "This work was supported in part by the U.S. Department of Commerce under Grant BS123456."

Abhinandan H. Patil is with the BITS Pilani, Goa, India, since Jan 2014 (e-mail: abhinandan_patil_1414@yahoo.com).
Neena Goveas, is with BITS Pilani, Goa, India. She is with the Department of CS and IS as Professor (e-mail: neena@goa.bits-pilani.ac.in).
Krishnan Rangarajan is with DCE, Bengaluru, Karnataka, India. He is with the CSE Department as Professor (e-mail: krishnanr@gmail.com).

Platform		applications on the client side
Conformiq	Conformiq	Test case generation from graphical model.
Blueprint	Blueprintsys	Test case generation from requirements.

II. Combinatorial Testing And NIST ACTS Tool

For real world software, the number of input parameters and their combinations can be very large making it impractical to develop test cases covering all the combinations of the input parameters. Combinatorial testing addresses this issue partially [2]. NIST offers ACTS covering array generator produces compact arrays covering 2-way to 6-way combinations.
We use the ACTS tool to generate the required combinations to generate a regression test suite. We use as SUT the Contiki operating system and its Cooja simulator. In Section III and IV we explain the details of the SUT chosen and the challenges in developing a practical regression test suite. Subsequent sections explain more about the ACTS tool usage and the details of our proposed tool f or auto generation of the test cases from the ACTS output.

III. Contiki the IoT operating system

Internet of Things (IoT) comprises of things or devices with unique identities that are connected to the internet. The choice of the operating system for the device depends on the purpose of the node. A typical IoT network comprises of expensive nodes and inexpensive nodes. Inexpensive nodes just collect and forward the data to the nearest expensive node. Expensive nodes on the other hand do have few analytics capabilities in addition to the functionalities possessed by the inexpensive nodes. A generic IoT device supports Connectivity, Processor, Audio/Video interfaces, I/O interfaces for sensors and actuators, Memory interfaces and Storage interfaces.

Table 2. Node operating systems

Node type	Example operating system
Inexpensive node	Contiki, RIOT and Tiny OS
Expensive node	Ubuntu core, Arch Linux, Amazon FreeRTOS, Android Things, Rasbian Linux etc

The Operating system will have all these constraints into consideration while being developed. In addition the IoT operating system needs to support the protocols desired. The IoT protocol stack contains mainly four layers:

i. Link layer
ii. Network layer
iii. Transport layer
iv. Application layer

Table 3. IoT layers and protocols

Layer	Protocol examples
Link layer	Ethernet, Bluetooth, Zig bee, Wi-Fi, Wi-max or long range communication protocol such as 3G/LTE/5G etc.
Network layer	IPV4, IPV6, 6 LOW PAN
Transport layer	TCP or UDP
Application layer	HTTP, COAP, Websockets, MQTT, XMPP or AMQP

Contiki, as the device layer OS, supports the protocols mentioned above. Contiki has the functionality implementation for all the above mentioned protocols. To test the protocol functionality a regression test suite needs to contain the corresponding test cases. Contiki operating system needs be tested to ensure that all the claimed platforms are supported by the Contiki operating system. Contiki supports platforms such as Exp5438, z1, wismote, micaz, sky, sentilla-usb and esb among others. In addition, there needs to be test cases to for the other functionality such as file system support etc..

IV. Cooja Simulator

The Cooja simulator in its current form was designed and developed by Fredrik Osterlind. Since then it has evolved by many contributors. The acronym Cooja is derived from Contiki Operating System Java Simulator. Originally the Contiki was developed for resource constrained Wireless Sensor Network (WSN) nodes [6]. The main goal of the Cooja simulator is extensibility. Cooja achieves it using the interfaces and plugins.
Interfaces is for node property such as

- Position of the node
- Button
- Radio transmitter

On the other hand plugin is used for interacting with simulation such as

- To control simulation speed
- To watch all network traffic between the simulated nodes

Since Cooja simulator supports several different simulation environments at a s ame time simulation of HetNets (Heterogeneous Networks) is possible. The advantage of using the Java is that the Java supports JNI (Java Native Interface) using which the simulator can talk to real Contiki operating systems.
The test cases of the Contiki/Cooja are in the form of XML files. These files have *.csc extensions. The test cases typically compile the firmware in the *.c format and embed the firm ware in the Simulator. The firmware compiled depends on the target platform. In this work we focus on the XML files with *.csc extension and their auto-generation for the task at hand.

