UNIVERSITY
OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Aleksi Klasila

Mbed OS Regression Test Selection and Optimization

Master's Thesis
Degree Programme in Computer Science and Engineering
4 2019

Klasila A. (2019) Mbed OS Regression Test Selection and Optimization. University of Oulu, Degree Programme in Computer Science and Engineering. Master's Thesis, 39 p.

ABSTRACT

Testing is a fundamental building block in the identification of bugs, errors and defects in both hardware and software. Effective testing of large projects requires automated testing, test selection and test optimization. Using CI (Continuous Integration) tools, and test selection and optimization techniques reduce development time and increase productivity. The prioritization, selection and minimization of tests are well-known problems in software testing.

Arm Mbed OS is a free, open-source embedded operating system designed specifically for the "things" in the IoT (Internet of Things). This thesis researches regression test selection (RTS) and optimization techniques (RTO). The main focus of the thesis is to develop a set of effective automated safe RTS (mbedRTS) and RTO (mbedRTO) techniques for Mbed OS pull request (PR) testing. This thesis refers to the set of developed techniques as Mbed OS regression test techniques (MbedRTT), also known as Mbed OS Smart Tester.

The empirical analysis of the researched, and developed MbedRTT techniques show promising results. Several developed MbedRTT techniques have already been adopted in Mbed OS Jenkins CI.

Keywords: Software testing, Test selection, Test optimization

Klasila A. (2019) Mbed OS -regressiotestien valinta ja optimointi. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 39 s.

TIIVISTELMÄ

Testaus on olennainen tekijä vikojen ja virheiden tunnistamisessa sekä ohjelmistossa että laitteistossa. Isojen projektien tehokas testaaminen vaatii automaattista testausta, testien valintaa ja testien optimointia. Jatkuvan integraation (engl. continuous integration) työkalut, testien valintatekniikat ja testien optimointitekniikat lyhentävät kehitykseen kuluvaa aikaa ja kasvattavat tuottavuutta. Testien priorisointi, valinta ja minimointi ovat tunnettuja ongelmia ohjelmistotestauksessa.

Arm Mbed OS on ilmainen avoimen lähdekoodin sulautettu käyttöjärjestelmä, joka on tarkoitettu erityisesti "asioille" asioiden Internetissä (engl. Internet of Things). Tässä työssä tutkitaan regressiotestauksen valinta- ja optimointimenetelmiä. Tämän työn päätehtävä on kehittää tehokkaita ja turvallisia valinta- (mbedRTS) ja optimointimenetelmiä (mbedRTO) Mbed OS pull request:ien regressiotestaukseen. Mbed OS -regressiotestausmenetelmillä (MbedRTT) viitataan tässä työssä kehitettyihin regressiotestausmenetelmiin, jotka tunnetaan myös nimellä Mbed OS älykäs testaaja (engl. Mbed OS Smart Tester).

Tutkittujen ja kehitettyjen MbedRTT-tekniikoiden empiirisen analyysin tulos näyttää lupaavalta. Mbed OS Jenkins CI:ssä on jo otettu käyttöön useita kehitettyjä MbedRTT-tekniikoita.

Avainsanat: Ohjelmiston testaus, Testien valinta, Testien optimointi

TABLE OF CONTENTS

FOREWORD

As a former member of Mbed OS test team, it was my job to maintain and develop Mbed OS testing systems and tools. My specific area of responsibility was Mbed OS Jenkins CI maintenance and development. The main goal of Mbed OS Jenkins CI is to test and report Mbed OS GitHub repository PR's and branches. As a part of my area of responsibility, I developed and researched tools and techniques for enabling stable and effective Mbed OS GitHub repository PR testing.

I would like to thank ARM Finland Oy for the opportunity to work on the Mbed OS project. I would also like to thank Olli-Pekka Puolitaival, Juha Röning (Dr. Tech., Professor) and others for the invaluable feedback and suggestions.

Oulu, 4.4.2019

Klasila Aleksi

ABBREVIATIONS

OS	**O**perating **S**ystem
CI	**C**ontinuous **I**ntegration
HW	**H**ard**w**are
SW	**S**oft**w**are
AWS	**A**mazon **W**eb **S**ervices
AWS S3	**A**mazon **S**imple **S**torage **S**ervice
AWS EC2	**A**mazon **E**lastic **C**ompute **C**loud
Mbed OS	Open-source e**mbed**ded **o**perating **s**ystem
Greentea	**G**eneric **re**gression **en**vironment for **te**st **a**utomation.
I/O	**I**nput/**O**utput
Stdout	**St**an**d**ard **out**put
JSON	**J**avaScript **O**bject **N**otation
CL	**C**hange **L**ist
TAP	Google's **T**est **A**utomation **P**latform
RT	**R**egression **T**esting
RTS	**R**egression **T**est **S**election
RTO	**R**egression **T**est **O**ptimization
RTP	**R**egression **T**est **P**rioritization
PR	**P**ull **R**equest
DUT	**D**evice **U**nder **T**est
MbedRTT	**Mbed** OS **R**egression **T**est **T**echniques
MbedRTS	**Mbed** OS **R**egression **T**est **S**election techniques

MbedRTO	**Mbed** OS **R**egression **T**est **O**ptimization techniques
GreenteaRTS	Mbed OS **Greentea R**egression **T**est **S**election technique
TargetRTS	Mbed OS **Target R**egression **T**est **S**election technique
JobRTS	Mbed OS **Job R**egression **T**est **S**election technique
SHA-1	**S**ecure **H**ash **A**lgorithm **1**
IDE	**I**ntegrated **D**evelopment **E**nvironment

1. INTRODUCTION

Embedded systems are electronically controlled systems where hardware and software are combined. Embedded systems are expensive to develop and test. Embedded systems testing ensures an embedded system to be as defect-free as possible. Most embedded systems are real time. In hard real time systems, if constraints are not met, a system crash could be of consequence. In a soft real time system, if constraints are not met, no crash will be of consequence. [7]

Mbed OS supports deterministic, multithreaded real time software execution [5]. Mbed OS embedded systems crash often requires hardware reset: power off and on.

Regression testing (RT) is a crucial, but potentially time consuming, part of software development [15]. Testing is a critical part of the development process of any embedded system [7]. Testing tools enable the testing of embedded systems. Mbed OS is active and relatively large project. There were on average 116 weekly commits during the past year (Fig. 1). Mbed OS includes tools and software test applications for functional unit testing, and integration testing for complex use cases [10]. Mbed OS testing challenges include, but are not limited to, Mbed OS supporting a wide range of target/toolchain combinations, and target hardware wearing down from frequent flashing.

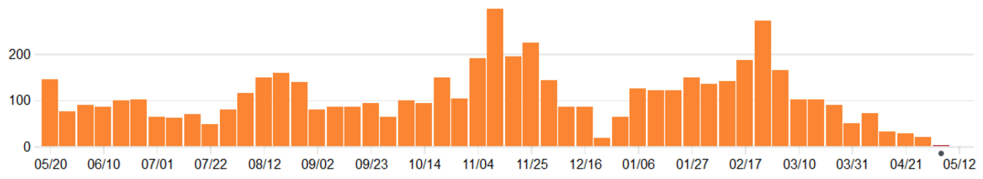


Figure 1: Mbed OS weekly commits.

Automated RT is an important part of the modern embedded systems software development testing process. The automated testing of large embedded systems projects requires automated RTS techniques; Only modifying documentation files should result in only testing the documentation files, only modifying the dependencies of a subset of all tests should result in only executing the affected tests.

Mbed OS embedded systems software testing includes exporting and building a set of test and example applications, and executing the built test applications on a set of targets [10]. Mbed OS tools [3] execute the test applications on the targets and interpret the results from standard output (Stdout). Mbed OS PR testing uses Generic regression environment for test automation (Greentea) for executing test applications on the targets [6]. Mbed OS test application execution using Greentea consists of resetting a target, flashing the target with a test application, executing the test application on the target, optionally controlling the execution of the test application on the target, and reporting the results of the test application. Although, embedded systems target hardware can be and often is simulated, physical target hardware still plays a significant role in embedded systems testing. Mbed OS PR’s are mostly tested on physical target hardware.

1.1. Overview of Mbed OS PR testing

Mbed OS PR’s are merged only if all reviewers approve the PR, Mbed OS Travis CI tests pass, and Mbed OS Jenkins CI tests pass (Fig. 2). Mbed OS Travis CI is usually executed before Mbed OS Jenkins CI. Mbed OS Travis CI executes light test cases, such as: build an example, and check style and documentation. Mbed OS uses public Travis CI, so test results and documentation are publicly available. This thesis does not focus on Mbed OS Travis CI testing. This thesis focuses on Mbed OS Jenkins CI testing. Mbed OS Jenkins CI refers to the Jenkins CI used in Mbed OS testing. Mbed OS Jenkins CI executes a more comprehensive set of tests, including but not limited to, building and executing test applications on a wide range of targets. Mbed OS uses internal Jenkins CI. [10]

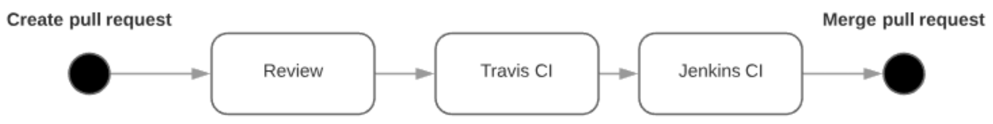
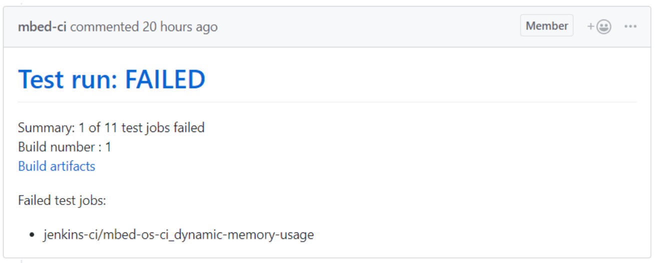


Figure 2: Simplified Mbed OS pull request state diagram

The current Mbed OS Jenkins CI was taken into use as a part of the Mbed OS testing process in the autumn of 2018. Mbed OS GitHub repository includes a Jenkinsfile. When a Mbed OS PR is tested in Mbed OS Jenkins CI, the Jenkinsfile is used [10]. Executing the Jenkinsfile implicitly loads a private Jenkins shared library and calls the main method. The main method checkouts the Mbed OS PR, builds subjobs (partially in parallel), and reports the results to the PR in GitHub as a comment (Fig. 3). Mbed OS Jenkins CI stores public results in Amazon Simple Storage Service (AWS S3) [18]. Mbed OS Jenkins CI stores private results and other private build information in various databases. Mbed OS Jenkins CI uses Amazon Elastic Compute Cloud (AWS EC2) for building and executing tests. The current Mbed OS Jenkins CI tests and reports most PR’s as isolated individual changes. Mbed OS and Mbed OS Jenkins CI demand for compute resources has been growing slowly making testing of each code change feasible (Fig. 1). Currently, Mbed OS Jenkins CI uses the main RTS and RTO techniques developed in this thesis. The inner workings of Mbed OS Jenkins CI are out of the scope of this thesis as MbedRTT does not depend on Mbed OS Jenkins CI.


mbed-ci commented 20 hours ago

Member

Test run: FAILED

Summary: 1 of 11 test jobs failed
Build number : 1
Build artifacts

Failed test jobs:

- jenkins-ci/mbed-os-ci_dynamic-memory-usage

Figure 3: Mbed OS Jenkins CI report in GitHub.

1.1.1. A brief description of Mbed OS Flaky Tests

Various Mbed OS tests depend on the Internet and other resources that are hard to control. Most Mbed OS tests that depend on resources that are hard to control are normally disabled in PR testing. Currently, Mbed OS Greentea tests are executed on physical target hardware. Between executing subsequent Greentea test suites, the targets are software reset. If for instance a Greentea test suite hard crashes a target, the software reset may not always fully reset the target, resulting in the next Greentea test suite execution failing. If a failing / flaky target is detected, the target is in most cases automatically set into maintenance, either manually or automatically hardware reset, and released back into the test system. Mbed OS flaky ("unreliable") tests and targets are still an issue, especially when the tools that reduce test flakiness fail to perform as expected.

1.1.2. An Empirical Analysis of Mbed OS PR content and testing

Over 30% of all Mbed OS PR's were never tested in Mbed OS Jenkins CI, or at least the results of the tests were never reported to the PR on GitHub. A large portion of the PR's for which the tests were never reported on GitHub were either created too long time ago (*issue number smaller than #2000 by at least a few hundred*) or tested as a part of a collection of PR's (PR's manually collected / merged into one large PR and tested all at the same time). Very few PR's were tested more than three times. The most common reason for a PR "requiring" multiple rounds of Mbed OS Jenkins CI testing (build and test Mbed OS) is "real" bugs in the PR modifications. A small percentage of the tested PR's fail due to instabilities in the testing system itself. When a PR test fails due to an unstable testing system, rebuilding only the failed tests is enough most of the time. (Fig. 4, Fig. 6)

On average, the more diverse modifications (documentation, source code, ...) a PR contains, the more times the PR is tested in Mbed OS Jenkins CI. PR's with only documentation changes are tested very rarely more than once in Mbed OS Jenkins CI. PR's with modifications to tests, resources and *targets.json*, for example, are tested relatively frequently at least twice. (Fig. 4)

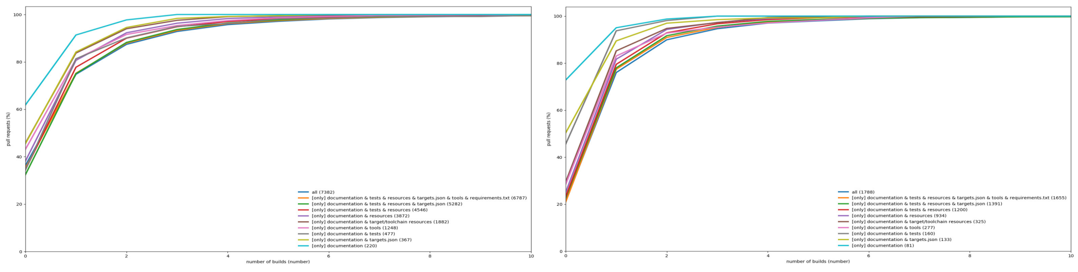


Figure 4: Integral of the percentage of the number of Mbed OS PR's over the number of Mbed OS Jenkins CI build rounds. The figure on the left includes all PR's. The figure on the right includes the last 1788 PR's (at the time).

