Journal of Software Engineering and Applications, 2015, 8, 303-312
Published Online July 2015 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2015.87031

Scientific Research Publishing

Test Suite Design Methodology Using Combinatorial Approach for Internet of Things Operating Systems

Abhinandan H. Patil[1], Neena Goveas[1], Krishnan Rangarajan[2]

[1]Department of Computer Science and Information Systems, Birla Institute of Technology and Science, Goa, India
[2]Department of Information Science, Dayanand Sagar College of Engineering, Bangalore, India
Email: P2013408@goa.bits-pilani.ac.in, Neena@goa.bits-pilani.ac.in, Krishnanr1234@gmail.com

Received 28 May 2015; accepted 12 July 2015; published 15 July 2015

Abstract

In this paper we describe how the test design can be done by using the Combinatorial Testing approach for internet of things operating systems. Contiki operating system is taken as a case study but we discuss what can be the approach for RIOT and Tiny OS operating systems. We discuss how the combinatorial coverage measurement can be gathered in addition to the traditional metrics code coverage. The test design generated by using Advanced Combinatorial Testing for Software is analyzed for Contiki operating system. We elaborate the code coverage gathering technique for Contiki simulator which happens to be in Java. We explain the usage of Combinatorial Coverage Measurement tool. Although we have explained the test design methodology for internet of things operating systems, the approach explained can be followed for other open source software.

Keywords

CT, ACTS, CCM, Code Cover, Contiki, Cooja

1. Introduction

Our previously published paper touches upon the test design using combinatorial testing approach for Contiki operating system [1]. In this paper we intend to extend the concept and explain what can be done for the Internet of Things (IoT) operating systems which do not have standard regression test suites viz. RIOT and Tiny OS. We analyze the Advanced Combinatorial Testing for Software (ACTS) generated test suite design and explain how the traditional effective metrics, code coverage can be gathered in addition to more relevant combinatorial cov-

How to cite this paper: Patil, A.H., Goveas, N. and Rangarajan, K. (2015) Test Suite Design Methodology Using Combinatorial Approach for Internet of Things Operating Systems. *Journal of Software Engineering and Applications*, **8**, 303-312.
http://dx.doi.org/10.4236/jsea.2015.87031

erage measurements using combinatorial coverage measurement tool (CCM) .

While we want to have minimal overlap of this paper with the one published already [1], we may re-visit few sections and elaborate more. At places, we get specific and explain implementation part of testing (such as what changes are done to the test setup for gathering the data).

National Institute for Standards and Technology (NIST) has been actively supported combinatorial testing. The research carried out by NIST has been well documented [2] [3]. NIST has made several tools which are available to public [4]. C Nie *et al*. conducts a survey of combinatorial testing and the same is available as part of survey report [5]. Software testing is inarguably essential part of all software development cycles and the same is discussed at high levels by several papers [6]. While few papers discuss software testing techniques at very high level [7], we get specific and discuss combinatorial testing in this paper.

Combinatorial testing is not only adopted by researchers, and evidence shows the adoption by large organizations [8]. Few papers have different perspectives about testing at large [9]. Test suite and test oracles are integral part of testing these days [10]. NIST tools for combinatorial testing take these test suites and test oracles as input entities. NIST tools are an integral part of combinatorial testing and many users have been using them successfully in testing activities of research/industry. ACTs and CCM aid the combinatorial testing immensely. ACTs has been tested with self [11]. CCM's success is documented [12]. While abstracting the formal specifications to generate software tests is an old concept [13], it is being still researched. Few researchers are focusing on different fields of combinatorial testing [14], we in this paper focus on classic approaches of combinatorial testing techniques. A general strategy for t-way software testing is documented in few papers [15] and their through put is documented in few papers [16].

As mentioned earlier, this paper focuses on classic approaches of combinatorial testing and the same is being elaborated in the following sections.

2. Typical Workflow for Base-Lining the Regression Test Suite

Figure 1 depicts the typical flow of work when we want to ascertain the effectiveness of the test suite using combinatorial approach. First step is choosing the operating system for Internet of Things. Then traverse through the source code of the open source code base folders to see if the test suite exists. If it exists gather the coverage data using the code coverage tools.

If the gathered data indicates inadequate test suite, redesign the test suite using the combinatorial approach and gather the data. If the coverage data is less, it calls for re-visiting the test design. Base line the test suite once the adequate test criterion is met.