V. Regression Test Suite and autogeneration of functional test cases

Contiki operating system and its regression test suite is made available publically. The regression test suite has test cases to cover the protocols mentioned in the Table 3 and all other parameters. We did a study of the length of the fuctional test case as the number of parameters increased [5]. We found that an input of all parameters will result in test cases with XML files of length exceeding 1500 lines. We found that using just two parameters at a t ime results in a m anageable length of XML files and does not significantly reduce code coverage. The test cases have an average length of 150-450 XML Lines Of Code (LOC) per test case. There was a need to we develop a tool that auto generates the XML files partially if not complete in all aspects. Our proposed Functional Test Case Generator for Contiki and Cooja (FTCGCC) was developed and was found to be effective in auto generation of XML files. We developed the tool using J2SE. Section VII and VIII elaborate more on the tool code, purpose and usage.

VI. Regression Test Suite Of Contiki Operating System

Contiki team maintains the test suite called regression test suite which is available to the user community. We studied the regression test suite that is publically available and looked at the code coverage aspects of it. We found that there was a need to re-design the test cases to meet required coverage in a short duration, preferably over-night execution [6], [7]. Further, it was noticed that the test cases were concentrated around few hardware (mote) types in the test suite [8]. We used the ACTS tool to generate a combination of parameters to ensure combinatorial coverage. The output of the ACTS tool gave output test cases which were evenly distributed around all the parameters. Further steps were needed to convert this test design into functional test cases.

It was found that the test cases combinations that were output from the ACTS tool needed to be processed to be ready to be inputs to Contiki as SUT. It was noticed that processing step was needed which would convert the test cases output by the ACTS to functional test cases. In addition we found that the base regression test suite of Contiki and Cooja consisted of 64 test cases with incomplete code coverage. There was need to add the additional test cases to the base regression test suite to see how the code coverage varies with the additional test cases using code coverage tools. For this, we used the code coverage tools CodeCover and OpenClover [9] [10] [11].

The design of the test suite was done in the ACTS tool. ACTS tool generated 289 test cases. It was noticed that the 289 test cases were super-set of the base regression test suite. Since the execution time needs to be typically over-night, there was a need to to limit the test cases. Authors studied the code coverage with additional 35 test cases being added.
There exist two types of scenarios as depicted in the Figure 1. when ACTS tool is involved.

- When the test cases are readily executable on the target SUT as in case of GUI SUT
- When the test cases in the ACTS design need to be converted into executable test cases called functional test cases using the intermediate step.

In this latter case, we need a tool to make a co mplete Generation to Execution tool. For Contiki OS we found that each test case requires more than 100 lines of XML code. And for the additional 35 test cases, straight calculation gives more than 3500 lines of XML code. In addition, manually coding further means that the process could be error prone. We looked at possible mechanism for automation of the functional test case generation. Our work has resulted in development of Functional Test Case Generator for Contiki and Cooja (FTCGCC). The tool takes as an input a human readable text file and generates XML files understandable by the Cooja simulator.

VII. Requirements Of The FTCGCC

High level requirements for the tool are as follows:

1. The FTCGCC should take the text file and generate the XML files which are in the *.csc format understandable by Cooja simulator
2. The text file consists of records separated by special characters “(“ and “)”
3. Each record maps to individual test case.
4. Each line in the record separated by special character “{“ and “}” called field within the test case, should map to a logical block of the test case.
5. The input text file could contain any number of records and any number of fields within the records. The parser should be generic enough to handle this.