Most Mbed OS PR’s modify only resources (a large subset of files a build depends on), *target.json* (defines supported target/toolchain combinations and their default configurations), documentation (files that should never affect Mbed OS Jenkins CI test verdict), tools (python tools used for building and testing Mbed OS) and tests (unit tests, Greentea tests, and Icetea tests). (Fig. 5)

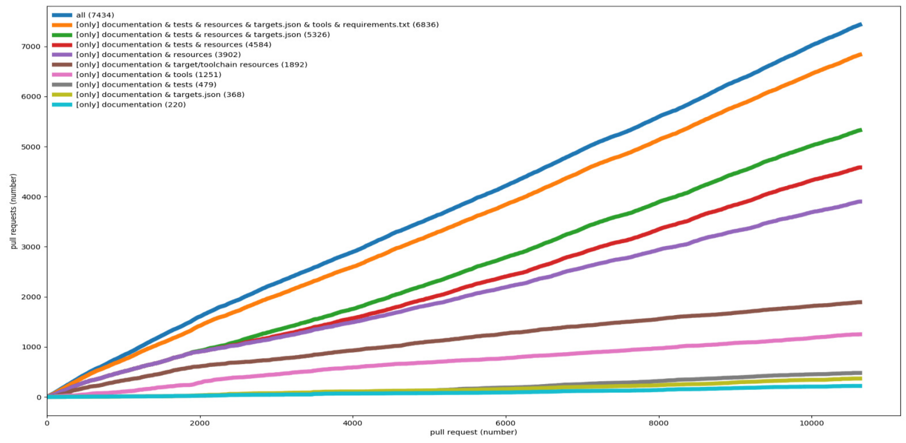


Figure 5: Integral of the number of Mbed OS PR’s over PR numbers.

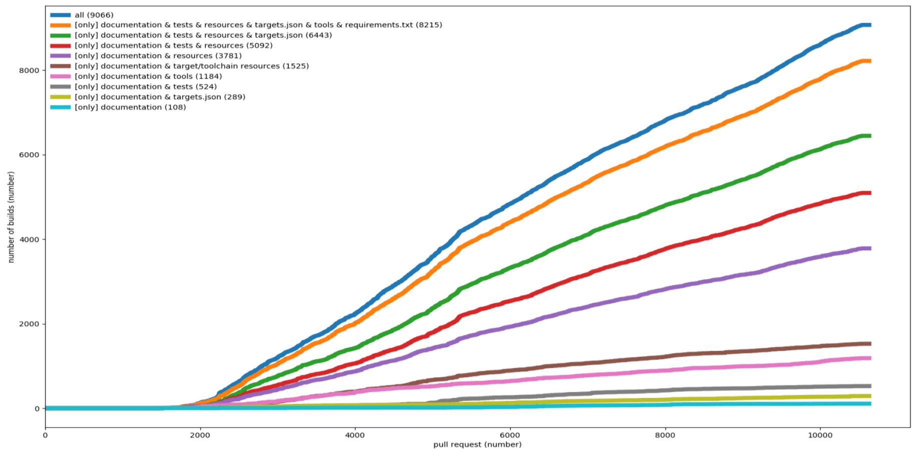
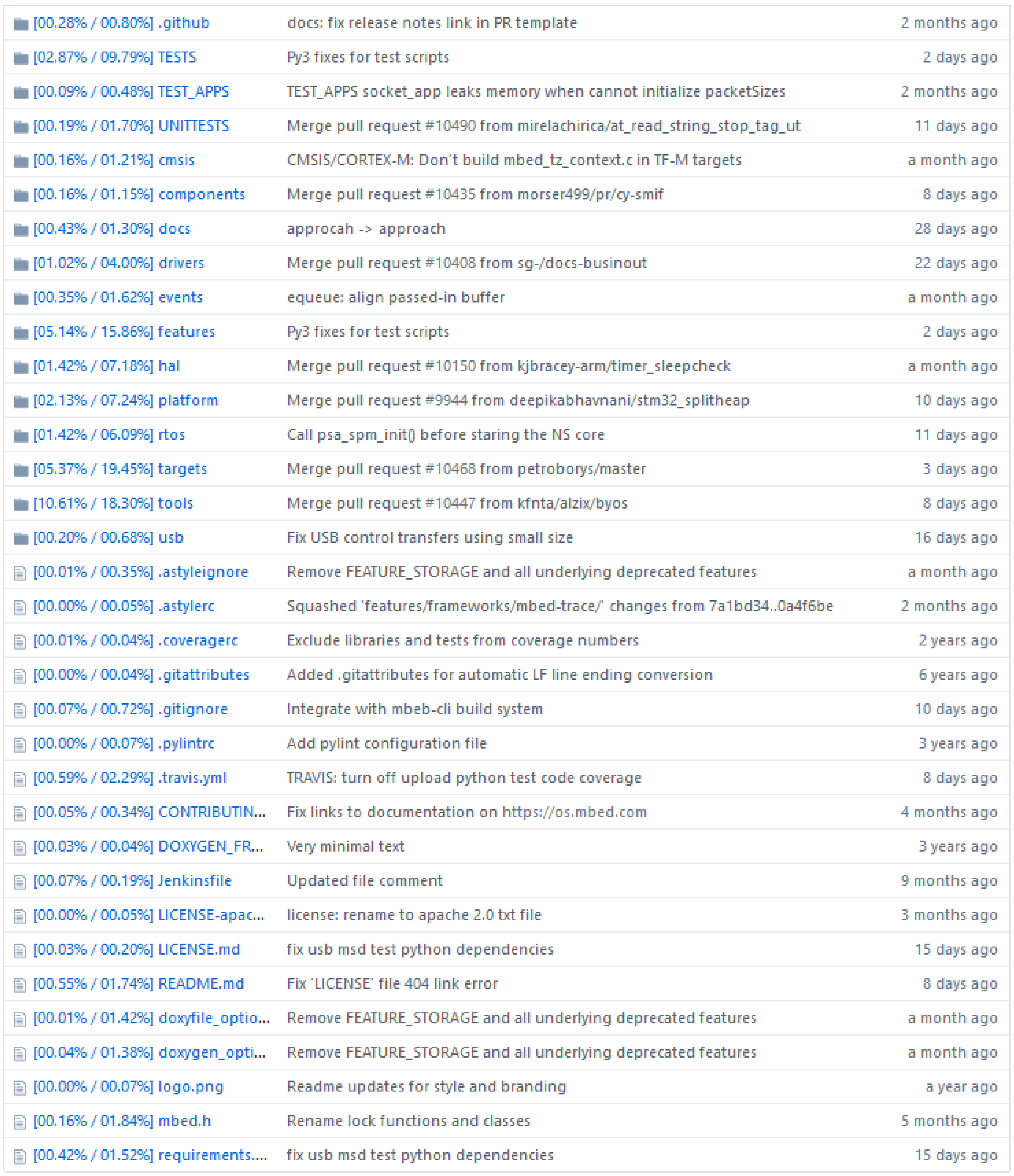


Figure 6: Integral of the number of Mbed OS Jenkins CI builds over PR number.

Over 50% of all Mbed OS PR’s modify only documentation and resources. All the files inside target/toolchain specific folders are included in target/toolchain specific resources/documentation. Any file in the target/toolchain specific folders can only affect the corresponding targets/toolchains. The set of target/toolchain specific resources/documentation is a subset of all resources/documentation. Resources is (at least in the scope of this thesis) considered a set which contains all target/toolchain specific resources for all target/toolchain combinations combined with common resources. Of all the publicly available Mbed OS tests in the Mbed OS GitHub repository, this thesis mostly only focuses on Greentea tests. Greentea tests are located inside the *TESTS* directories. Currently, Unit tests and Icetea tests combined constitute a very small part of the overall Mbed OS Jenkins CI testing. (Fig. 7)



[00.28% / 00.80%] .github	docs: fix release notes link in PR template	2 months ago
[02.87% / 09.79%] TESTS	Py3 fixes for test scripts	2 days ago
[00.09% / 00.48%] TEST_APPS	TEST_APPS socket_app leaks memory when cannot initialize packetSizes	2 months ago
[00.19% / 01.70%] UNITTESTS	Merge pull request #10490 from mirelachirica/at_read_string_stop_tag_ut	11 days ago
[00.16% / 01.21%] cmsis	CMSIS/CORTEX-M: Don't build mbed_tz_context.c in TF-M targets	a month ago
[00.16% / 01.15%] components	Merge pull request #10435 from morser499/pr/cy-smif	8 days ago
[00.43% / 01.30%] docs	approcah -> approach	28 days ago
[01.02% / 04.00%] drivers	Merge pull request #10408 from sg-/docs-businout	22 days ago
[00.35% / 01.62%] events	equeue: align passed-in buffer	a month ago
[05.14% / 15.86%] features	Py3 fixes for test scripts	2 days ago
[01.42% / 07.18%] hal	Merge pull request #10150 from kjbracey-arm/timer_sleepcheck	a month ago
[02.13% / 07.24%] platform	Merge pull request #9944 from deepikabhavnani/stm32_splitheap	10 days ago
[01.42% / 06.09%] rtos	Call psa_spm_init() before staring the NS core	11 days ago
[05.37% / 19.45%] targets	Merge pull request #10468 from petroborys/master	3 days ago
[10.61% / 18.30%] tools	Merge pull request #10447 from kfnta/alzix/byos	8 days ago
[00.20% / 00.68%] usb	Fix USB control transfers using small size	16 days ago
[00.01% / 00.35%] .astyleignore	Remove FEATURE_STORAGE and all underlying deprecated features	a month ago
[00.00% / 00.05%] .astylerc	Squashed 'features/frameworks/mbed-trace/' changes from 7a1bd34..0a4f6be	2 months ago
[00.01% / 00.04%] .coveragerc	Exclude libraries and tests from coverage numbers	2 years ago
[00.00% / 00.04%] .gitattributes	Added .gitattributes for automatic LF line ending conversion	6 years ago
[00.07% / 00.72%] .gitignore	Integrate with mbeb-cli build system	10 days ago
[00.00% / 00.07%] .pylintrc	Add pylint configuration file	3 years ago
[00.59% / 02.29%] .travis.yml	TRAVIS: turn off upload python test code coverage	8 days ago
[00.05% / 00.34%] CONTRIBUTIN...	Fix links to documentation on https://os.mbed.com	4 months ago
[00.03% / 00.04%] DOXYGEN_FR...	Very minimal text	3 years ago
[00.07% / 00.19%] Jenkinsfile	Updated file comment	9 months ago
[00.00% / 00.05%] LICENSE-apac...	license: rename to apache 2.0 txt file	3 months ago
[00.03% / 00.20%] LICENSE.md	fix usb msd test python dependencies	15 days ago
[00.55% / 01.74%] README.md	Fix 'LICENSE' file 404 link error	8 days ago
[00.01% / 01.42%] doxyfile_optio...	Remove FEATURE_STORAGE and all underlying deprecated features	a month ago
[00.04% / 01.38%] doxygen_opti...	Remove FEATURE_STORAGE and all underlying deprecated features	a month ago
[00.00% / 00.07%] logo.png	Readme updates for style and branding	a year ago
[00.16% / 01.84%] mbed.h	Rename lock functions and classes	5 months ago
[00.42% / 01.52%] requirements....	fix usb msd test python dependencies	15 days ago

Figure 7: Mbed OS GitHub repository root directory files and directories. The percentages refer to the percentages of PR's that included only modifications to the corresponding files and folders and PR's that included at least modifications to the corresponding files and folders.

1.2. A brief description of the MbedRTT approach

This thesis researches the current state of the art techniques for automated RTS. This thesis describes a set of RTS and RTO techniques developed for Mbed OS RT. The Mbed OS RTS and RTO techniques are also known as MbedRTT, Mbed OS Smart Tester, or MbedRTS and MbedRTO. The developed techniques aim to significantly reduce unnecessary testing (especially on physical target hardware), and speed up the necessary testing. Currently, Mbed OS Jenkins CI uses three of the developed RTS techniques: JobRTS, GreenteaRTS, and TargetRTS. The developed RTS techniques are also known as Job Selector, Greentea Selector, and Target Toolchain Selector.

JobRTS selects required jobs based on the names of the files modified in a PR and a JSON configuration file (JobRTS config file). The main goal of JobRTS is to have Mbed OS Jenkins CI test nothing for PR's with only documentation changes and only test Greentea/unit tests for PR's with only Greentea/unit test modifications. The current JobRTS config file was generated manually. This thesis will not implement automated JobRTS config file generation, however automated JobRTS config file generation would likely only result in a much more effective usage of JobRTS. The currently used combination of JobRTS and JobRTS config file affects less than 10% of PR's.

TargetRTS selects required target/toolchain combinations based on the intersection of names of the modified files in a PR and the names of the resource files of all Mbed OS 5 supported target/toolchain combinations. The current revision of TargetRTS is activated only if a PR modifies nothing but documentation and resource files. Currently, TargetRTS affects at least to a degree slightly over 50% of PR's.

GreenteaRTS selects Greentea test suites (test applications / test binaries) based on test execution history and inherent binary dependency information. If a Greentea test suite with the same binary file hash has passed previously, it can be skipped. Currently, GreenteaRTS affects over 90% of PR's.

1.3. Related research

A multitude of studies on regression test prioritization, selection and minimization have been conducted in an attempt to reduce unnecessary testing and speed up necessary testing. The previous studies include studies on both static and dynamic regression test techniques. This chapter describes some of the most relevant (in the context of Mbed OS RT) studies.