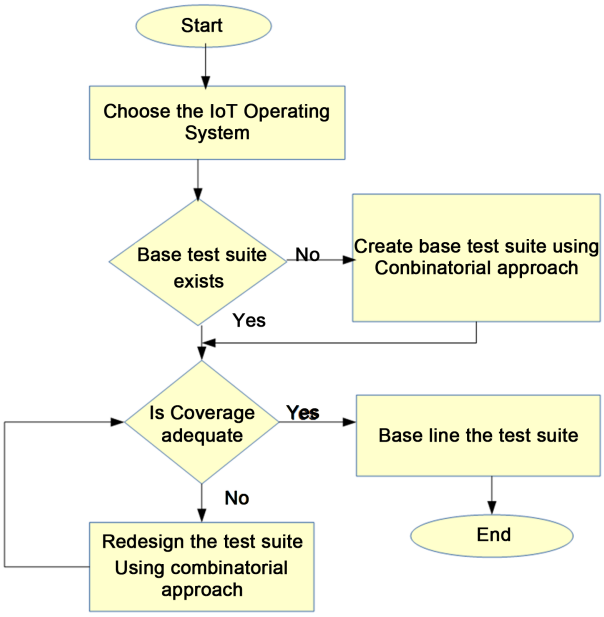

Figure 1. Typical work flow for base lining the test suite.

As can be seen from the diagram it can be iterative process. We document the process that was used for case study operating system in further sections. We describe the approach to be followed when the test suite already exists and when it does not. Section 3 is for the case when the base test suite already exists and Section 4 is for the case when the test suite does not exist.

3. Process of Redesigning the Regression Test Suite If It Already Exists

Figure 2 depicts the process in the case when test suite already exists. This section is for the case when the base lined test suite already exists as in the case of Contiki operating system version 2.7. We can use the either parameter based re-design or configuration based re-design as explained in the book and manual [1] [2] or combination of both. The coverage can be gathered using CCM and traditional coverage tools such as CodeCover. We did preliminary investigation using freely available tool CodeCover for the existing test suite. The coverage was less than 20%. **Appendix B** gives the data gathered using the CodeCover.

Then we visited the existing test suite to know the reason for low coverage. Few areas of improvements were observed in the existing test suite.

- No formal test design document existed.
- It appears that the test cases were concentrated around few mote types (hardware or configurations in the context of combinatorial testing).

We went through the whole regression test suite to extract the configurations supported and input parameters being used. We came up with **Table 1** to be populated in the ACTS test model. When the ACTS was populated using these set of values, the generated test design document is as shown in **Appendix A**.

Contiki Specific Details

Contiki is open source operating system widely used and accepted for Internet of Things. It has base-lined regression test suite for version 2.7. Contiki gives the user friendly operating system in the form of instant Contiki which has Ubuntu like the feel with the tool chains to make the iterative development easy. The developers can use the instant Contiki to test the patches and testers can use the same environment for ascertaining the reliability of the operating system without procuring the hardware for all the mote types.

Contiki gives the simulator which is called Cooja. The Cooja simulator talks to the Contiki using Java Native Interface (JNI). The test cases are called csc files which are understandable by Cooja. We found Eighty three test cases of this type in the regression folder. However, these test cases were concentrated around few mote types.

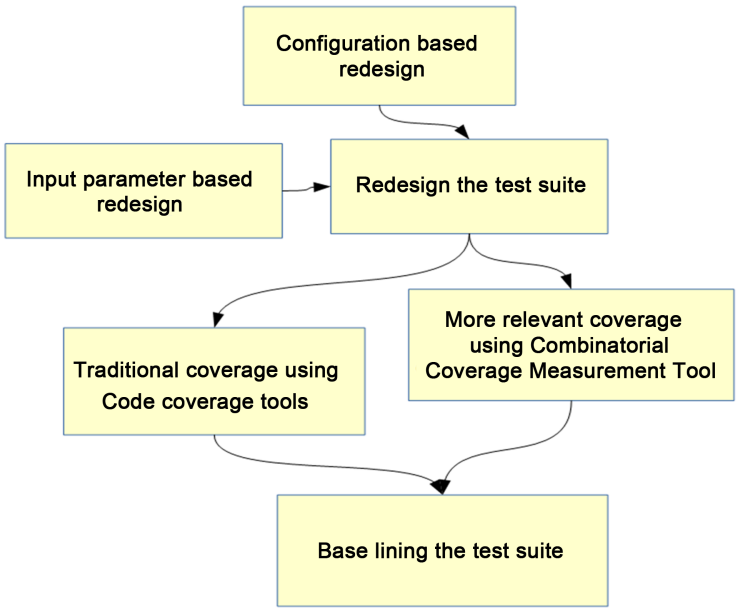

Figure 2. Process of base lining the test suite if it already exists.