VIII. High Level Design Of FTCGCC

The tool is designed such that it takes the input text file output of the ACTS tool consisting of records and fields. In the output, the field and field values are separated by special character “,”. If the input file consists of N records, the FTCGCC outputs N test cases. The test case creation is two pass mechanism:

- In the first pass, complete test case except the JavaScript is created using the FTCGCC.
- Since the JavaScript is scenario specific, it needs to be coded manually to suit the scenario under study.
- In the second pass, the JavaScript is embedded manually in the test case created in the first pass.

Figure 1. gives the high-level functionality of the tool. As can be seen from the diagram, tool accepts the text file as input and generates the test cases as per the requirement. If the input file instructs the tool to generate “m” test cases, the tool generates the “m” test cases. Further sections elaborate the tool in more details. As will be explained in the subsequent sections, the test cases are complete in all respect except for the scenario specific JavaScript. JavaScript contains the scenario specific information which needs to be coded using the JavaScript language.

Figure 2. gives the blocks involved in the design of the FTCGCC.

- First the input file is split using the Java's regexp package, which is described in Section 5.1. This step splits the input file in the form of records and fields. Each record maps to one test case and each field within the record maps to logical block within the test case.
- Next, the data structures within the tool are populated as per the parsed input file. These data structures act as driver data. The driver drives the generic engine which is meant for generating the XML files. The Cooja code is reused heavily in the generic engine.

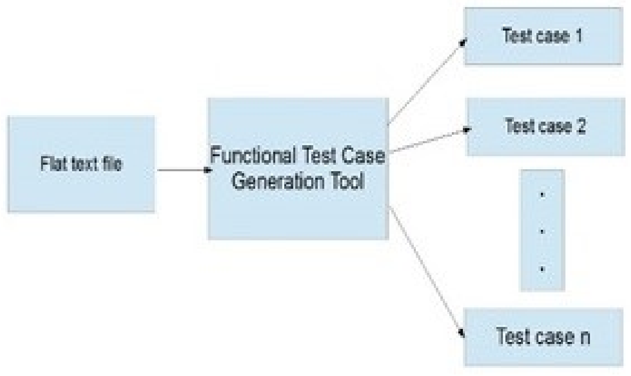


Fig 1. Functionality of FTCGCC

Table 4. Stages and Functionality of FTCGCC

Stage	Description
Stage 1	Creation of input file to FTCGCC
Stage 2	FTCGCC creates records and fields using the Java regexp package
Stage 3	FTCGCC populates the vital data structures which act as driver data
Stage 4	FTCGCC drives the generic engine using the driver data of stage 3.
Stage 5	FTCGCC outputs the skeleton XML files in *.csc format.
Stage 6	The scenario specific JavaScript is embedded in the XML
Stage 7	Fully functional test cases are ready

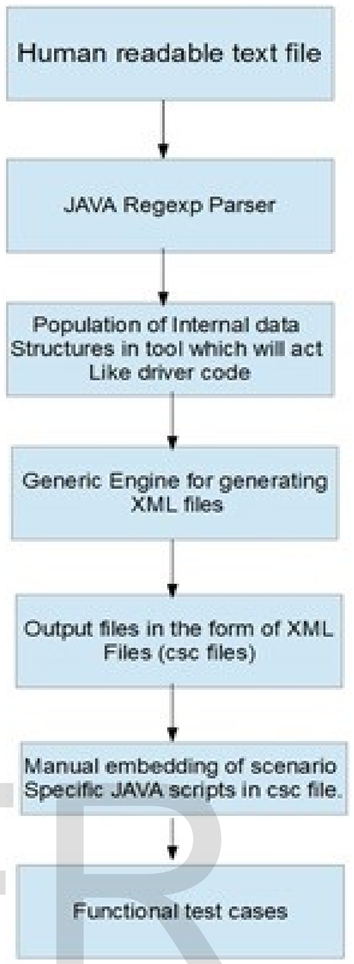

Human readable text file
JAVA Regexp Parser
Population of Internal data Structures in tool which will act Like driver code
Generic Engine for generating XML files
Output files in the form of XML Files (csc files)
Manual embedding of scenario Specific JAVA scripts in csc file.
Functional test cases