1.3.1. Google's Test Automation Platform (TAP)

The vast majority of Google's software assets are stored in a single, shared repository. Google chose the monolithic-source-management strategy in 1999 when the existing Google codebase was migrated from CVS to Perforce. The size of the Google codebase has grown exponentially. The Google codebase includes approximately one billion files and has a history of approximately 35 million commits spanning Google's entire 18-year existence. The repository contains 86TB of data, including approximately two billion lines of code in nine million unique source files. [11]

In an average day, Google's Test Automation Platform (TAP) system integrates and tests more than 13K code projects, requiring 800K builds and 150 Million test runs. In the past TAP tried to test each code change, but found that the (demand for) compute resources were growing quadratically with two multiplicative linear factors: (1) the code submission rate which (for Google) has been growing roughly linearly and (2) the size of the test pool which also has been growing linearly. This caused unsustainable demand for compute resources, hence TAP invented a mechanism to slow down one of the linear factors by breaking a TAP day into a sequence of epochs called milestones, each of which integrates and tests a snapshot of Google's codebase. [12]

More than 99% of all tests run by the TAP system pass or flake. Test history of 5.5 Million affected tests in a given time period, only 63K ever failed; the rest never failed even once. Test targets, which are essentially buildable and executable code units labeled as tests in meta BUILD files, that are more than a distance of 10 (in terms of number of dependency edges) from the changed code hardly ever break. Test targets that directly or indirectly depend on the modified files are called AFFECTED test targets. TAP needs to execute AFFECTED test targets to ensure that the latest changes did not cause breakages. TAP executes AFFECTED test targets only at their latest affecting CL (Change List) because there will be overlap in how test targets are affected across CLs in a milestone. Probability of a test target transitioning from a previously known PASSED to FAILED (or FAILED to PASSED) when executed at the CL consistently increases with file modification frequency and number of unique users. [12]

1.3.2. Ekstazi: Lightweight Test Selection

The inputs to a traditional RTS technique are two software revisions (new and old), and the dependency information from the test runs on the old revision. EKSTAZI, a lightweight RTS tool tracks the dynamic dependencies of tests on files and requires no integration with version-control systems. EKSTAZI does not explicitly compare the old and new revisions. Instead, EKSTAZI computes for each test class what files it depends on. A test class need not be run in the new revision if none of its dependent files changed. [15]

A typical RTS technique has three phases: the analysis (A) phase selects what tests to run in the current revision, the execution (E) phase runs the selected tests, and the collection (C) phase collects information for the next revision. EKSTAZI checks if the checksums of all dependent files are still the same. If so, the test class is not selected. If the checksum of any dependent file of a test class differs or does not exist, the class is selected. The Execution phase executes the selected tests. The collection phase creates the dependency files for the executed test classes. [15]

1.3.3. An Empirical Analysis of Flaky Tests

Test outcomes are not reliable for tests that can intermittently pass or fail even for the same code version. A flaky test outcome is non-deterministic with respect to a given software version. The most common approach to combat flaky tests is to run a flaky test multiple times, and if it passes any run, declare it passing, even if it fails in several other runs. Although the current approaches used to deal with flaky tests may alleviate their impact, they are more "workarounds" rather than solutions, and may even *hide* real bugs. [17]

The study on the following section studied only a subset of all software projects, so the results may not generalize. The top three categories of flaky tests are Async Wait (the test execution makes an asynchronous call and does not properly wait for the result), Concurrency (the test non-determinism is due to different threads interacting in a non-desirable manner, excluding Async Wait), and Test Order Dependency (test outcome depends on the order in which the tests are run). Most tests are flaky from the first time they were written. Most flaky tests categorized have an outcome that does not depend on the platform. Fixing a flaky test without correctly identifying and understanding the root cause may not completely solve the problem. Common fixes for flaky tests include, but are not limited to: replace sleep with waiting for something, use deterministic code instead of non-deterministic code, use locks and guard conditions where required, a clean state between tests and remove test order dependencies. [17]

1.3.4. An Extensive Study of Static Regression Test Selection in Modern Software Evolution

RTS can be broadly split into dynamic and static techniques. A typical dynamic RTS technique computes test dependencies dynamically while running the tests. A typical static RTS technique computes test dependencies statically before running tests. On average, typically, static RTS techniques tend to select a higher percentage of tests than dynamic RTS techniques. While the number of selected tests is an important internal metric in RTS, the time taken for testing is the relevant external metric because a developer using RTS perceives it based on this time. Often, mostly depending on the level of granularity, static RTS techniques are faster to execute than dynamic RTS techniques. Often, depending on the level of granularity, non-determinism, and real-time constraints, dynamic RTS techniques are safer and more precise than static RTS techniques. RTS techniques provide more benefits for projects with longer-running tests. [13]

1.3.5. Dynamic Integration Test Selection Based on Test Case Dependencies

The result of any test case depends on the results of one (itself) or more test cases. Test cases can be prioritized and ordered based on the extent to which they are redundant given that other test cases fail. If one or more of the test cases a test case depends on, fail, the test case can, in some cases, be considered failed without having to execute it. [14]

2. MbedRTT

The developed MbedRTS techniques are safe RTS techniques. Safe MbedRTS techniques refer to a set of safe Mbed OS RTS techniques which aim to select all tests that may be affected by code changes (Fig. 8). If a MbedRTS technique is not extremely close to absolutely certain that not all tests are required, it selects all tests. If a MbedRTS technique is extremely close to absolutely certain that only a subset of tests is required, it selects only the required tests. The current MbedRTT techniques do not include any regression test prioritization (RTP) techniques.

Mbed OS Jenkins CI uses the testing history, and the content and names of the files modified in a PR to selectively skip tests which are either known to have passed previously or are known not to affect the test behaviour and outcome. Each Mbed OS Jenkins CI test (such as a job) is only affected by a subset of Mbed OS GitHub repository files; For instance, hardware and unit tests are not required when only documentation is modified. Mbed OS Jenkins CI tests nothing if only documentation is modified. Mbed OS Jenkins CI tests only Greentea tests if only Greentea tests and documentation are modified. Mbed OS Jenkins CI tests only specific target/toolchain combinations if only the resources of specific target/toolchain combinations are modified. Each Mbed OS hardware test application depends only on a subset of the files in the Mbed OS GitHub repository, modifying some Mbed OS GitHub repository files may require all test applications to be executed, but modifying test specific files requires only the affected tests to be executed.

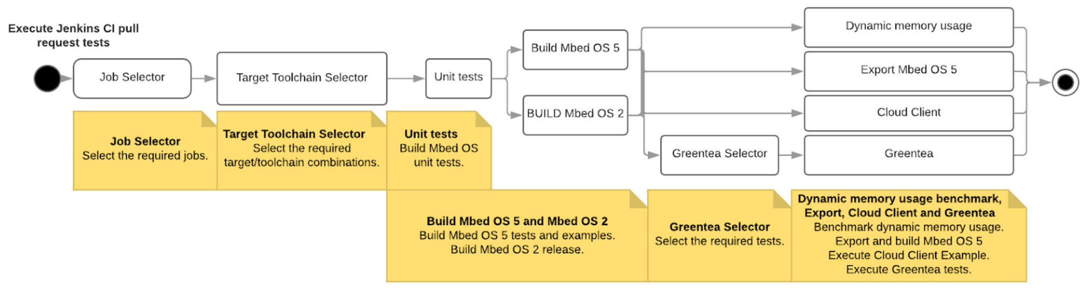

Execute Jenkins CI pull request tests
Job Selector
Target Toolchain Selector
Unit tests
Build Mbed OS 5
BUILD Mbed OS 2
Greentea Selector
Dynamic memory usage
Export Mbed OS 5
Cloud Client
Greentea
Job Selector
Select the required jobs.
Target Toolchain Selector
Select the required target/toolchain combinations.
Unit tests
Build Mbed OS unit tests.
Build Mbed OS 5 and Mbed OS 2
Build Mbed OS 5 tests and examples.
Build Mbed OS 2 release.
Greentea Selector
Select the required tests.
Dynamic memory usage benchmark, Export, Cloud Client and Greentea
Benchmark dynamic memory usage.
Export and build Mbed OS 5
Execute Cloud Client Example.
Execute Greentea tests.

Figure 8: Mbed OS Smart Tester integration in Mbed OS Jenkins CI

2.1. Tools, requirements and dependencies

MbedRTT depends on Mbed OS tools. Mbed OS tools are implemented in Python. Python is a programming language that lets you work quickly and integrate systems more effectively [1]. MbedRTS techniques are implemented in Python for easy and effective integration with the existing Mbed OS tools.

Currently, Mbed OS PR's are tested in Mbed OS Jenkins CI. Currently, MbedRTT is loosely integrated in Mbed OS Jenkins CI; Mbed OS Jenkins CI uses MbedRTT by executing groovy scripts which execute python commands in a shell. MbedRTT does not depend on Mbed OS Jenkins CI. MbedRTT does not use any

Jenkins CI specific features. The independency of MbedRTT from Jenkins CI allows for the possible effective future use of MbedRTT outside of Mbed OS Jenkins CI.

JobRTS depends on Python and the Mbed OS GitHub repository directory structure. GreenteaRTS depends on deterministic Greentea test suite binaries and a database the Greentea test results can be stored in and retrieved from. Deterministic Greentea test suite binaries depend on Mbed OS tools, toolchains, and some environment variables. For the maximum determinism of Greentea test suite binaries, it is recommended to build Mbed OS Greentea test suites in a docker container or in another easy to control environment. TargetRTS depends on having access to the set of Mbed OS 5 supported target/toolchain combinations and Mbed OS resource scanning tools, both of which are publicly available in Mbed OS GitHub repository.

Mbed OS and Mbed OS Jenkins CI requirement for compute depends somewhat on the enabled MbedRTS techniques. The following timings presented are the empirical observed maximum values. JobRTS takes less than a second to execute on most computers. GreenteaRTS takes less than 10 seconds to execute on most computers. GreenteaRTS database query for results takes the longest. TargetRTS takes less than 3 minutes to execute on *m5.4xlarge* Amazon EC2 Instance. TargetRTS takes less than 25 minutes to execute on *Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz.*

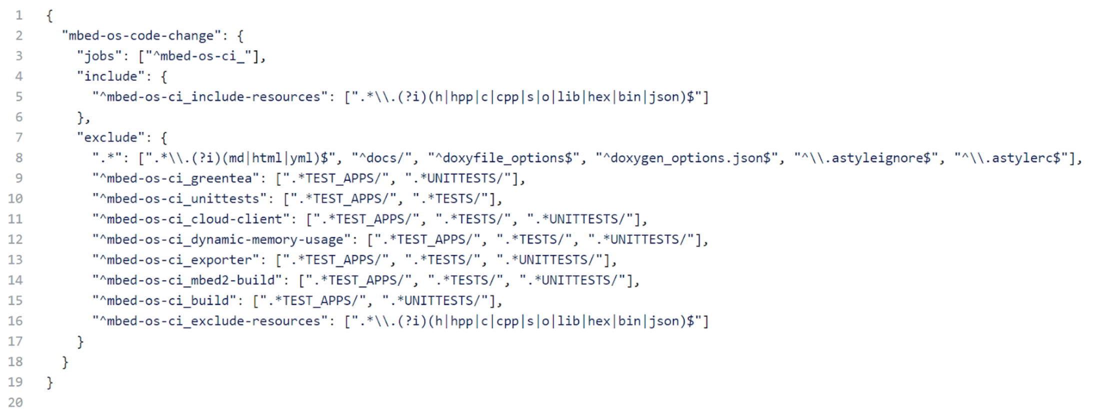
2.2. Safe MbedRTT techniques

JobRTS is a static RTS technique and uses a JSON configuration file in conjugation with *git diff*. JobRTS selects required jobs. The JobRTS config file can be either manually or automatically generated. TargetRTS is a static RTS technique and uses static build resource dependency scanning in conjugation with *git diff*. TargetRTS selects required target/toolchain combinations. GreenteaRTS is a dynamic RTS technique and uses inherit dependency information of binary files in conjugation with a database of previous results. GreenteaRTS selects required Greentea test suites.

2.2.1. JobRTS

A fairly significant percentage of Mbed OS PR’s modify only the files of no more than one of the following sets of files: tools, documentation, resources, tests. Each Mbed OS Jenkins CI job depends only on a subset of the Mbed OS repository files. If a PR does not modify any of the files a job depends on, the job can be skipped. JobRTS is a very simple and crude, yet somewhat effective RTS technique.

JobRTS uses a JSON configuration file (Fig. 9) and *git diff* to determine which Mbed OS Jenkins CI jobs can be skipped. JobRTS config file lists include and exclude file patterns for each Mbed OS Jenkins CI job. If any modified file matches any include pattern and does not match any exclude pattern for a particular Mbed OS Jenkins CI job, the job is selected, otherwise the job is skipped.



```
{
  "mbed-os-code-change": {
    "jobs": ["^mbed-os-ci_"],
    "include": {
      "^mbed-os-ci_include-resources": [".*\\.(?i)(h|hpp|c|cpp|s|o|lib|hex|bin|json)$"]
    },
    "exclude": {
      ".*": [".*\\.(?i)(md|html|yml)$", "^docs/", "^doxyfile_options$", "^doxygen_options.json$", "^\\.astyleignore$", "^\\.astylerc$"],
      "^mbed-os-ci_greentea": [".*TEST_APPS/", ".*UNITTESTS/"],
      "^mbed-os-ci_unittests": [".*TEST_APPS/", ".*TESTS/"],
      "^mbed-os-ci_cloud-client": [".*TEST_APPS/", ".*TESTS/", ".*UNITTESTS/"],
      "^mbed-os-ci_dynamic-memory-usage": [".*TEST_APPS/", ".*TESTS/", ".*UNITTESTS/"],
      "^mbed-os-ci_exporter": [".*TEST_APPS/", ".*TESTS/", ".*UNITTESTS/"],
      "^mbed-os-ci_mbed2-build": [".*TEST_APPS/", ".*TESTS/", ".*UNITTESTS/"],
      "^mbed-os-ci_build": [".*TEST_APPS/", ".*UNITTESTS/"],
      "^mbed-os-ci_exclude-resources": [".*\\.(?i)(h|hpp|c|cpp|s|o|lib|hex|bin|json)$"]
    }
  }
}

```

Figure 9: job-selector.json: Job Selector JSON configuration file

The current revision of Mbed OS Jenkins CI uses a manually generated and maintained JobRTS config file. Comprehensive research on automated JobRTS config file generation has not been performed. Automated JobRTS config file generation depends on figuring Mbed OS Jenkins CI job dependencies automatically. One potentially possible approach to figuring Mbed OS Jenkins CI job dependencies automatically is to monitor what files the job accesses. A Mbed OS Jenkins CI job does not depend on the files the job never accesses.