Table 1. Acts input parameters.

Parameters	Parameter Values
Platform	Exp5438, z1, wismote, micaz, sky, jcreate, sentilla-usb, esb, native, cooja
base	Multithreading, coffee, check pointing
Rime	collect, rucb, deluge, runicast, trickle, mesh
Net Performance	Net Perf, Net Perf-lpp, Net Perf-cxmac
collect	shell-collect, shell-collect-lossy
ipv4	telnet-ping, webserver
ipv6	ipv6-udp, udp-fragmentation, unicast-fragmentation, ipv6-rpl-collect
RPL	up-root, root-reboot, large-network, up and down routes, temporary rootloss, random rearrngement, rpl-dao
ipv6apps	servreg-hack, coap

As already mentioned, **Appendix A** gives the test design generated using ACTS for **Table 1** input. Let us visit the column 2 of the design. We can see that the generated test cases are spread across the mote (hardware) types. Further, the generated test design takes care of the input parameters as well for the test cases.

Now the task at hand is mapping these generated test cases to functional test cases (xml files called csc) which are understandable by Cooja and gathering the coverage data again. The coverage data should improve in principle. We are working on this.

4. Process of Designing the Regression Test Suite If It Does Not Exist

Figure 3 depicts the case when test suite does not exist. This process is more suited for operating systems which do not have standard regression test suite viz. RIOT and TinyOS. Since the functional specification and test design are both missing in case of these operating systems, we will have to come up with the functional specification document first. This will be our understanding of the functionality that these operating systems support. Once the functionality of these operating systems is understood we will have to come up with the test design. Configuration to be supported and input parameters to be supplied for each test case will act as starting point for populating the ACTS test model. Once test design is generated, we will have to understand the test environment for these operating systems and the test design need to be mapped to functional test cases to be executed for gathering the coverage data. The CCM coverage will not be appropriate as the test cases generated using ACTS tool will always give 100% combinatorial coverage. Traditional coverage such as code coverage may be handy.

5. Contiki Environment Specific Changes to Be Done

In this section we document the changes that we did in the Contiki environment for the tasks at hand. Since we get implementation specific for case study operating system, this section can be conveniently skipped by the readers who are not interested in specific details for given operating system.

1. Log in as user in the instant Contiki environment.
2. Search for the .travis. yml
3. Add the build type you are interested in:

- BUILD_TYPE = “ipv6-apps”
- BUILD_TYPE = “CT”

-BUILD_TYPE = “compile-8051-ports”

4. Under the directory../contiki-2.7/regression-tests create a folder 02-CT
5. Under contiki-2.7/regression-tests/02-CT directory create *.csc files you are interested in viz. 01-custom.csc 02-custom.csc
6. The Make file should look like include../Makefile. simulation-test
7. Create a 01-custom.csc file in the Cooja tool. Use the test script editor to create a java script which will be essential while running the test case from command line using the makefile.