Fig 2. Stages and block diagram of FTCGCC

IX. Software Implementation

To design a tool with the taking care of all the requirements, existing *.csc files of the Contiki and Cooja were analyzed. The implementation phase came up with the generic *.csc template as depicted in Figure 3. Vital entities that drive the simulator of the Contiki were observed. In this phase we came up with the template input file to be supplied to the tool and the same is depicted in Figure 4.
The csc which stands for the Cooja simulation configuration file contains the simulation configuration information understandable by the Cooja simulator. The simulation configuration mainly contains blocks that can be classified in the following categories
i) Simulation
ii) Plugins
iii) Scenario specific JavaScript
Simulation block contains radio medium information with mote type (along with firmware information) and mote information. Mote type and mote information is repeatable block.

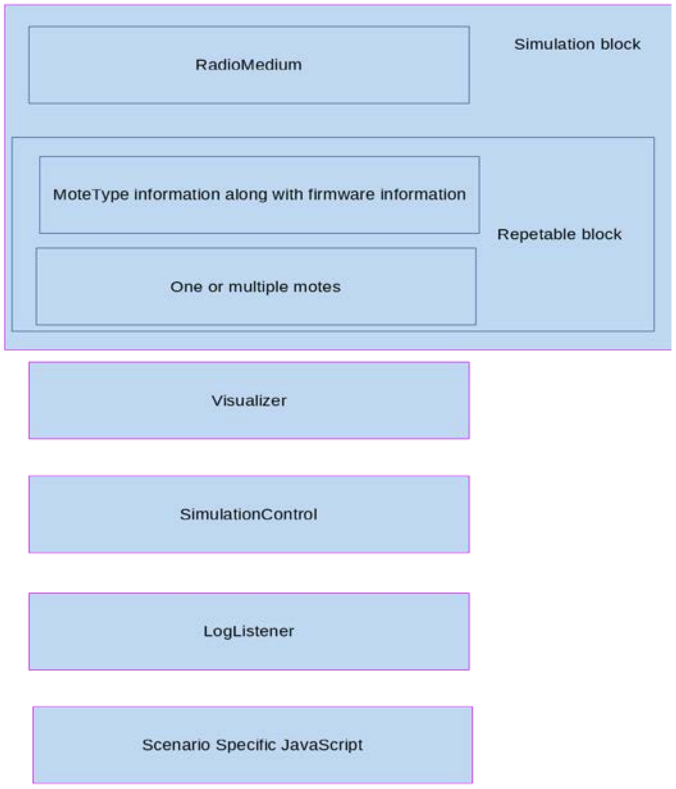


Figure 3. Generic structure of the csc file.

A. *Java's regexp parser*

The java.util.regex package is designed to perform sophisticated pattern matching operations. Regular expression is a s tring of characters describing character sequences. Regular expressions also called patterns can be a s et of characters, wildcard character combinations along with various quantifiers.