2.2.2. GreenteaRTS

Ekstazi [16] stores test class dependency information in dependency file. Mbed OS build tools store Greentea test application dependency information in the binary. Unlike *Ekstazi,* which requires dependency file hashes to be calculated in order to determine whether a test class needs to be run, GreenteaRTS only requires calculating the hashes of the test application binaries in order to determine whether a test application needs to be run. A test suite need not be run if a suite with the same binary hash has already passed. GreenteaRTS naturally handles newly added suites/targets/toolchains: if there is no dependency information for some suite/target/toolchain combination, it is selected. (Fig. 10)

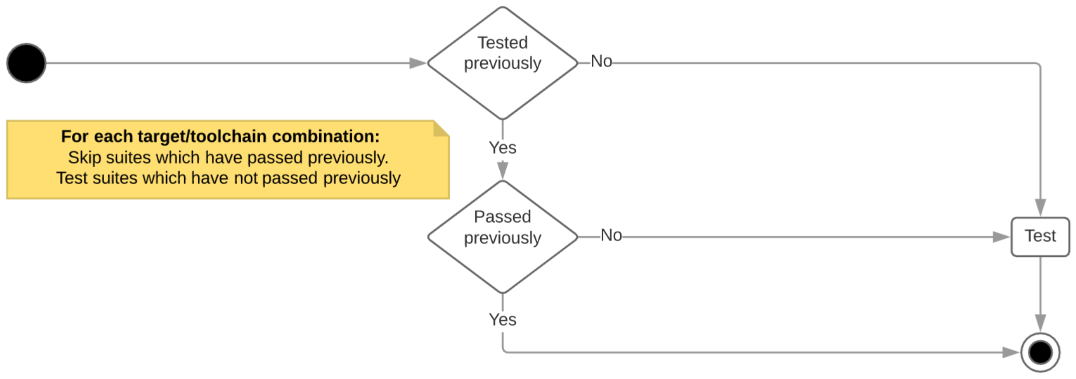
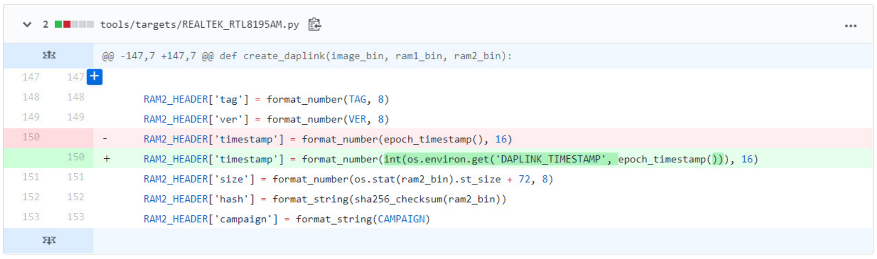

Tested previously
No
Yes
For each target/toolchain combination:
Skip suites which have passed previously.
Test suites which have not passed previously
Passed previously
No
Test
Yes

Figure 10: Greentea Selector selects test suites for each target/toolchain pair based on execution history.

Currently, GreenteaRTS uses Greentea results stored in either MongoDB or InfluxDB. GreenteaRTS is highly compatible with most databases and can be switched to use another type of database. GreenteaRTS takes a *build_data* file as a parameter. The *build_data* file is generated when building Greentea tests. The *build_data* file includes the paths to the generated binaries. GreenteaRTS calculates SHA-1 hash for each binary. GreenteaRTS selects results from the database where the result binary SHA-1 matches any calculated *build_data* binary SHA-1 and the result is pass. GreenteaRTS can be configured to ignore results with too old timestamps. Currently, in Mbed OS Jenkins CI, GreenteaRTS is configured to ignore results with a timestamp over four days old.

For maximum performance, GreenteaRTS requires deterministic binary hashes. If, for example, a binary includes a non-constant timestamp, the binary hash is non-deterministic, as it is different every time. GreenteaRTS selects a suite practically always if the binary hash of the suite is non-deterministic. GreenteaRTS does not select a suite with non-deterministic binary hash only when executing GreenteaRTS with the exact same binaries as previously (when rebuilding Mbed OS Jenkins CI Greentea test job). A number of Mbed OS tools modifications were required and implemented in order to reduce Greentea test binary hash non-determinism. (Fig. 11)



```
2 tools/targets/REALTEK_RTL8195AM.py
@@ -147,7 +147,7 @@ def create_daplink(image_bin, ram1_bin, ram2_bin):

        RAM2_HEADER['tag'] = format_number(TAG, 8)
        RAM2_HEADER['ver'] = format_number(VER, 8)
-       RAM2_HEADER['timestamp'] = format_number(epoch_timestamp(), 16)
+       RAM2_HEADER['timestamp'] = format_number(int(os.environ.get('DAPLINK_TIMESTAMP', epoch_timestamp())), 16)
        RAM2_HEADER['size'] = format_number(os.stat(ram2_bin).st_size + 72, 8)
        RAM2_HEADER['hash'] = format_string(sha256_checksum(ram2_bin))
        RAM2_HEADER['campaign'] = format_string(CAMPAIGN)
```

Figure 11: Required REALTEK_RTL8195AM tools modifications in order to get deterministic binary hashes

The current revision of GreenteaRTS considers a test result to depend only on the selected binary/target combination. Greentea test result does not depend only on the binary/target combination, but also on the test tools, some of which reside in the Mbed OS GitHub repository, some of which are specific to a subset of all Greentea test suites. Currently, it is extremely rare for a PR to modify the Greentea specific tools in the Mbed OS GitHub repository after their addition, and potentially affect Greentea test suite results. Nevertheless, depending on future Mbed OS modifications, it might be beneficial to research, analyse, and evaluate Greentea test suite execution dependencies in more detail.

A subset of all target/toolchain/Greentea test suite combinations still exhibit non-deterministic behavior. The reasons for the non-determinism present are currently mostly due to the timestamps and usage of absolute file paths.

2.2.3. TargetRTS

Over 50% of PR's modify only resources (Mbed OS resource files) and documentation. Over 25% of PR's modify only target/toolchain specific resources (resources which are not common to all Mbed OS 5 supported target/toolchain combinations) and documentation. If a PR modifies only resources and documentation, only the target/toolchain combinations with intersecting resources with the modified resources require testing. The current revision of TargetRTS classifies only a subset of potential resource files (by file extension) as resources (Fig. 12).

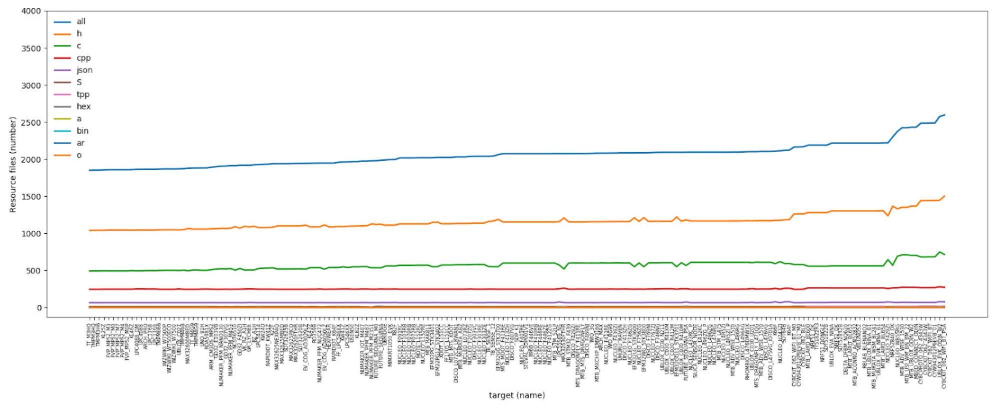


Figure 12: Mbed OS 5 supported targets resources.

Any Mbed OS 5 supported target/toolchain combination depends on less than 25% of all resources. Almost 4000 out of almost 7500 Mbed OS PR's modified only resources and documentation. Any target/toolchain combination requires testing in less than 25% (*1000/4000*) of Mbed OS PR's which modify only resources and documentation. Any target/toolchain combination requires testing in less than 60% (*3500/7500* + *1000/7500*) of all Mbed OS PR's. (Fig. 13)

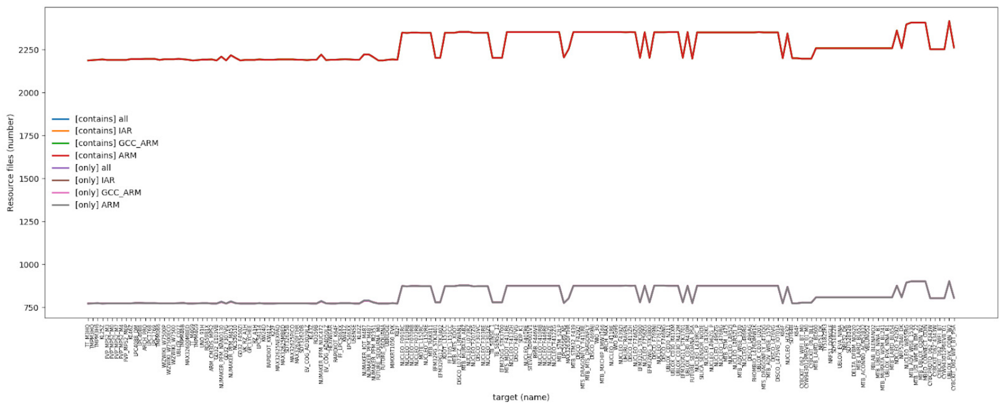


Figure 13: Mbed OS PR's with resource modifications count by target. The figure includes both all PR's with resource modifications (*[contains]*) and PR's with only resource modifications (*[only]*)

Each target/toolchain combination supports a set of features. Each target/toolchain combination depends on a set of resources, including, but not limited to common resources and feature specific resources. Some resources are common to all target/toolchain combinations, some resources are shared between targets within a target family, and some resources are unique to specific targets or target/toolchain combinations. (Fig. 14)

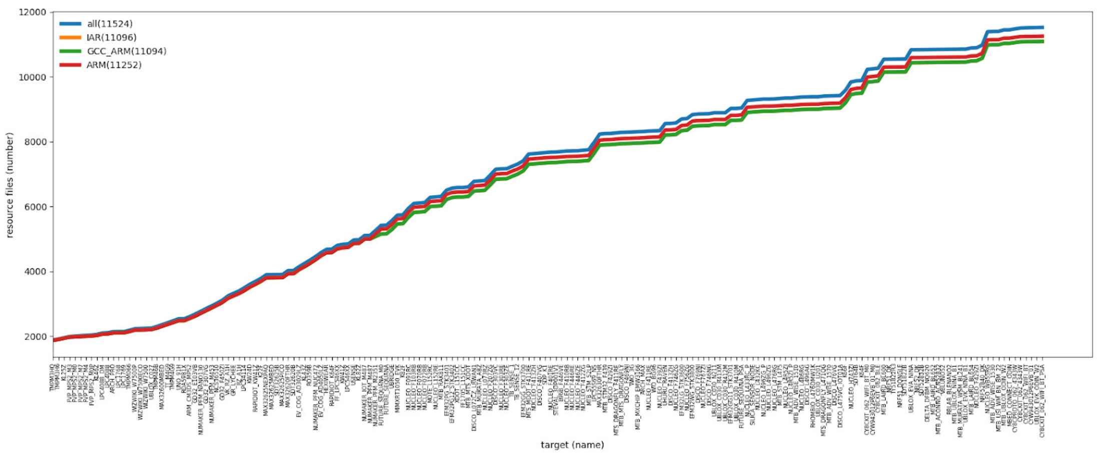
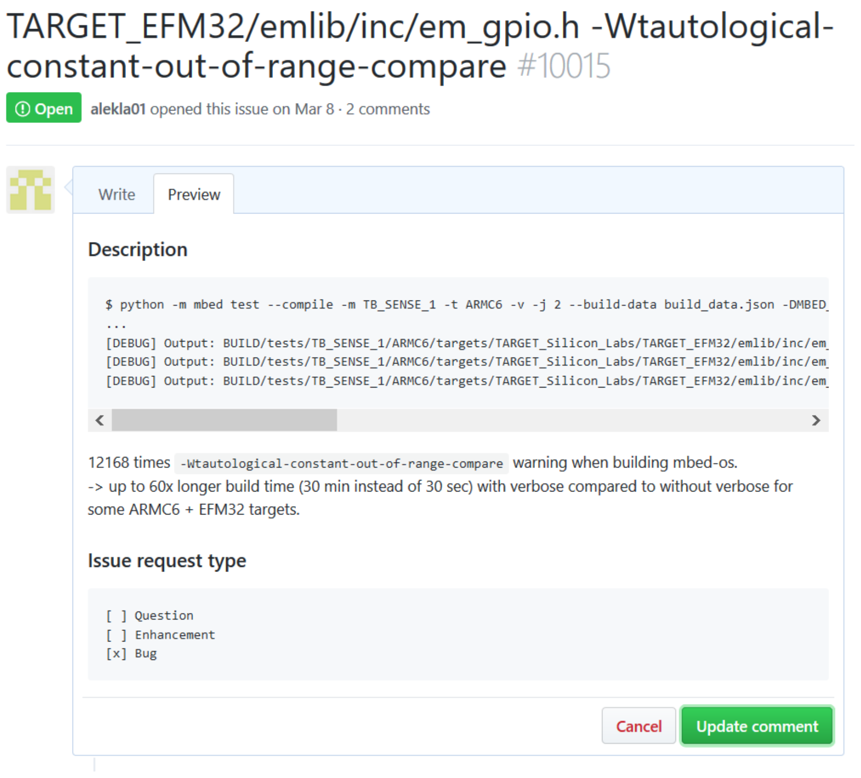


Figure 14: Integral of unique resources over Mbed OS 5 targets.