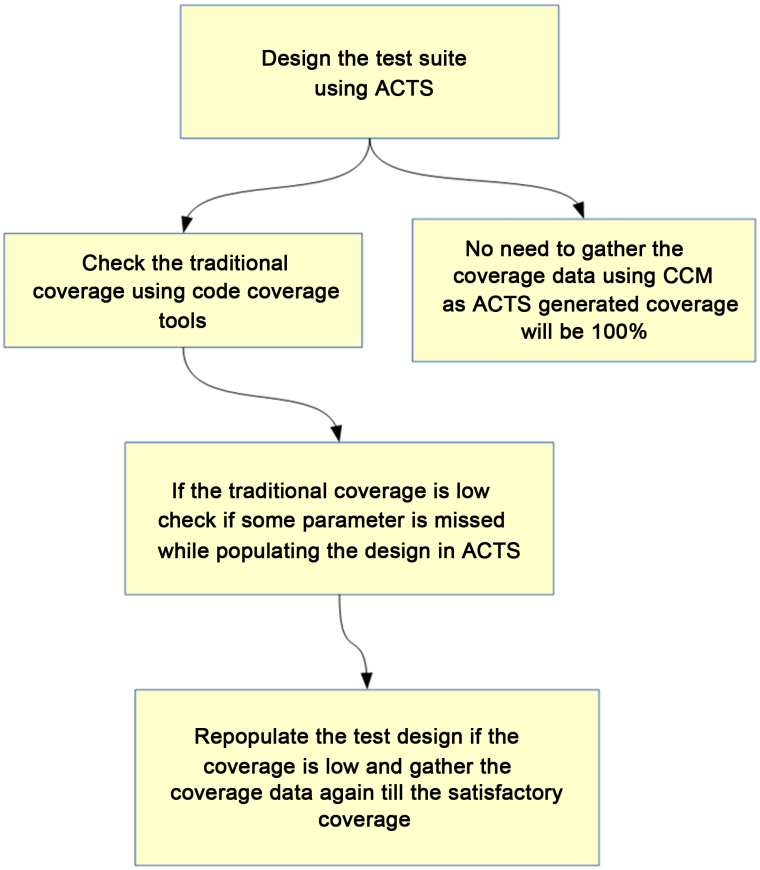

Figure 3. Process of base-lining the test suite if it does not exist.

8. Modify the build.xml suitably as explained in **Appendix C**.
9. Run the regression test suite as usual.
10. Test run will create many *.clf files.
11. Create a script for analyze, merge and generate report.

6. Conclusion

In this paper we presented the approaches that could be employed for designing the regression test suite using combinatorial approach. We explained how the bench marking of the regression test suite could be done using the traditional approaches such as code coverage in addition to coverage gathered using combinatorial coverage measurement tools.

7. Future Work

In this paper we investigated how the combinatorial approach could be applied for the cases when the regression test suite already existed viz. Contiki case. We were working on ascertaining the gain due to Combinatorial testing.

We were planning to explore the Combinatorial testing for the cases when the regression test suite did not exist viz. RIOT and TinyOS.

Further we were planning to model the System Under Test (SUT) and use New Symbolic Model Verifier (NuSMV) in conjunction with ACTs.

References

[1] Patil, A.H., Goveas, N. and Rangarajan, K. (2015) Re-architecture of Contiki and Cooja Regression Test Suite Using

Combinatorial Testing Approach. *ACM SIGSOFT Software Engineering Notes*, **40**, 1-3.

[2] Richard Kuhn, D., Kacker, R.N. and Lei, Y. (2013) Introduction to Combinatorial Testing. Text Book.

[3] Richard Kuhn, D., Kacker, R.N. and Lei. Y. (2013) Practical Combinatorial Testing Manual. NIST Special Publications 800-142.

[4] NIST. http://csrc.nist.gov/groups/SNS/acts/index.html

[5] Nie, C., *et al*. (2011) A Survey of Combinatorial Testing. ACM Computing Surveys, Vol. 43, No. 2, Article 11.

[6] Ammann, P. and Offutt, J. (2008) Introduction to Software Testing. Cambridge University Press, New York. http://dx.doi.org/10.1017/CBO9780511809163

[7] Beizer, B. (1990) Software Testing Techniques. 2nd Edition, Van Nostrand Reinhold, New York.

[8] ASTQB (2014) Introducing Combinatorial Testing in Large Organizations.

[9] Bach, J. and Shroeder, P. (2004) Pairwise Testing—A Best Practice That Isn't. *Proceedings of* 22*nd Pacific Northwest Software Quality Conference*, Portland, 180-196.

[10] Baresi, L. and Young, M. (2001) Test Oracles. Department of Computer and Information Science, University of Oregon, Eugene. http://www.cs.uoregon.edu/michal/pubs/oracles.html

[11] NourozBorazjany, M., Yu, L., Lei, Y., Kacker, R.N. and Kuhn, D.R. (2012) Combinatorial Testing of ACTS: A Case Study. 2012 *IEEE* 5*th International Conference on Software Testing, Verification and Validation*, 17-21 April 2012, Montreal, 971.

[12] Combinatorial Coverage Measurement NASA IV&V Workshop, 11-13 September 2012.