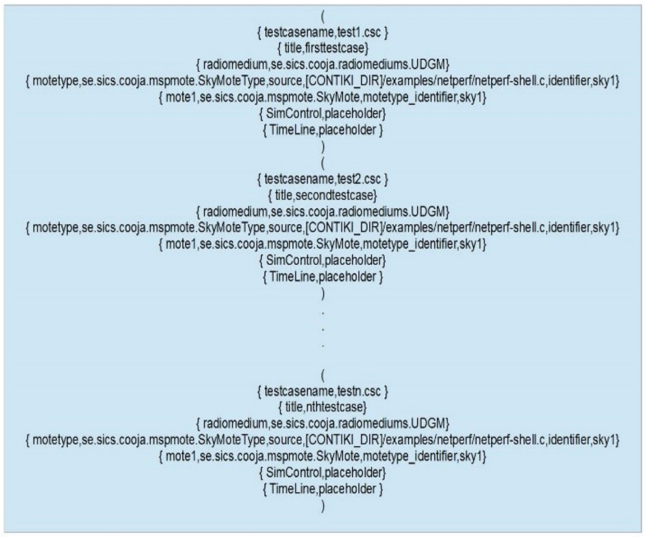
Regexp is a powerful package for text processing. Particularly, following operations can be done very easily with the help of Regexp

- Text processing
- Text manipulation
- Tokenisation

The input file was designed such that:

- Each record which maps to test case starts with special character "(" and ends with ")"
- Each field which maps to logical block in the test case starts with special character "{" and ends with "}"
- Field name and field values are separated by special character ","

We created the patterns for each of the unique character sequences mentioned above and we split the character sequences accordingly.



```
(
{ testcasename,test1.csc }
{ title,firsttestcase}
{ radiomedium,se.sics.cooja.radiomediums.UDGM}
{ motetype,se.sics.cooja.mspmote.SkyMoteType,source,[CONTIKI_DIR]/examples/netperf/netperf-shell.c,identifier,sky1}
{ mote1,se.sics.cooja.mspmote.SkyMote,motetype_identifier,sky1}
{ SimControl,placeholder}
{ TimeLine,placeholder }
)
(
{ testcasename,test2.csc }
{ title,secondtestcase}
{ radiomedium,se.sics.cooja.radiomediums.UDGM}
{ motetype,se.sics.cooja.mspmote.SkyMoteType,source,[CONTIKI_DIR]/examples/netperf/netperf-shell.c,identifier,sky1}
{ mote1,se.sics.cooja.mspmote.SkyMote,motetype_identifier,sky1}
{ SimControl,placeholder}
{ TimeLine,placeholder }
)
.
.
.

(
{ testcasename,testn.csc }
{ title,nthtestcase}
{ radiomedium,se.sics.cooja.radiomediums.UDGM}
{ motetype,se.sics.cooja.mspmote.SkyMoteType,source,[CONTIKI_DIR]/examples/netperf/netperf-shell.c,identifier,sky1}
{ mote1,se.sics.cooja.mspmote.SkyMote,motetype_identifier,sky1}
{ SimControl,placeholder}
{ TimeLine,placeholder }
)
```

Fig 4. Sample input file for the FTCGCC

B. *Java's document object model parser*

In our proposed tool FTCGCC we integrate a third party software Java Document Object Model (JDOM). JDOM was created by Jason Hunter and Brett. JDOM is a m ethod of representing XML document for easy reading/writing and manipulation. JDOM is an open source initiative with Apache style license. JDOM integrates with Document Object Model (DOM) and Simple API for XML (SAX).

JDOM salient features include

- It is light weight
- It can represent full document
- It supports document modification
- It is easy to use

FTCGCC imports Application Programming Interfaces (APIs) of following four classes

- org.jdom.Document
- org.jdom.Element
- org.jdom.output.XMLOutputter
- org.jdom.output.Format

C. *Data structure and functions used in FTCGCC*

Java's ArrayList can dynamically grow or shrink and is a variable length array of object references. Vector is similar to ArrayList, but is synchronized. Table 5. gives the important data structures and functions. Description column details the intended functionality.