Over 70% of PR's modify only resources, *targets.json*, tests, and documentation. Over 80% of the test modifications are Greentea test modifications. Each Greentea test (suite) supports a subset of target/toolchain combinations. Each *targets.json* modification affects a subset of all supported target/toolchain combinations. The current revision of TargetRTS always selects all target/toolchain combinations if a PR modifies Greentea tests or *targets.json*. Only the union of target/toolchain combinations which support the modified tests, are affected by the *targets.json* modifications or have intersecting resource files with the modified resource files are required. If TargetRTS were to be modified to also consider *targets.json* and Greentea test modifications, the percentage of PR's TargetRTS affects would increase significantly.

2.2.4. Disable verbose build traces

Disabling verbose build traces when building Mbed OS applications can reduce build time from over an hour to a few minutes for some target/toolchain combinations. The drastic improvement in build time is mostly due to the removal of all warning traces. Some target/toolchain combinations may produce the same warning trace over 12K times when building Mbed OS once (Fig. 15). Currently, Mbed OS Jenkins CI builds normally with verbose build traces disabled. Mbed OS Jenkins CI builds with verbose build traces enabled only when rebuilding a failed build. Rebuilding a failed build with verbose traces enabled is an automated process in Mbed OS Jenkins CI.



TARGET_EFM32/emlib/inc/em_gpio.h -Wtautological-constant-out-of-range-compare #10015

Open alekla01 opened this issue on Mar 8 · 2 comments

Description

```
$ python -m mbed test --compile -m TB_SENSE_1 -t ARMC6 -v -j 2 --build-data build_data.json -DMBED_
...
[DEBUG] Output: BUILD/tests/TB_SENSE_1/ARMC6/targets/TARGET_Silicon_Labs/TARGET_EFM32/emlib/inc/em_
[DEBUG] Output: BUILD/tests/TB_SENSE_1/ARMC6/targets/TARGET_Silicon_Labs/TARGET_EFM32/emlib/inc/em_
[DEBUG] Output: BUILD/tests/TB_SENSE_1/ARMC6/targets/TARGET_Silicon_Labs/TARGET_EFM32/emlib/inc/em_
```

12168 times `-Wtautological-constant-out-of-range-compare` warning when building mbed-os.
-> up to 60x longer build time (30 min instead of 30 sec) with verbose compared to without verbose for some ARMC6 + EFM32 targets.

Issue request type

```
[ ] Question
[ ] Enhancement
[x] Bug
```

Figure 15: Mbed OS GitHub repository issue #10015 description.

2.3. Researched, but not implemented MbedRTT techniques and improvements

A multitude of potential Mbed OS RTS and RTO techniques were researched but never implemented. Some of the implemented MbedRTT techniques could be further improved and combined for even more effective Mbed OS RT.

2.3.1. Use the build folders of previous builds to optimize build time

Mbed OS build tools and toolchains can optimize Mbed OS building if a build directory exists. A source file needs updating / compiling only when either the file itself or any of its dependencies have been modified. An *elf* file needs to be relinked and a binary file needs to be recreated only when any of the dependencies of the program have been modified.

Using the build folders of previous builds to optimize build time requires the build folders be available. Currently, Mbed OS Jenkins CI stores the build folders of Mbed OS 5 test and example builds in AWS S3. Currently, Mbed OS Jenkins CI does not store the build folders of exporter tests in AWS S3. Mbed OS Jenkins CI exporter tests export and build Mbed OS examples using many different ides.

The potential time savings of using the build folders of previous builds are noticeable. Reduction of up to 80% in examples build time for some target/toolchain and target/ide combinations were observed in initial testing. Related to using the build folders of previous builds to optimize build time, Mbed OS Jenkins CI makes use of cached build folders by first building with verbose disabled and after failing with verbose enabled. This thesis neither researches the usage of the build folders of previous builds in more detail nor present any conclusive results on the subject.

2.3.2. MbedRTS based on file access monitoring

Each Mbed OS test accesses (depends) only a subset of all Mbed OS GitHub repository files. For example, building all Mbed OS Greentea test suites for K66F/GCC_ARM target/toolchain combination depends only on a relatively small (deterministic) subset of all Mbed OS files (Fig. 16). Mbed OS build tools do not include Greentea test suite specific tools. If a pull request only modifies tool files and source files, and the modified tool files are not accessed when building the first Greentea test suite for a selected target/toolchain combination, building all but the suites with intersecting source files with the modified source files can be skipped. Building all Mbed OS Greentea test suites for K66F/GCC_ARM target/toolchain combination depends on less than 40% (63 / 163) of all Python files in the tools directory in the Mbed OS GitHub repository. Files accessed by a program can be listed. The files a test, such as building Mbed OS Greentea test suites, depends on, can be listed. A Mbed OS test outcome is independent of modifications to files or the removal of files the test does not depend on. If, for example, a test passes at a first git commit, and the second git commit only modifies or removes files executing the test at the first git commit did not access, executing the test at the second git commit is not required. Besides the previously mentioned example use cases, there are many other potential use cases for the information acquired by monitoring file accesses. File access monitoring could potentially be used to further improve JobRTS, TargetRTS and other Smart Tester related techniques. Implementing Mbed OS regression test selection based on file access monitoring requires further research, may not be beneficial in Mbed OS testing at this point in time, and is out of the scope of this thesis.

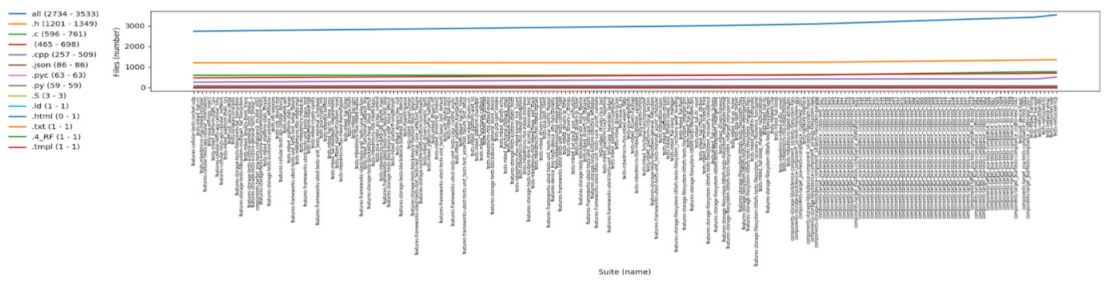


Figure 16: Integral over unique files accessed when building Mbed OS Greentea test suites for K66F/GCC_ARM target/toolchain combination ordered by the number of accessed files. The build folder and other caches were cleared between building subsequent suites.

2.3.3. Static Greentea RTS techniques

If a PR modifies only tests, only the modified tests require testing. Over 5% of PR's modify only tests. Though easily implementable, test suites, test applications, to build selection for PR's with only test modifications is out of the scope of the thesis.

Each supported target/toolchain combination supports over 150 Greentea test suites (Fig. 17). Each Greentea test suite depends on a set of libraries, including, but not limited to Mbed. Each library supports a set of target/toolchain combinations. Each library contains a set of resources, the subsets of which belong to the subsets of all supported target/toolchain combinations. Greentea test suites to build can be selected based on test suite dependencies, either similarly to target/toolchain combination selection or perhaps more effectively somewhat similarly to *Ekstazi* [16]; by monitoring the intersection of accessed files and modified files, a safe reduction of dependent modified files can be performed on a target/toolchain combination basis. Greentea test suites to build selection is out of the scope of the thesis.

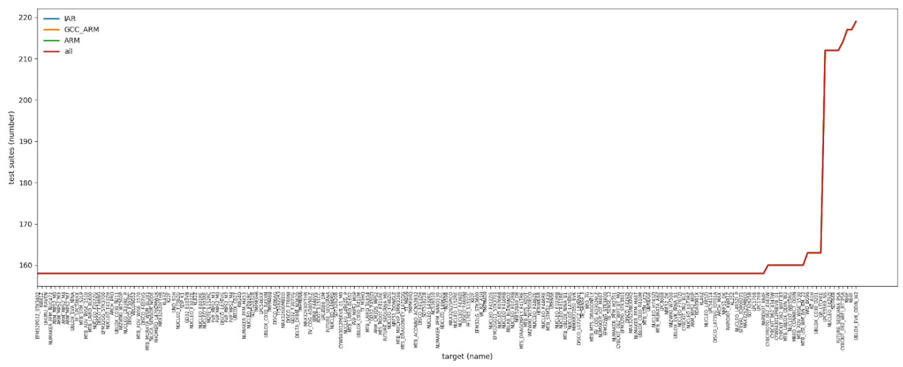


Figure 17: Greentea test suites

2.3.4. Combine JobRTS and TargetRTS

JobRTS config file defines the dependencies of each Mbed OS Jenkins CI job. TargetRTS defines the dependencies of each target/toolchain combination if only resources are modified. This intersecting dependency information can be used in Mbed OS Jenkins CI to reduce unnecessary testing.

For example, a PR modifies both tools and resources. The PR is tested in Mbed OS Jenkins CI and fails to build all but a subset of target/toolchain combinations. The failure is fixed by modifying only a subset of resource files. Now, it is known that only the union of the failed target/toolchain combinations and the target/toolchain combinations with intersecting resource files with the modified resource files require building as all other target/toolchain combination are known to pass. This particular form of combining JobRTS and TargetRTS is out of the scope of this thesis. The current revision of MbedRTT loosely combines JobRTS and

TargetRTS by using JobRTS in TargetRTS to filter out documentation files before checking if all modified files are resource files.

2.3.5. Optimize exporter test

The current revision of Mbed OS Jenkins CI exporter test builds examples with verbose build traces enabled. Currently, disabling verbose build traces for Mbed OS Jenkins CI exporter test would save very little time. Nevertheless, depending on future Mbed OS Jenkins CI exporter test target/IDE combinations, it may become beneficial for Mbed OS Jenkins CI exporter test to build with verbose traces disabled.

Currently, Mbed OS Jenkins CI exporter test builds 12 examples with K64F/µVision target/IDE combination. The current Mbed OS Jenkins CI exporter test revision launches µVision 12 times for K64F/µVision target/IDE combination. Launching an IDE can take over 3 minutes. Launching µVision 12 times can take over 35 minutes (over 90% of the overall exporter test time). Potential Mbed OS Jenkins CI exporter test optimization techniques include, but are not limited to: optimize IDE launch time, minimize IDE relaunch count.

3. Case Studies: Safe MbedRTT techniques

This thesis includes case studies of all MbedRTT techniques currently enabled in Mbed OS Jenkins CI. The effectiveness of MbedRTT is measured empirically from available Mbed OS Jenkins CI test results. This thesis does not evaluate the different MbedRTT techniques against each other as the techniques target different areas of Mbed OS testing.

The probabilities of JobRTS and TargetRTS affecting PR testing can be approximated using the available data of the history of Mbed OS PR modifications (Fig. 5, Fig. 6). The effectiveness of GreenteaRTS can be approximated by comparing the number of selected Greentea test cases with the number of all selectable Greentea test cases. The empirically measured execution times of Mbed OS Jenkins CI testing for different PR's can be used to approximate the reduction in execution time of Mbed OS Jenkins CI testing with MbedRTT enabled compared to with MbedRTT disabled.

3.1. Case Study: JobRTS

This case study uses the current revision of Mbed OS Jenkins CI JobRTS config file (Fig. 9). If a PR modifies only documentation (any file that requires no Mbed OS Jenkins CI testing can effectively be considered documentation by JobRTS), Mbed OS Jenkins CI skips all testing. For example, modifying only *.travis.yml* file requires no Mbed OS Jenkins CI testing (Fig. 18). Mbed OS Jenkins CI checkouts the PR and executes the JobRTS. Mbed OS Jenkins CI execution time is no longer than a few minutes. The probability of a PR only modifying documentation files is (or at least has traditionally been) less than 5% (*220 / 7434*).

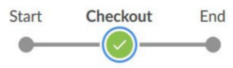


Figure 18: Mbed OS Jenkins CI testing for a PR that modifies only *.travis.yml*.

If a PR modifies only tests, Mbed OS Jenkins CI can skip all but building and executing the modified tests. If a PR modifies only unit tests, Mbed OS Jenkins CI executes only unit tests. Mbed OS Jenkins CI test execution takes no longer than a few minutes. If a PR modifies only Greentea tests, Mbed OS Jenkins CI builds and executes only Greentea tests (Fig. 19). Mbed OS Jenkins CI test execution takes approximately an hour depending on Mbed OS Jenkins CI stability, queue length and whether GreenteaRTS is enabled. The probability of a PR only modifying tests is less than 10% (*479 / 7434*).

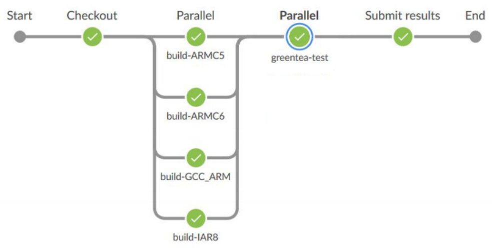


Figure 19: Mbed OS Jenkins CI testing for a PR that modifies only Greentea tests.

If a PR does not modify only tests and documentation (PR modifies also other files), Mbed OS Jenkins CI does not skip any tests (Fig. 20). Mbed OS Jenkins CI test execution takes usually less than two hours depending on Mbed OS Jenkins CI stability, queue length, and whether GreenteaRTS is enabled. The probability of a PR not modifying only tests and documentation is more than 90% (*6955 / 7434*).