[13] Ammann, P. and Black, P.E. (1999) Abstracting Formal Specifications to Generate Software Tests via Model Checking. *Proceedings of* 18*th Digital Avionics Systems Conference*, **2**, 10.A.6.1-10.A.6.10. http://dx.doi.org/10.1109/dasc.1999.822091

[14] Kuhn, D.R., Higdon, J.M., Lawrence, J.F., Kacker, R.N. and Lei, Y. (2012) Combinatorial Methods for Event Sequence Testing. 5*th International Workshop on Combinatorial Testing*, Montreal, 17-21 April 2012, 601-609.

[15] Lei, Y. (2007) IPOG—A General Strategy for t-Way Software Testing. 14*th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, Tucson, 26-29 March 2007, 549-556.

[16] Price, C., Kuhn, R., *et al*. (2013) Evaluating the t-Way Technique for Determining the Thoroughness of a Test Suite. *NASA IV&V Workshop*, Fairmont, September 2013.

Appendix A: ACTS generated test design for Contiki Operating System

Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10
Test Case#	Platform	base	Rime	NetPerformanc	collect	ipv4	ipv6	RPL	ipv6apps
0	Exp5438	coffee	rucb	NetPerf-lpp	shell-collect-lossy	webserver	udp-fragmentation	up-root	coap
1	Exp5438	checkpointing	deluge	NetPerf-cxmac	shell-collect	telnet-ping	unicast-fragmentatic	root-reboot	servreg-hack
2	Exp5438	Multithreading	runicast	NetPerf	shell-collect-lossy	telnet-ping	ipv6-rpl-collect	large-network	coap
3	Exp5438	coffee	trickle	NetPerf-cxmac	shell-collect	webserver	ipv6-udp	upanddownroutes	servreg-hack
4	Exp5438	checkpointing	mesh	NetPerf	shell-collect	webserver	udp-fragmentation	temporaryrootloss	coap
5	Exp5438	Multithreading	collect	NetPerf-lpp	shell-collect	webserver	unicast-fragmentatic	randomrearrngement	servreg-hack
6	Exp5438	checkpointing	rucb	NetPerf-cxmac	shell-collect-lossy	telnet-ping	ipv6-rpl-collect	rpl-dao	servreg-hack
7	z1	Multithreading	deluge	NetPerf	shell-collect-lossy	telnet-ping	ipv6-udp	up-root	coap
8	z1	coffee	runicast	NetPerf-lpp	shell-collect	webserver	udp-fragmentation	root-reboot	servreg-hack
9	z1	checkpointing	trickle	NetPerf-lpp	shell-collect-lossy	telnet-ping	unicast-fragmentatic	large-network	coap
10	z1	Multithreading	mesh	NetPerf-cxmac	shell-collect-lossy	telnet-ping	ipv6-rpl-collect	upanddownroutes	coap
11	z1	coffee	collect	NetPerf	shell-collect-lossy	telnet-ping	ipv6-udp	temporaryrootloss	servreg-hack
12	z1	checkpointing	rucb	NetPerf	shell-collect	telnet-ping	ipv6-udp	randomrearrngement	coap
13	z1	Multithreading	deluge	NetPerf-lpp	shell-collect	webserver	udp-fragmentation	rpl-dao	coap
14	wismote	checkpointing	runicast	NetPerf-cxmac	shell-collect	webserver	unicast-fragmentatic	up-root	servreg-hack
15	wismote	Multithreading	trickle	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	root-reboot	coap
16	wismote	coffee	mesh	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	large-network	servreg-hack
17	wismote	checkpointing	collect	NetPerf-cxmac	shell-collect-lossy	telnet-ping	udp-fragmentation	upanddownroutes	coap
18	wismote	Multithreading	rucb	NetPerf	shell-collect-lossy	telnet-ping	unicast-fragmentatic	temporaryrootloss	coap
19	wismote	coffee	deluge	NetPerf-lpp	shell-collect-lossy	webserver	ipv6-rpl-collect	randomrearrngement	servreg-hack
20	wismote	coffee	runicast	NetPerf	shell-collect-lossy	telnet-ping	ipv6-udp	rpl-dao	coap
21	micaz	checkpointing	trickle	NetPerf-cxmac	shell-collect	webserver	udp-fragmentation	up-root	servreg-hack
22	micaz	coffee	mesh	NetPerf-lpp	shell-collect-lossy	telnet-ping	unicast-fragmentatic	root-reboot	coap
23	micaz	Multithreading	collect	NetPerf-cxmac	shell-collect	webserver	ipv6-rpl-collect	large-network	coap
24	micaz	checkpointing	rucb	NetPerf	shell-collect	telnet-ping	ipv6-udp	upanddownroutes	coap
25	micaz	checkpointing	deluge	NetPerf-lpp	shell-collect-lossy	webserver	ipv6-rpl-collect	temporaryrootloss	servreg-hack
26	micaz	checkpointing	runicast	NetPerf-cxmac	shell-collect	telnet-ping	udp-fragmentation	randomrearrngement	servreg-hack
27	micaz	checkpointing	trickle	NetPerf-cxmac	shell-collect	webserver	unicast-fragmentatic	rpl-dao	servreg-hack
28	sky	coffee	mesh	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
29	sky	checkpointing	collect	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