Table 5. Important data structures and functions

Entity	Description
private Vector<Mote> motes	Stores the motes in Vector
private Vector<MoteType> moteTypes	Stores the moteTypes in Vector

public static ArrayList<Element> config	Stores the XML elements in ArrayList
public MoteType[] getMoteTypes()	Returns all mote types in simulation
public MoteType getMoteType(String identifier)	Returns mote type with given identifier.
public Collection<Element> getConfigXML()	Returns the current simulation config represented by XML elements.
public boolean setConfigXML()	Sets the current simulation config depending on the given configuration
public void saveSimulationConfig(File file)	Saves the Simulation Config as an XML

X. FTCGCC Usage In Contiki Environment

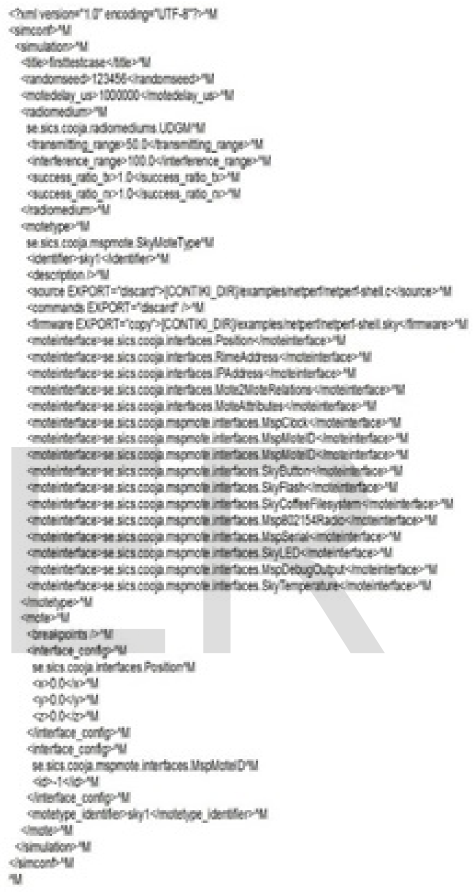
After designing and developing our FTCGCC tool we simplified the steps needed to use it. The procedure for its use are summarized below.

1. github "https://github.com/Abhinandan1414/CoojaTestCaseGeneration", download the content
2. Create an input file with the syntax which adheres to syntax of "GenTest.txt"
3. Set the CLASSPATH to $CLASSPATH:cooja.jar:.
4. Copy the artifacts to directory of your setup
5. Create directory lib and copy jdom.jar jsyntaxpane.jar log4j.jar JDOM_LICENSE JSYNTAXPANE_LICENSE LOG4J_LICENSE
6. Then compile the GenTestcsc.java
7. Run GenTestcsc

The user needs to visit the github. User needs to download the content. An input file with the name GenTest.txt needs to be created with the syntax mentioned. The class path needs to be set to path which reflects current path with the cooja.jar appended. Since the Cooja makes use of third party libraries, the third party libraries need to be copied in a manner explained. The source file GenTestcsc.java needs to be compiled. The last step is running the GenTestcsc.java. The output will be the test cases as instructed by the input file, GenTest.txt.