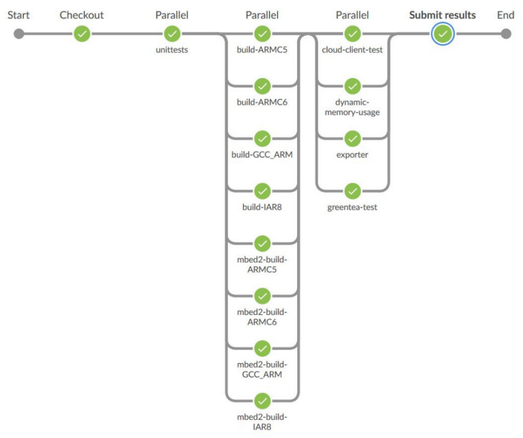


Figure 20: Mbed OS Jenkins CI testing for a pull request that does not modify only documentation and tests.

Although, currently, JobRTS reduces overall Mbed OS Jenkins CI testing by less than 10%, it is worth it, as the implementation is simple and there are very few downsides to using it. The only “real” downside to using JobRTS is the currently manually generated and maintained JobRTS config file. Currently, JobRTS config file requires no maintenance, as the only excluded files are files which are very unlikely to affect Mbed OS Jenkins CI test verdict. With automated JobRTS config file generation, the effectiveness of JobRTS would only increase.

3.2. Case Study: GreenteaRTS

If a Greentea test suite binary has already passed it does not require testing. Most PR’s affect the hash of very few Greentea test suite binaries. GreenteaRTS reduces PR Mbed OS Jenkins CI Greentea testing by over 90% on average (Fig. 21).

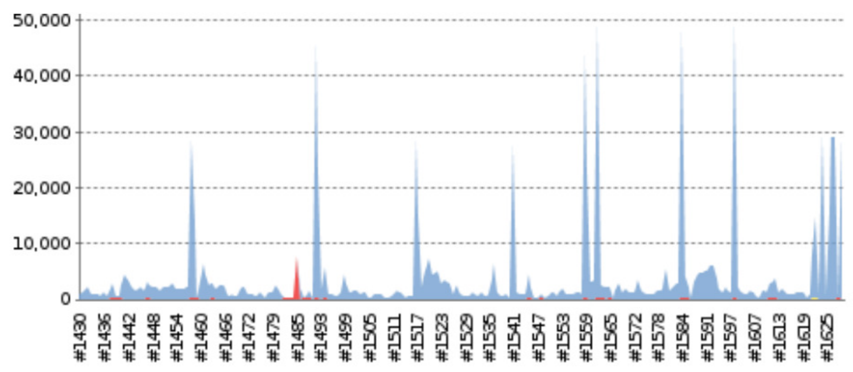
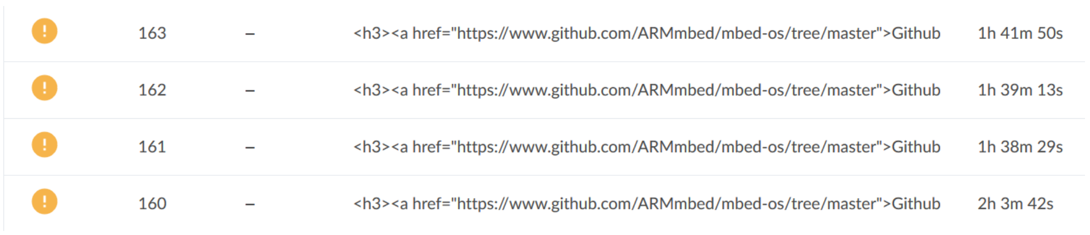


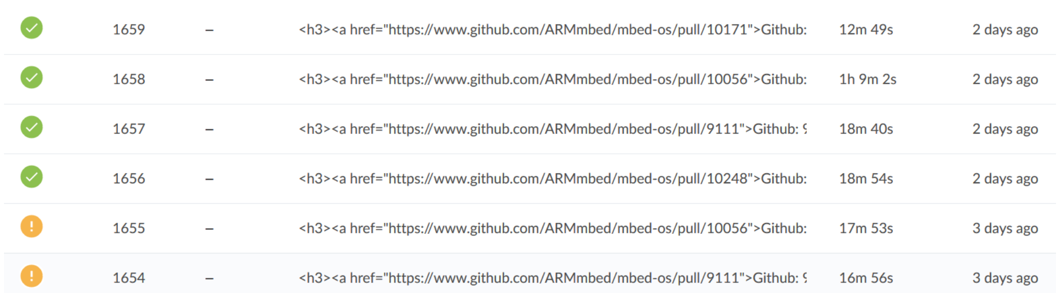
Figure 21: Greentea test case count for PR’s with Greentea Selector enabled.

Mbed OS Jenkins CI executes Greentea tests for a PR in just under two hours on average when GreenteaRTS is not enabled (Fig. 22). Mbed OS Jenkins CI executes Greentea tests for a PR in less than 30 minutes on average when GreenteaRTS is enabled (Fig. 23).



!	163	–	<h3><a href="https://www.github.com/ARMmbed/mbed-os/tree/master">Github	1h 41m 50s
!	162	–	<h3><a href="https://www.github.com/ARMmbed/mbed-os/tree/master">Github	1h 39m 13s
!	161	–	<h3><a href="https://www.github.com/ARMmbed/mbed-os/tree/master">Github	1h 38m 29s
!	160	–	<h3><a href="https://www.github.com/ARMmbed/mbed-os/tree/master">Github	2h 3m 42s

Figure 22: Greentea execution times when Greentea Selector is disabled.



✓	1659	–	<h3><a href="https://www.github.com/ARMmbed/mbed-os/pull/10171">Github:	12m 49s	2 days ago
✓	1658	–	<h3><a href="https://www.github.com/ARMmbed/mbed-os/pull/10056">Github:	1h 9m 2s	2 days ago
✓	1657	–	<h3><a href="https://www.github.com/ARMmbed/mbed-os/pull/9111">Github: !	18m 40s	2 days ago
✓	1656	–	<h3><a href="https://www.github.com/ARMmbed/mbed-os/pull/10248">Github:	18m 54s	2 days ago
!	1655	–	<h3><a href="https://www.github.com/ARMmbed/mbed-os/pull/10056">Github:	17m 53s	3 days ago
!	1654	–	<h3><a href="https://www.github.com/ARMmbed/mbed-os/pull/9111">Github: !	16m 56s	3 days ago

Figure 23: Greentea execution times when Greentea Selector is enabled.

There is only a limited number of physical targets available for Greentea testing in Mbed OS Jenkins CI. Physical targets wear down relative to the number of times they are flashed. Eventually physical targets stop working and need to be replaced. GreenteaRTS reduces target flashing on average and hence reduces the frequency at which targets need to be replaced. The reduction in Mbed OS Jenkins CI Greentea test execution time also allows more Greentea tests to be executed in parallel and sequentially, if required.

Mbed OS Jenkins CI requires GreenteaRTS to operate efficiently. When GreenteaRTS does not work, Mbed OS Jenkins CI Greentea testing becomes somewhat unstable (Fig. 24). When Mbed OS Jenkins CI Greentea testing is unstable, Mbed OS Jenkins CI Greentea test needs to be executed more than once on average for each PR. Executing Mbed OS Jenkins CI with GreenteaRTS either disabled or not working wastes time and resources.

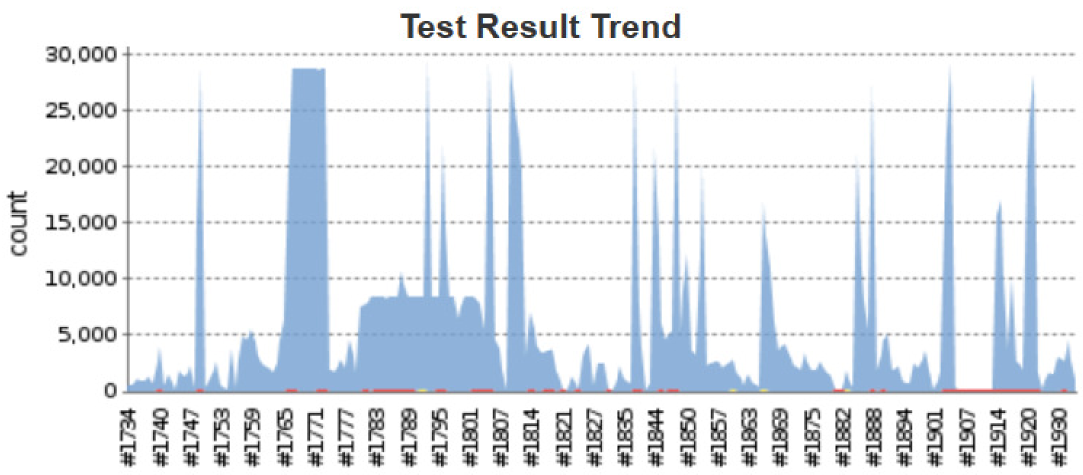


Figure 24: Unique Greentea test cases executed count by job number. From approximately build number #1765 to #1771 GreenteaRTS could not retrieve results from the database. From approximately build number #1765 to #1807 GreenteaRTS could not send results to the database.

Currently, Mbed OS Jenkins CI nightly Greentea test executes over 40K test cases (once every night). The current revision of Mbed OS Jenkins CI nightly Greentea test has GreenteaRTS disabled. Mbed OS Jenkins CI nightly Greentea test case count increases over time. Mbed OS Jenkins CI Greentea Test executes at most around 30K test cases for a PR. Mbed OS Jenkins CI Greentea test is executed for from 0 to easily over 10 PR’s in a day. The number of Greentea test cases executed for PR’s in a day is on average lower than the number of Greentea test cases executed in Mbed OS Jenkins CI nightly Greentea test. (Fig. 25)

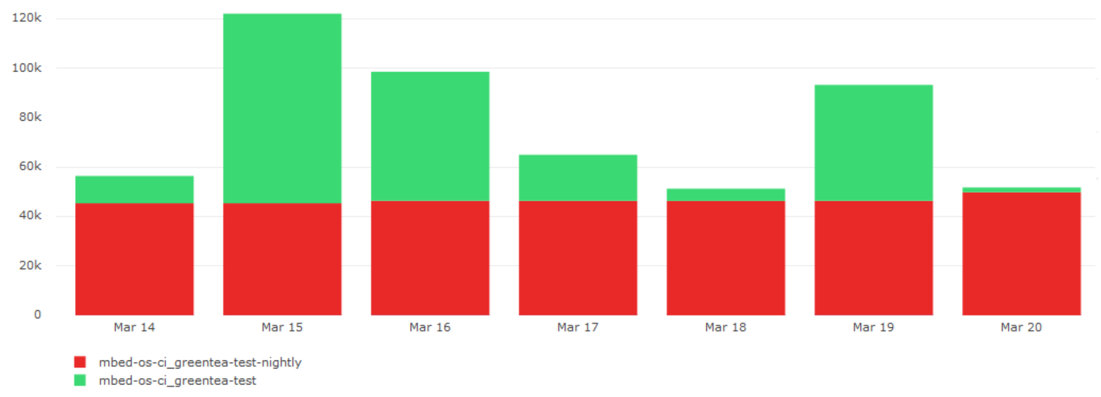


Figure 25: Greentea test case count for pull requests versus nightly.

3.3. Case Study: TargetRTS

If a PR modifies only resource files, only building, exporting, and testing the affected target/toolchain combinations is required (Fig. 26). If a PR modifies other files but resource files, any target/toolchain combination may be required. The current revision of TargetRTS is activated only if a PR modifies only resources. Over 50% (*3902 / 7434*) of PR's modify only resources. Around or slightly over 25% (*1892 / 7434*) of PR's modify only target/toolchain specific resources.

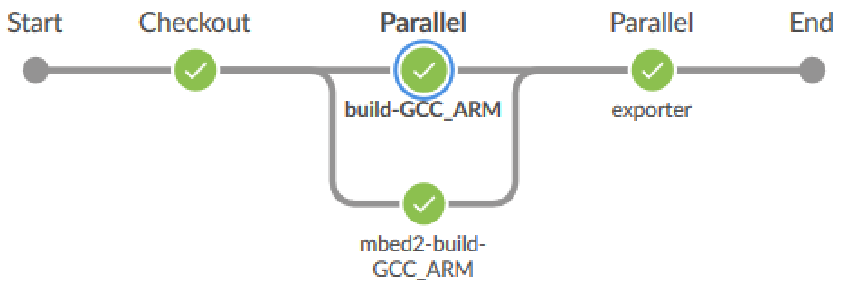
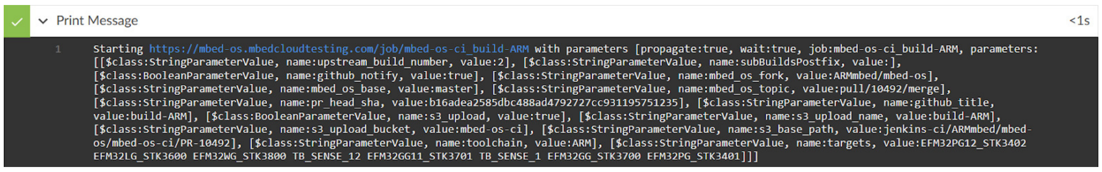


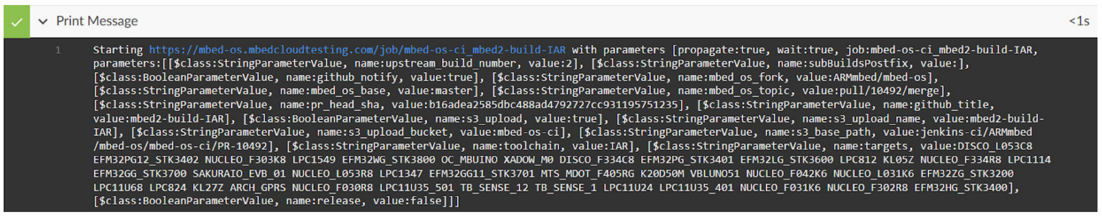
Figure 26: Build and exporting for one target/toolchain combination with all but the required resources removed.