30	sky	Multithreading	rucb	NetPerf-cxmac	shell-collect	webserver	udp-fragmentation	large-network	servreg-hack
31	sky	checkpointing	deluge	NetPerf-lpp	shell-collect-lossy	telnet-ping	unicast-fragmentatic	upanddownroutes	servreg-hack
32	sky	checkpointing	runicast	NetPerf-cxmac	shell-collect	telnet-ping	unicast-fragmentatic	temporaryrootloss	coap
33	sky	checkpointing	trickle	NetPerf-lpp	shell-collect	webserver	ipv6-rpl-collect	randomrearrngement	coap
34	sky	coffee	mesh	NetPerf-lpp	shell-collect-lossy	telnet-ping	ipv6-rpl-collect	rpl-dao	coap
35	jcreate	coffee	collect	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
36	jcreate	checkpointing	rucb	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
37	jcreate	Multithreading	deluge	NetPerf-cxmac	shell-collect	webserver	udp-fragmentation	large-network	coap
38	jcreate	Multithreading	runicast	NetPerf-lpp	shell-collect	webserver	unicast-fragmentatic	upanddownroutes	servreg-hack
39	jcreate	checkpointing	trickle	NetPerf-lpp	shell-collect-lossy	webserver	ipv6-udp	temporaryrootloss	coap
40	jcreate	coffee	mesh	NetPerf-cxmac	shell-collect	telnet-ping	ipv6-udp	randomrearrngement	servreg-hack
41	jcreate	checkpointing	collect	NetPerf-cxmac	shell-collect	webserver	unicast-fragmentatic	rpl-dao	servreg-hack
42	sentilla-usb	coffee	rucb	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
43	sentilla-usb	checkpointing	deluge	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
44	sentilla-usb	Multithreading	runicast	NetPerf-cxmac	shell-collect-lossy	webserver	udp-fragmentation	large-network	coap
45	sentilla-usb	coffee	trickle	NetPerf	shell-collect-lossy	webserver	unicast-fragmentatic	upanddownroutes	coap
46	sentilla-usb	coffee	mesh	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	temporaryrootloss	coap
47	sentilla-usb	Multithreading	collect	NetPerf	shell-collect-lossy	telnet-ping	ipv6-udp	randomrearrngement	coap
48	sentilla-usb	Multithreading	collect	NetPerf-cxmac	shell-collect	webserver	unicast-fragmentatic	rpl-dao	servreg-hack
49	esb	coffee	rucb	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
50	esb	checkpointing	deluge	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
51	esb	Multithreading	runicast	NetPerf-cxmac	shell-collect-lossy	webserver	udp-fragmentation	large-network	servreg-hack
52	esb	Multithreading	trickle	NetPerf	shell-collect-lossy	webserver	unicast-fragmentatic	upanddownroutes	servreg-hack
53	esb	coffee	mesh	NetPerf-lpp	shell-collect	webserver	ipv6-rpl-collect	temporaryrootloss	servreg-hack
54	esb	Multithreading	collect	NetPerf	shell-collect	webserver	ipv6-udp	randomrearrngement	coap
55	esb	coffee	trickle	NetPerf-cxmac	shell-collect	telnet-ping	udp-fragmentation	rpl-dao	servreg-hack
56	native	coffee	rucb	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
57	native	checkpointing	deluge	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
58	native	Multithreading	runicast	NetPerf-cxmac	shell-collect	webserver	udp-fragmentation	large-network	servreg-hack
59	native	coffee	trickle	NetPerf-lpp	shell-collect-lossy	webserver	unicast-fragmentatic	upanddownroutes	coap
60	native	coffee	mesh	NetPerf-cxmac	shell-collect	webserver	unicast-fragmentatic	temporaryrootloss	servreg-hack
61	native	Multithreading	collect	NetPerf-cxmac	shell-collect	telnet-ping	ipv6-udp	randomrearrngement	coap
62	native	checkpointing	rucb	NetPerf-cxmac	shell-collect-lossy	webserver	ipv6-rpl-collect	rpl-dao	coap
63	cooja	coffee	rucb	NetPerf	shell-collect-lossy	webserver	ipv6-rpl-collect	up-root	servreg-hack
64	cooja	checkpointing	deluge	NetPerf-lpp	shell-collect	telnet-ping	ipv6-udp	root-reboot	coap
65	cooja	Multithreading	runicast	NetPerf-cxmac	shell-collect	telnet-ping	udp-fragmentation	large-network	servreg-hack
66	cooja	Multithreading	trickle	NetPerf	shell-collect-lossy	telnet-ping	unicast-fragmentatic	upanddownroutes	servreg-hack
67	cooja	coffee	mesh	NetPerf-cxmac	shell-collect-lossy	telnet-ping	ipv6-udp	temporaryrootloss	coap
68	cooja	Multithreading	collect	NetPerf-lpp	shell-collect-lossy	telnet-ping	ipv6-rpl-collect	randomrearrngement	coap
69	cooja	checkpointing	collect	NetPerf	shell-collect	webserver	udp-fragmentation	rpl-dao	coap