XI. FTCGCC Usage Results

The tool FTCGCC was successfully used for generating additional test cases to be added to the base regression test suite of Contiki and Cooja and saved approximately 100 lines of XML coding per test case. A sample XML file generated is depicted in the Figure 5. As can be seen from the figure 5, the test case is complete in all aspects except the JavaScript which is embedded manually in the subsequent steps. The file contains the information such as number of motes and mote configuration, radio medium and mote position information. Complete code to be used along with the usage details is in Git hub repository [12]. Final logs of the test cases are kept in the Google docs repository [13]. The directory contains the OpenClover log files which can be readily read in the browser of choice. To begin with the index.html needs to be opened in the browser.



```
<?xml version="1.0" encoding="UTF-8"?>^M
<simconf>^M
 <simulation>^M
  <title>firsttestcase</title>^M
  <randomseed>123456</randomseed>^M
  <motedelay_us>1000000</motedelay_us>^M
  <radiomedium>^M
   se.sics.cooja.radiomediums.UDGM^M
   <transmitting_range>50.0</transmitting_range>^M
   <interference_range>100.0</interference_range>^M
   <success_ratio_tx>1.0</success_ratio_tx>^M
   <success_ratio_rx>1.0</success_ratio_rx>^M
  </radiomedium>^M
  <motetype>^M
   se.sics.cooja.mspmote.SkyMoteType^M
   <identifier>sky1</identifier>^M
   <description />^M
   <source EXPORT="discard">[CONTIKI_DIR]/examples/netperf/netperf-shell.c</source>^M
   <commands EXPORT="discard" />^M
   <firmware EXPORT="copy">[CONTIKI_DIR]/examples/netperf/netperf-shell.sky</firmware>^M
   <moteinterface>se.sics.cooja.interfaces.Position</moteinterface>^M
   <moteinterface>se.sics.cooja.interfaces.RimeAddress</moteinterface>^M
   <moteinterface>se.sics.cooja.interfaces.IPAddress</moteinterface>^M
   <moteinterface>se.sics.cooja.interfaces.Mote2MoteRelations</moteinterface>^M
   <moteinterface>se.sics.cooja.interfaces.MoteAttributes</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.MspClock</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.MspMoteID</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.MspMoteID</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.SkyButton</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.SkyFlash</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.SkyCoffeeFilesystem</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.Msp802154Radio</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.MspSerial</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.SkyLED</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.MspDebugOutput</moteinterface>^M
   <moteinterface>se.sics.cooja.mspmote.interfaces.SkyTemperature</moteinterface>^M
  </motetype>^M
  <mote>^M
   <breakpoints />^M
   <interface_config>^M
    se.sics.cooja.interfaces.Position^M
    <x>0.0</x>^M
    <y>0.0</y>^M
    <z>0.0</z>^M
   </interface_config>^M
   <interface_config>^M
    se.sics.cooja.mspmote.interfaces.MspMoteID^M
    <id>-1</id>^M
   </interface_config>^M
   <motetype_identifier>sky1</motetype_identifier>^M
  </mote>^M
 </simulation>^M
</simconf>^M
^M
```

Figure 5. Sample output of FTCGCC

XII. Conclusion

This work presents tool Functional Test Case Generator for Contiki and Cooja. With this tool it was possible to successfully generate functional test cases. Augmentation of the existing regression test suite was studied in th is work. Using our tool, a user could customize the number of test cases which are added to the existing regression test suite. This will result in a customized generation with the requirement of code coverage, execution time etc. Our tool will reduce the manual effort of manual coding needed for the functional test case generation. Since the manual coding is reduced there was reductions in the test case errors and the manual effort. Further since the test case generation was automated it is possible to re-generate the test cases as per the need of coverage and execution time the test suite. The input file to the tool needs to be modified as per the needs.

The procedure that was followed during the study of the Contiki Operating System and its Cooja simulator can be used with any other multiparameter multi input software. No information other than the publically available data was needed to follow the procedure described in this work and design a suitable tool. In the future we will be looking at other such software and their regression test suites. The study and demonstration in this work can be used to apply the similar techniques for other open source software.