For example, if a PR modifies only EFM32 specific resources, TargetRTS selects only EFM32 targets, and Mbed OS Jenkins CI builds, exports, and tests only EFM32 targets (Fig. 27). Currently, TargetRTS does not implement resource scanning for Mbed OS 2 specific targets. Currently, Mbed OS 2 build target/toolchain combinations includes the selected Mbed OS 5 target/toolchain combinations and Mbed OS 2 specific target/toolchain combinations (Fig. 28).



```
Print Message                                                                                   <1s
Starting https://mbed-os.mbedcloudtesting.com/job/mbed-os-ci_build-ARM with parameters [propagate:true, wait:true, job:mbed-os-ci_build-ARM, parameters:
[[$class:StringParameterValue, name:upstream_build_number, value:2], [$class:StringParameterValue, name:subBuildsPostfix, value:],
[$class:BooleanParameterValue, name:github_notify, value:true], [$class:StringParameterValue, name:mbed_os_fork, value:ARMmbed/mbed-os],
[$class:StringParameterValue, name:mbed_os_base, value:master], [$class:StringParameterValue, name:mbed_os_topic, value:pull/10492/merge],
[$class:StringParameterValue, name:pr_head_sha, value:b16adea2585dbc488ad4792727cc931195751235], [$class:StringParameterValue, name:github_title,
value:build-ARM], [$class:BooleanParameterValue, name:s3_upload, value:true], [$class:StringParameterValue, name:s3_upload_name, value:build-ARM],
[$class:StringParameterValue, name:s3_upload_bucket, value:mbed-os-ci], [$class:StringParameterValue, name:s3_base_path, value:jenkins-ci/ARMmbed/mbed-
os/mbed-os-ci/PR-10492], [$class:StringParameterValue, name:toolchain, value:ARM], [$class:StringParameterValue, name:targets, value:EFM32PG12_STK3402
EFM32LG_STK3600 EFM32WG_STK3800 TB_SENSE_12 EFM32GG11_STK3701 TB_SENSE_1 EFM32GG_STK3700 EFM32PG_STK3401]]]
```

Figure 27: Mbed OS 5 target/toolchain combinations for a pull request that modifies only EFM32 target specific resource files.



```
Print Message                                                                                   <1s
Starting https://mbed-os.mbedcloudtesting.com/job/mbed-os-ci_mbed2-build-IAR with parameters [propagate:true, wait:true, job:mbed-os-ci_mbed2-build-IAR,
parameters:[[$class:StringParameterValue, name:upstream_build_number, value:2], [$class:StringParameterValue, name:subBuildsPostfix, value:],
[$class:BooleanParameterValue, name:github_notify, value:true], [$class:StringParameterValue, name:mbed_os_fork, value:ARMmbed/mbed-os],
[$class:StringParameterValue, name:mbed_os_base, value:master], [$class:StringParameterValue, name:mbed_os_topic, value:pull/10492/merge],
[$class:StringParameterValue, name:pr_head_sha, value:b16adea2585dbc488ad4792727cc931195751235], [$class:StringParameterValue, name:github_title,
value:mbed2-build-IAR], [$class:BooleanParameterValue, name:s3_upload, value:true], [$class:StringParameterValue, name:s3_upload_name, value:mbed2-build-
IAR], [$class:StringParameterValue, name:s3_upload_bucket, value:mbed-os-ci], [$class:StringParameterValue, name:s3_base_path, value:jenkins-ci/ARMmbed
/mbed-os/mbed-os-ci/PR-10492], [$class:StringParameterValue, name:toolchain, value:IAR], [$class:StringParameterValue, name:targets, value:DISCO_L053C8
EFM32PG12_STK3402 NUCLEO_F303K8 LPC1549 EFM32WG_STK3800 OC_MBUINO XADOW_M0 DISCO_F334C8 EFM32PG_STK3401 EFM32LG_STK3600 LPC812 KL05Z NUCLEO_F334R8 LPC1114
EFM32GG_STK3700 SAKURAIO_EVB_01 NUCLEO_L053R8 LPC1347 EFM32GG11_STK3701 MTS_MDOT_F405RG K20D50M VBLUNO51 NUCLEO_F042K6 NUCLEO_L031K6 EFM32ZG_STK3200
LPC11U68 LPC824 KL27Z ARCH_GPRS NUCLEO_F030R8 LPC11U35_501 TB_SENSE_12 TB_SENSE_1 LPC11U24 LPC11U35_401 NUCLEO_F031K6 NUCLEO_F302R8 EFM32HG_STK3400],
[$class:BooleanParameterValue, name:release, value:false]]]
```

Figure 28: Mbed OS 2 target/toolchain combinations for a pull request that modifies only EFM32 target specific resource files.

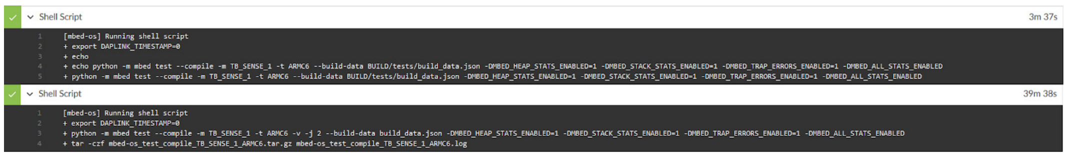
If a PR modifies only resources, TargetRTS takes less than 3 minutes to execute on *m5.4xlarge* Amazon EC2. If a PR does not modify only resources, TargetRTS takes less than 1 second to execute on *m5.4xlarge* Amazon EC2 Instance. (Fig. 29)

Figure 29: Resource scanning time on a m5.4xlarge Amazon EC2 Instance for all Mbed OS 5 supported target/toolchain combinations.

Mbed OS Jenkins CI testing is parallelized only to a degree. If Mbed OS Jenkins CI maximum node instance cap is exceeded, testing is queued. Mbed OS Jenkins CI maximum node instance cap is practically always exceeded when building all target/toolchain combinations for one PR. Building and exporting all target/toolchain combinations can take over 1.5h depending on how many PR's are under testing at the same time. TargetRTS can reduce test execution time drastically depending on how large number of target/toolchain combinations gets selected. In the best case scenario (if only a small subset of target/toolchain combinations are selected) TargetRTS can reduce overall Mbed OS Jenkins CI test execution time for a PR down to less than 30 minutes.

3.4. Case Study: Verbose build traces

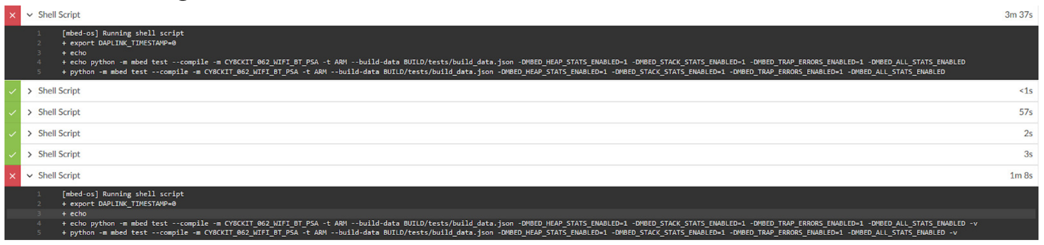
Building Mbed OS with verbose (build traces enabled) can take over 60 times longer than with verbose disabled. Build time with verbose disabled takes on average less than 5 minutes in Mbed OS Jenkins CI. Build time with verbose enabled takes up to 40 minutes for some target/toolchain combinations. Building Mbed OS multiple times subsequently (first with verbose disabled and then with verbose enabled) and not removing build folder before building with verbose enabled is much faster than building Mbed OS multiple times subsequently and removing build folder before building with verbose enabled; For this reason, Mbed OS Jenkins CI builds first with verbose disabled, and then if the building failed, with verbose enabled. (Fig. 30, Fig. 31)



```
[mbed-os] Running shell script
+ export DAPLINK_TIMESTAMP=0
+ echo
+ echo python -m mbed test --compile -m TB_SENSE_1 -t ARMC6 --build-data BUILD/tests/build_data.json -DMBED_HEAP_STATS_ENABLED=1 -DMBED_STACK_STATS_ENABLED=1 -DMBED_TRAP_ERRORS_ENABLED=1 -DMBED_ALL_STATS_ENABLED
+ python -m mbed test --compile -m TB_SENSE_1 -t ARMC6 --build-data BUILD/tests/build_data.json -DMBED_HEAP_STATS_ENABLED=1 -DMBED_STACK_STATS_ENABLED=1 -DMBED_TRAP_ERRORS_ENABLED=1 -DMBED_ALL_STATS_ENABLED
```

```
[mbed-os] Running shell script
+ export DAPLINK_TIMESTAMP=0
+ python -m mbed test --compile -m TB_SENSE_1 -t ARMC6 -v -j 2 --build-data build_data.json -DMBED_HEAP_STATS_ENABLED=1 -DMBED_STACK_STATS_ENABLED=1 -DMBED_TRAP_ERRORS_ENABLED=1 -DMBED_ALL_STATS_ENABLED
+ tar -czf mbed-os_test_compile_TB_SENSE_1_ARMC6.tar.gz mbed-os_test_compile_TB_SENSE_1_ARMC6.log
```

Figure 30: Build times with verbose enabled and disabled.

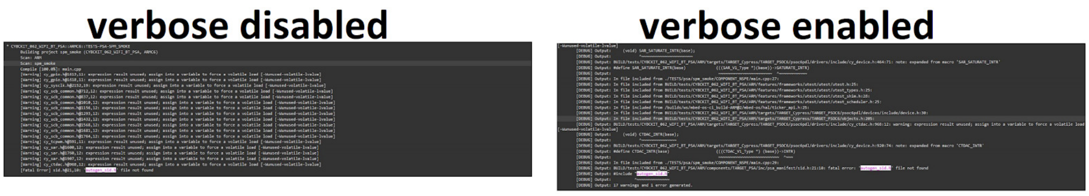


```
[mbed-os] Running shell script
+ export DAPLINK_TIMESTAMP=0
+ echo
+ echo python -m mbed test --compile -m CY8CKIT_062_WIFI_BT_PSA -t ARM --build-data BUILD/tests/build_data.json -DMBED_HEAP_STATS_ENABLED=1 -DMBED_STACK_STATS_ENABLED=1 -DMBED_TRAP_ERRORS_ENABLED=1 -DMBED_ALL_STATS_ENABLED
+ python -m mbed test --compile -m CY8CKIT_062_WIFI_BT_PSA -t ARM --build-data BUILD/tests/build_data.json -DMBED_HEAP_STATS_ENABLED=1 -DMBED_STACK_STATS_ENABLED=1 -DMBED_TRAP_ERRORS_ENABLED=1 -DMBED_ALL_STATS_ENABLED
```

```
[mbed-os] Running shell script
+ export DAPLINK_TIMESTAMP=0
+ echo
+ echo python -m mbed test --compile -m CY8CKIT_062_WIFI_BT_PSA -t ARM --build-data BUILD/tests/build_data.json -DMBED_HEAP_STATS_ENABLED=1 -DMBED_STACK_STATS_ENABLED=1 -DMBED_TRAP_ERRORS_ENABLED=1 -DMBED_ALL_STATS_ENABLED -v
+ python -m mbed test --compile -m CY8CKIT_062_WIFI_BT_PSA -t ARM --build-data BUILD/tests/build_data.json -DMBED_HEAP_STATS_ENABLED=1 -DMBED_STACK_STATS_ENABLED=1 -DMBED_TRAP_ERRORS_ENABLED=1 -DMBED_ALL_STATS_ENABLED -v
```

Figure 31: Build times with verbose first disabled and then enabled after failing.

Mbed OS Jenkins CI rebuilds Mbed OS with verbose enabled after failure because building Mbed OS with verbose disabled produces logs with very little information, and it is not always possible for Mbed OS developers to rebuild the failed builds with verbose enabled by themselves (Fig. 32).



```
* CYBCKIT_062_WIFI_BT_PSA::ARMC6::TESTS-PSA-SPM_SMOKE
    Building project spm_smoke (CYBCKIT_062_WIFI_BT_PSA, ARMC6)
    Scan: ARM
    Scan: spm_smoke
    Compile [100.0%]: main.cpp
    [Warning] cy_gpio.h@1613,11: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_gpio.h@1618,11: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_sysclk.h@2152,19: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_scb_common.h@712,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_scb_common.h@837,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_scb_common.h@1010,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_scb_common.h@1156,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_scb_common.h@1293,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_scb_common.h@1432,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_scb_common.h@1568,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_scb_common.h@1681,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_scb_common.h@1764,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_tcpwm.h@591,11: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_sar.h@1600,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_sar.h@1760,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_sar.h@1907,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Warning] cy_ctdac.h@968,12: expression result unused; assign into a variable to force a volatile load [-Wunused-volatile-lvalue]
    [Fatal Error] sid.h@21,10: 'autogen_sid.h' file not found
```

```
[-Wunused-volatile-lvalue]
      [DEBUG] Output:     (void) SAR_SATURATE_INTR(base);
      [DEBUG] Output:            ^~~~~~~~~~~~~~~~~~~~~~~
      [DEBUG] Output: BUILD/tests/CYBCKIT_062_WIFI_BT_PSA/ARM/targets/TARGET_Cypress/TARGET_PSOC6/psoc6pdl/drivers/include/cy_device.h:464:71: note: expanded from macro 'SAR_SATURATE_INTR'
      [DEBUG] Output: #define SAR_SATURATE_INTR(base)            (((SAR_V1_Type *)(base))->SATURATE_INTR)
      [DEBUG] Output:                                            ~~~~~~~~~~~~~~~~~~~~~~~~  ^~~~~~~~~~~~
      [DEBUG] Output: In file included from ./TESTS/psa/spm_smoke/COMPONENT_NSPE/main.cpp:27:
      [DEBUG] Output: In file included from BUILD/tests/CYBCKIT_062_WIFI_BT_PSA/ARM/features/frameworks/utest/utest/utest.h:25:
      [DEBUG] Output: In file included from BUILD/tests/CYBCKIT_062_WIFI_BT_PSA/ARM/features/frameworks/utest/utest/utest_types.h:25:
      [DEBUG] Output: In file included from BUILD/tests/CYBCKIT_062_WIFI_BT_PSA/ARM/features/frameworks/utest/utest/utest_shim.h:28:
      [DEBUG] Output: In file included from BUILD/tests/CYBCKIT_062_WIFI_BT_PSA/ARM/features/frameworks/utest/utest/utest_scheduler.h:25:
      [DEBUG] Output: In file included from /builds/ws/mbed-os-ci_build-ARM@2/mbed-os/hal/ticker_api.h:25:
      [DEBUG] Output: In file included from BUILD/tests/CYBCKIT_062_WIFI_BT_PSA/ARM/targets/TARGET_Cypress/TARGET_PSOC6/psoc6pdl/devices/include/device.h:30:
      [DEBUG] Output: In file included from BUILD/tests/CYBCKIT_062_WIFI_BT_PSA/ARM/targets/TARGET_Cypress/TARGET_PSOC6/objects.h:205:
      [DEBUG] Output: BUILD/tests/CYBCKIT_062_WIFI_BT_PSA/ARM/targets/TARGET_Cypress/TARGET_PSOC6/psoc6pdl/drivers/include/cy_ctdac.h:968:12: warning: expression result unused; assign into a variable to force a volatile load
[-Wunused-volatile-lvalue]
      [DEBUG] Output:     (void) CTDAC_INTR(base);
      [DEBUG] Output:            ^~~~~~~~~~~~~~~~
      [DEBUG] Output: BUILD/tests/CYBCKIT_062_WIFI_BT_PSA/ARM/targets/TARGET_Cypress/TARGET_PSOC6/psoc6pdl/drivers/include/cy_device.h:920:74: note: expanded from macro 'CTDAC_INTR'
      [DEBUG] Output: #define CTDAC_INTR(base)                   (((CTDAC_V1_Type *) (base))->INTR)
      [DEBUG] Output:                                            ~~~~~~~~~~~~~~~~~~~~~~~~~~  ^~~~
      [DEBUG] Output: In file included from ./TESTS/psa/spm_smoke/COMPONENT_NSPE/main.cpp:29:
      [DEBUG] Output: BUILD/tests/CYBCKIT_062_WIFI_BT_PSA/ARM/components/TARGET_PSA/inc/psa_manifest/sid.h:21:10: fatal error: 'autogen_sid.h' file not found
      [DEBUG] Output: #include "autogen_sid.h"
      [DEBUG] Output:          ^~~~~~~~~~~~~~~
      [DEBUG] Output: 17 warnings and 1 error generated.
```