Appendix B: Code Coverage data gathered for existing test suite of Contiki and Cooja using CodeCover

BITS Goa Campus Int | Facebook | cooja - TestSessionFi

file:///C:/Users/abhinandan/Desktop/SimpleJavaAppReport.html-files/se/sics/cooja/index.html

up to the title page

se . sics . cooja

	Statement Coverage		Branch Coverage		Loop Coverage		Strict Condition Coverage	
contikimote	102 / 747	13 %	40 / 323	12 %	7 / 99	7 %	11 / 191	5 %
dialogs	162 / 2130	7 %	45 / 907	4 %	15 / 225	6 %	6 / 483	1 %
interfaces	79 / 421	18 %	40 / 178	22 %	8 / 48	16 %	10 / 100	10 %
plugins	327 / 4875	6 %	83 / 2091	3 %	16 / 620	2 %	19 / 1127	1 %
motes	11 / 148	7 %	6 / 80	7 %	0 / 27	0 %	4 / 43	9 %
util	14 / 379	3 %	11 / 256	4 %	3 / 114	2 %	3 / 145	2 %
radiomediums	169 / 353	47 %	85 / 195	43 %	53 / 120	44 %	31 / 115	26 %
emulatedmote	16 / 135	11 %	0 / 52	0 %	0 / 3	0 %	0 / 26	0 %
positioners	0 / 125	0 %	0 / 38	0 %	0 / 0	---	0 / 19	0 %
HasQuickHelp	0 / 0	---	0 / 0	---	0 / 0	---	0 / 0	---
MoteInterfaceHandler	19 / 42	45 %	14 / 46	30 %	3 / 33	9 %	6 / 23	26 %
CoreComm	8 / 57	14 %	6 / 32	18 %	1 / 18	5 %	0 / 10	0 %
Plugin	0 / 0	---	0 / 0	---	0 / 0	---	0 / 0	---
RadioMedium	1 / 1	100 %	0 / 0	---	0 / 0	---	0 / 0	---
ContikiError	0 / 3	0 %	0 / 0	---	0 / 0	---	0 / 0	---
GUI	357 / 1259	28 %	211 / 787	26 %	43 / 207	20 %	45 / 419	10 %
ConvertedRadioPacket	2 / 2	100 %	0 / 0	---	0 / 0	---	0 / 0	---
COOJARadioPacket	0 / 1	0 %	0 / 0	---	0 / 0	---	0 / 0	---
AbstractionLevelDescription	0 / 0	---	0 / 0	---	0 / 0	---	0 / 0	---
MoteMemory	0 / 0	---	0 / 0	---	0 / 0	---	0 / 0	---
RadioConnection	19 / 28	67 %	4 / 14	28 %	0 / 0	---	0 / 7	0 %
SectionMoteMemory	0 / 70	0 %	0 / 24	0 %	0 / 21	0 %	0 / 17	0 %
COOJAProject	8 / 21	38 %	1 / 28	3 %	0 / 6	0 %	0 / 16	0 %
RadioPacket	0 / 0	---	0 / 0	---	0 / 0	---	0 / 0	---
AddressMemory	0 / 1	0 %	0 / 0	---	0 / 0	---	0 / 0	---
SimEventCentral	46 / 68	67 %	17 / 36	47 %	11 / 45	24 %	4 / 22	18 %