REFERENCES

[1] C. Nie, "Practical Combinatorial Testing," ACM, 2014.
[2] R. Kuhn, Y. Lee and R. N.Kacker, Introduction to Combinatorial Testing, A Chapman and Hall Books, 2013.
[3] R. Kuhn, Y. Lee and R. N. Kacker, "Practical Combinatorial Testing Manual," 2017.
[4] T. ACTS, 2018. [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/index.html.
[5] A. H. Patil, N. Goveas and K. Rangarajan, “"Test Suite Design Methodology Using Combinatorial Approach for Internet of Things Operating Systems," Journal of Software Engineering and Applications,2015, 8, 303-312. doi: 10.4236/jsea.2015.87031
[6] Contiki, "Contiki," 2018. [Online]. Available: http://www.contiki-os.org.
[7] Contiki, "Contiki Supported hardware platforms," 2018. [Online]. Available: http://www.contiki-os.org/hardware.html.
[8] A. H. Patil, N. Goveas and K. Rangarajan, "Re-architecture of Contiki and Cooja Regression Test Suites using Combinatorial Testing Approach," ACM SIGSOFT SEN, 2015.
[9] CodeCover, "CodeCover main," 2018. [Online]. Available: http://codecover.org.
[10] OpenClover, "OpenClover main," 2018. [Online]. Available: http://openclover.org.
[11] A. H. Patil, N. Goveas and K. Rangarajan, "Generating Effective Test Suite for Multiparameter Software using ACTS Tool and its Verification using Code Coverage Tools," IJSER, 2018.
[12] A. H. Patil, N. Goveas and K. Rangarajan, "Test case Autogeneration code Git h ub repository," 2018. [Online]. Available: https://github.com/Abhinandan1414/CoojaTestCaseGeneration.
[13] A. H. Patil, N. Goveas and K. Rangarajan, "Test execution logs," 2018. [Online]. Available: https://drive.google.com/drive/folders/0B2vHzPHgs0nVZWxtSE5sVVdGUmc.
[14] IBM TCATG, “IBM Test Case Auto Generator,”, 2018, [Online]. Available :https://www.ibm.com/support/knowledgecenter/SSB2MU_8.1.3/com.btc.tcatg.user.doc/topics/com.btc.tcatg.atgug.doc.html
[15] TCGTool, “Test Case Generation Tool”, 2018, [Online]. Available https://sourceforge.net/projects/tcgtool/
[16] Randoop, “Randoop Tool”, 2018, [Online]. Available https://randoop.github.io/randoop/
[17] ATestingGP, “Automatic Testing Platform,” 2018, [Online]. Available http://atestingp.sourceforge.net/
[18] Conformiq, “Conformiq Tool,” 2018 , [Online]. Available https://www.conformiq.com/
[19] BlueprintSys, “BluePrintSys,” 2018 [Online]. Available https://www.blueprintsys.com/

Abhinandan H. Patil is a research scholar at BITS Pilani, Goa campus since Jan 2014. Before joining BITS Pilani, Goa he was working in wireless software industry mainly on wireless simulator for telecom network. He has 10+ years of experience in wireless telecom domain. His research interests include Software engineering, Wireless networks, Simulator tools for wireless networks, Verification and validation, AI/ML, Data Science and Cloud computing. His previous industrial exposure spans across companies like Motorola, Kshema Technologies (MPhasis now), Infosys.
Prior to joining BITS Pilani, Goa, he was pursuing M. Tech in Computer Science and Engineering from VTU PG Block Belgaum. Abhinandan holds Bachelor’s degree from Karnataka University Dharwad.

Neena Goveas is with the Department of Computer Science at BITS Pilani K K Birla Goa campus. For her PhD thesis, she worked on "Mean field approaches to thermodynamic properties of magnetic systems" at IIT Bombay, advisor Prof. G. Mukhopadhyay. She worked on INDO-US sponsored project "Development and characterization of materials suitable for magneto-optic Devices" at A. C. R. E., I. I. T. Bombay. She continued her research as a DSTYoung Scientist in a Project entitled "Study of low dimensional magnetic systems" at IIT Guwahati. Other projects she was associated as a PI or Co-PI are “Development of Remotely Configurable Arbitrary Ramp Generator for FMCW Reflectometry, BRNS” and “Implementation of Wireless Sensor Network for Process Monitoring of GAIL’s Pipeline, GAIL India Ltd”. Her main theme of research work is to study complex systems. Using various mean field and computational approaches to understand their properties. Recent research work is on Network Science and its applications to transport, social and computer networks; modeling of Cyber Physical Systems and Wireless Sensor Networks; Construction of test suites for large software systems.

Krishnan Rangarajan Krishnan Rangarajan is Professor in CSE department at Dayanand Sagar College of Engineering, Bangalore. He holds Ph.D from University of Central Florida.
His research interests include Computer Vision, Software engineering topics like testing, usability, security, software architecture.