Figure 32: Mbed OS Build logs with verbose disabled and enabled

4. Conclusions

This thesis described the results of a project performed on Mbed OS. The overarching goal of this project was to develop and empirically evaluate safe test selection and optimization techniques that can aid Mbed OS developers by providing them with quick feedback as well as reduce instability and unnecessary testing in Mbed OS Jenkins CI.

Mbed OS RT is important but resource intensive, and sometimes unstable, process. This thesis studied Mbed OS, Mbed OS PR's, and Mbed OS Jenkins CI to understand, identify, evaluate, research, and develop techniques to solve the most common root causes of unstable, unnecessary, or ineffective Mbed OS RT.

The empirical analysis of the researched, and developed Mbed OS RTS and RTO techniques show promising results. Several developed Mbed OS RTS and RTO techniques have already been adopted in Mbed OS Jenkins CI. JobRTS reduces overall Mbed OS Jenkins CI PR testing by less than 10%. TargetRTS reduces, or at least has the potential to reduce, overall Mbed OS Jenkins CI PR testing by over 25%. GreenteaRTS reduces Mbed OS Jenkins CI PR Greentea testing by over 90% on average. Building Mbed OS with verbose disabled can be over 60 times faster than building with verbose enabled.

4.1. Threats to validity

The MbedRTT implementations may contain bugs. The figures in this thesis may contain bugs or inaccuracies. To mitigate the risks, well-known languages, techniques, and tools were used when available. Unit, and integration tests were developed and executed for both the MbedRTT scripts and the scripts used in generating the figures.

The GitHub PR's included in this thesis include over 99% of all closed, merged, and currently open Mbed OS PR's from all forks and branches to all ARMmbed Mbed OS branches. The only PR's not included are either empty, corrupted, or otherwise unavailable PR's. It is quite difficult to automatically collect accurate Mbed OS PR test information, such as what were the names of the modified files in a PR at the time of the test execution, as the information available is either stored for only a limited time (AWS S3), or somewhat inaccurate (GitHub). This thesis considers the PR related information (such as the percentages of the different subsets of modified files) to be constant in most cases, so some of the results may be too generalized.

4.2. Application possibilities

MbedRTT is currently used only in Mbed OS Jenkins CI. The developed MbedRTT techniques are not Mbed OS Jenkins CI dependent. JobRTS and TargetRTS can be used locally, for example, to simply determine what testing will happen in Mbed OS Jenkins CI. A subset of the selected tests can then be executed locally in order to determine whether Mbed OS Jenkins CI is likely to pass or not. The local usage of GreenteaRTS can use either Mbed OS Jenkins CI Greentea results database or local Greentea results database. Mbed OS Jenkins CI Greentea results database can be used for either both downloading and uploading Greentea results, or just downloading Greentea results.

JobRTS is not Mbed OS specific. JobRTS can fairly easily be used in other projects, for example, to skip unnecessary testing for files in specific folders or files with specific file extensions. TargetRTS is Mbed OS specific. TargetRTS can be used to prioritize target/toolchain combinations by only including resource files in modified files. The most likely target/toolchain combinations to require testing are the target/toolchain combinations with intersecting resource files with the modified resource files. The technique used in GreenteaRTS is not Mbed OS specific. The technique requires test units, for example, test binaries, and their dependencies to be hashable and deterministic.

REFERENCES

[1] "Welcome to Python.org", *Python.org*, 2019. [Online]. Available: https://www.python.org/. [Accessed: 10- May- 2019].

[2] "ARMmbed/mbed-os", *GitHub*, 2019. [Online]. Available: https://github.com/ARMmbed/mbed-os. [Accessed: 03- Apr- 2019].

[3] "ARMmbed/mbed-os-tools", *GitHub*, 2019. [Online]. Available: https://github.com/armmbed/mbed-os-tools. [Accessed: 03- Apr- 2019].

[4] "OpenTMI/opentmi", *GitHub*, 2019. [Online]. Available: https://github.com/OpenTMI/opentmi. [Accessed: 03- Apr- 2019].

[5] "Mbed OS | Mbed", *Arm Mbed*, 2019. [Online]. Available: https://www.mbed.com/en/platform/mbed-os/. [Accessed: 04- Apr- 2019].

[6] "ARMmbed/mbed-os-tools", *GitHub*, 2019. [Online]. Available: https://github.com/ARMmbed/mbed-os-tools/tree/master/packages/mbed-greentea. [Accessed: 04- Apr- 2019].

[7] Nptel.ac.in. (2019). [online] Available at: https://nptel.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Embedded%20systems/Pdf/Lesson-38.pdf [Accessed 19 Apr. 2019].

[8] Pipeline. (2019). *Pipeline*. [online] Available at: https://jenkins.io/doc/book/pipeline/ [Accessed 19 Apr. 2019].

[9] Extending with Shared Libraries. (2019). *Extending with Shared Libraries*. [online] Available at: https://jenkins.io/doc/book/pipeline/shared-libraries [Accessed 19 Apr. 2019].

[10] Os.mbed.com. (2019). *Continuous integration (CI) testing - Contributing | Mbed OS 5 Documentation*. [online] Available at: https://os.mbed.com/docs/mbed-os/v5.12/contributing/ci.html [Accessed 27 Apr. 2019].

[11] Rachel Potvin, J. (2019). *Why Google Stores Billions of Lines of Code in a Single Repository*. [online] Cacm.acm.org. Available at: https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext [Accessed 28 Apr. 2019].

[12] Static.googleusercontent.com. (2019). [online] Available at: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45861.pdf [Accessed 28 Apr. 2019].

[13] Mir.cs.illinois.edu. (2019). [online] Available at: http://mir.cs.illinois.edu/~marinov/publications/LegunsenETAL16StaticRTS.pdf [Accessed 28 Apr. 2019].

[14] Anon, (2019). [online] Available at: https://www.researchgate.net/publication/295908347_Dynamic_Integration_Test_Selection_Based_on_Test_Case_Dependencies [Accessed 28 Apr. 2019].

[15] Mir.cs.illinois.edu. (2019). [online] Available at: http://mir.cs.illinois.edu/~marinov/publications/GligoricETAL15EkstaziDemo.pdf [Accessed 7 May 2019].

[16] M. Gligoric, L. Eloussi and D. Marinov, "Ekstazi: Lightweight Test Selection," *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Florence, 2015, pp. 713-716.

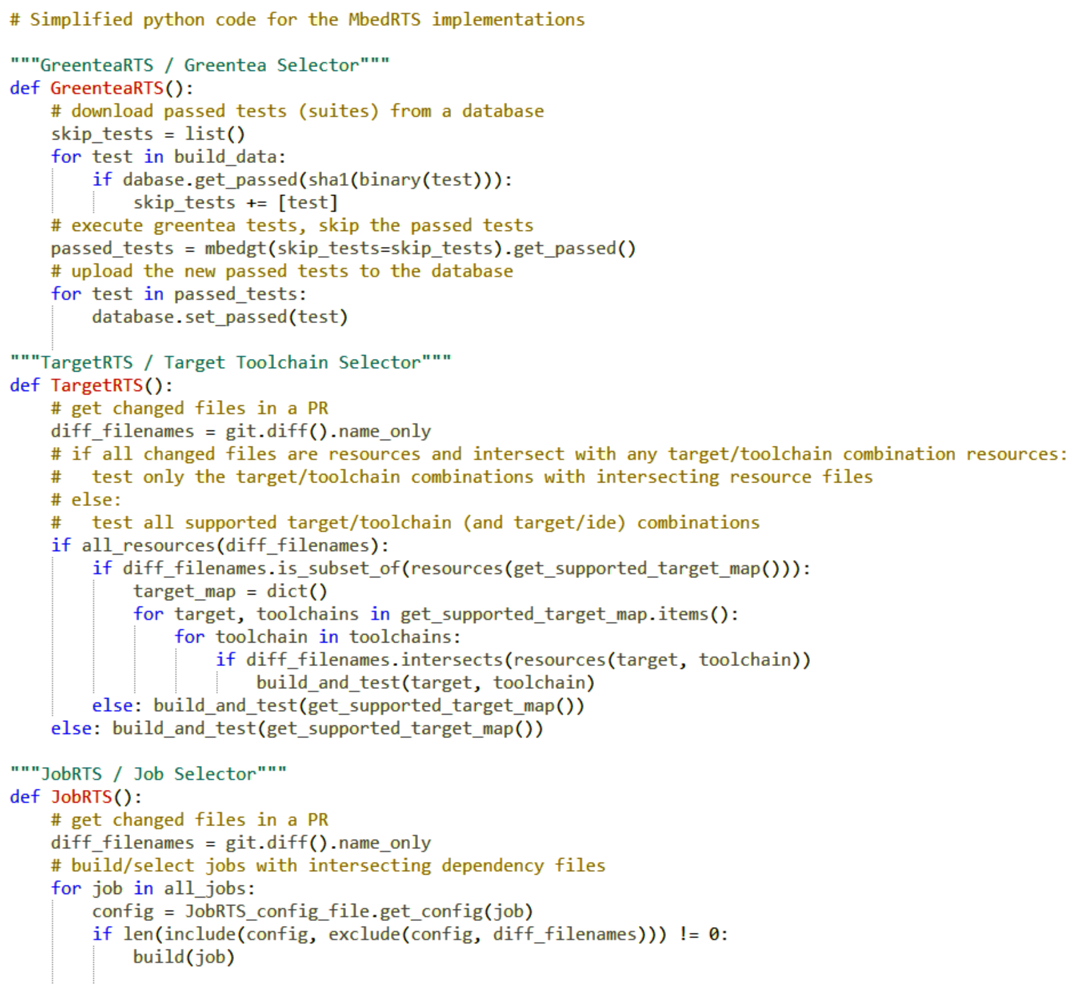
[17] Mir.cs.illinois.edu. (2019). [online] Available at: http://mir.cs.illinois.edu/~qluo2/fse14LuoHEM.pdf [Accessed 7 May 2019].

[18] Mbed-os-ci.s3-website-eu-west-1.amazonaws.com. (2019). [online] Available at: http://mbed-os-ci.s3-website-eu-west-1.amazonaws.com/?prefix=jenkins-ci/ARMmbed/mbed-os/ [Accessed 8 May 2019].

5. APPENDICES

Appendix 1. Simplified MbedRTT python implementations

Appendix 1. Simplified MbedRTT python implementations



```python
# Simplified python code for the MbedRTS implementations

"""GreenteaRTS / Greentea Selector"""
def GreenteaRTS():
    # download passed tests (suites) from a database
    skip_tests = list()
    for test in build_data:
        if dabase.get_passed(sha1(binary(test))):
            skip_tests += [test]
    # execute greentea tests, skip the passed tests
    passed_tests = mbedgt(skip_tests=skip_tests).get_passed()
    # upload the new passed tests to the database
    for test in passed_tests:
        database.set_passed(test)

"""TargetRTS / Target Toolchain Selector"""
def TargetRTS():
    # get changed files in a PR
    diff_filenames = git.diff().name_only
    # if all changed files are resources and intersect with any target/toolchain combination resources:
    #   test only the target/toolchain combinations with intersecting resource files
    # else:
    #   test all supported target/toolchain (and target/ide) combinations
    if all_resources(diff_filenames):
        if diff_filenames.is_subset_of(resources(get_supported_target_map())):
            target_map = dict()
            for target, toolchains in get_supported_target_map.items():
                for toolchain in toolchains:
                    if diff_filenames.intersects(resources(target, toolchain))
                        build_and_test(target, toolchain)
        else: build_and_test(get_supported_target_map())
    else: build_and_test(get_supported_target_map())

"""JobRTS / Job Selector"""
def JobRTS():
    # get changed files in a PR
    diff_filenames = git.diff().name_only
    # build/select jobs with intersecting dependency files
    for job in all_jobs:
        config = JobRTS_config_file.get_config(job)
        if len(include(config, exclude(config, diff_filenames))) != 0:
            build(job)
```