Links 10:18 PM 9/26/2014

Appendix C: Tweaking of Ant Build.xml for Gathering the Coverage Data

```
<?xml version="1.0"?>
<project name="COOJA Simulator" default="run" basedir=".">
<property name="java" location="java"/>
.
.
<property name="args" value=""/>
<property name="codecoverDir"
value="/home/user/Desktop/CodeCover/codecover-batch-1.0/lib"/>
<property name="sourceDir" value="/home/user/contiki-2.7/tools/cooja/java"/>
<property name="instrumentedSourceDir" value="instrumented"/>
<property name="mainClassName" value="se.sics.cooja.GUI"/>
<taskdef name="codecover" classname="org.codecover.ant.CodecoverTask"
classpath="${codecoverDir}/codecover-ant.jar"/>
<target name="clean">
<delete>
<fileset dir="." includes="*.clf"/>
</delete>
<delete file="codecover.xml"/>
<delete file="report.html"/>
<delete dir="report.html-files"/>
</target>
```

```
<target name="instrument-sources" depends="clean">
<codecover>
<instrument containerId="c" language="java"
destination="${instrumentedSourceDir}" charset="utf-8"
copyUninstrumented="yes">
<source dir="${sourceDir}">
<include name="**/*.java"/>
</source>
</instrument>
<save containerId="c" filename="codecover.xml"/>
</codecover>
</target>
<target name="compile-instrumented" depends="instrument-sources">
<javac srcdir="${instrumentedSourceDir}" destdir="${instrumentedSourceDir}"
encoding="utf-8" target="1.7" debug="true"
classpath="${codecoverDir}/lib/codecover-instrumentationjava.
jar:/home/user/contiki-2.7/tools/cooja/lib/log4j.jar:/home/user/contiki-
2.7/tools/cooja/lib/jdom.jar:/home/user/contiki-2.7/tools/cooja/lib/jsyntaxpane.jar"
includeAntRuntime="false"></javac>
</target>
<target name="run-instrumented" depends="compile-instrumented, copy configs">
<java classpath="${instrumentedSourceDir}:${codecoverDir}/lib/codecoverinstrumentation-
java.jar:/home/user/contiki-
2.7/tools/cooja/lib/log4j.jar:/home/user/contiki-
2.7/tools/cooja/lib/jdom.jar:/home/user/contiki-2.7/tools/cooja/lib/jsyntaxpane.jar"
fork="true" failonerror="true" classname="${mainClassName}">
<jvmarg value="-Dorg.codecover.coverage-log-file=test.clf"/>
</java>
</target>
<target name="create-report" >
<codecover>
<load containerId="c" filename="codecover.xml"/>
<analyze containerId="c" coverageLog="*.clf" name="Test Session"/>
<save containerId="c" filename="codecover.xml"/>
<report containerId="c" destination="report.html"
template="/home/user/Desktop/CodeCover/codecover-batch-1.0/reporttemplates/
HTML_Report_hierarchic.xml">
<testCases>
<testSession pattern=".*">
<testCase pattern=".*"/>
</testSession>
</testCases>
</report>
</codecover>
</target>
<target name="help">
<echo>
.
.
<target name="copy configs" depends="init">
<mkdir dir="${build}"/>
<copy todir="/home/user/contiki-2.7/tools/cooja/instrumented">
<fileset dir="${config}"/>
```

```
</copy>
.
.
.
<target name="jar_cooja" depends="init, compile, copy configs, compile instrumented
">
<mkdir dir="${dist}"/>
<jar destfile="${dist}/cooja.jar" base dir="/home/user/contiki-
2.7/tools/cooja/instrumented">
<manifest>
<attribute name="Main-Class" value="se.sics.cooja.GUI"/>
<attribute name="Class-Path" value=". lib/log4j.jar lib/jdom.jar
lib/jsyntaxpane.jar"/>
</manifest>
</jar>
<mkdir dir="${dist}/lib"/>
<copy todir="${dist}/lib">
<fileset dir="${lib}"/>
</copy>
</target>
</project>
```