Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Roman May

Advanced Testbed Resource Allocation

Semester Project
Januar to April 2016

Supervisor: Prof. Dr. Lothar Thiele
Supervisor: Roman Lim

Abstract

FlockLab [1] is a testbed for embedded wireless sensor network applications, built in 2012. It is publicly accessible via a web front-end. Over the past few years the number of users accessing FlockLab and hence the utizilation has increased drastically [2]. Since FlockLab is only capable of running one test at a time, this has led to increased waiting times for all users.

To counter this problem, the goal of this project is to increase the test throughput of FlockLab by modifying it to run tests concurrently. To achieve this goal we modify FlockLab on a test system and provide a scheduling for parallel tests.

In the evaluation we compare different variants of our scheduling algorithm and verify the scheduling. Further, we execute tests with real observers of FlockLab to verify that our setup works with real hardware. Finally, we compare sequential and parallel scheduling. The results show that parallel scheduling significantly increases the test throughput compared to sequential scheduling.

Contents

List of Figures

List of Tables

Listings

1. Introduction

1.1. Motivation and Contributions

In the development of wireless sensor network applications, testbeds are indispensible. They provide great opportunites for application debugging and performance and power measurements. FlockLab [1] is a publicly accessible wireless sensor network testbed located at ETH Zürich, that supports multiple wireless sensor nodes and provides accurate measurements and helpful functions for debugging. Since the start of operation in 2012, the number of users, as well as the number of executed tests, have grown (see figure 1). The increasing number of tests, and hence higher utilization of FlockLab, has led to increased waiting times for the users.

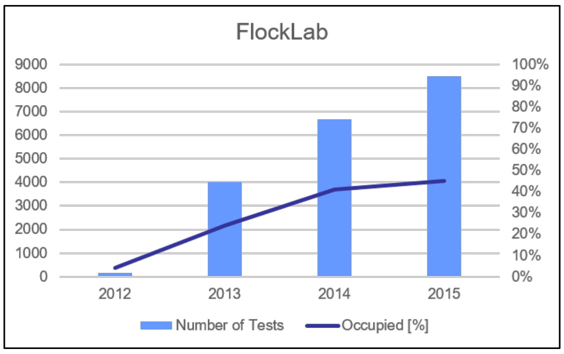


Figure 1: Number of tests over the last years [2].

FlockLab is only capable of running one test after another even if only parts of the testbed are used. In this project, we aim to modify FlockLab to run tests in parallel on unused components, thus increasing the test throughput.

At the moment, FlockLab consists of 32 observers. Observers are platforms that host up to four slots for sensor nodes and implement the necessary hard- and software for the different services of FlockLab. This gives us the opportunity to not only run tests in parallel on different observers, but also on different target nodes on the same observer, given that the tests don't interfere with each other.

First, we create a concept on how to allow concurrent tests, that includes a model of the resource constraints. The resource constraints are modeled with hardware resources and a mapping between the test configurations and these resources. Afterwards, we use this model to develop an algorithm, which checks the schedulability of new tests and schedules them if it's possible. On a test system, we use this concept to modify FlockLab for running test in parallel.

In the evaluation, we verify the proper scheduling of the algorithm, measure it's performance, execute parallel tests on real observers of FlockLab and finally, compare sequential and parallel test scheduling.

This report is organized as follows. First, we cover related work in section 2, continue with background information in section 3 and elaborate our concept in section 4. The implementation is covered in section 5, followed by the evaluation in section 6. Finally, we present an outlook and a conclusion in section 7 and 8, respectively.

2. Related Work

As already mentioned in the previous section, our project is based on FlockLab. We provide an extension to run parallel tests. On FlockLab, this is the first project towards parallelization of tests.

There are several other testbeds for WSN, but there is often little information available about parallel running tests. For example Kansei [3] and Twist [4] provide no clear information about parallel tests. However, there are still some testbeds that either indicate parallel tests like MoteLab [5], NetEye [6] and DSN [7] or clearly state it like SenseNeT [8] and MoteMaster [9]. Indriya [10] doesen't provide information about parallel tests in the paper, but since it's based on MoteLab, it's likely that it supports the same features.

MoteLab developed a feature that allows parallel tests in different lab zones, while DSN supports the use of multiple servers with different DSN networks. The difference to our solution is that they only allow parallel tests in different zones or networks, while we support parallelization on observer and even node level.

NetEye, SenseNet and MoteMaster let the user chose the target nodes separately and therefore allow parallel tests on node level. However, our solution has the possibility of using different target nodes on the same observer in parallel.

Furthermore, it's often not clear from the descriptions of the different testbeds if they provide some kind of automatic scheduling like we do with the possibility to run tests as soon as possible. Also, the scheduling is more involved on FlockLab due to its complex hardware that supports more than just programming and logging of target nodes.

3. Background

3.1. Wireless Sensor Networks

With the advance in technology, the need for accurate, real-time environment information arose. Wireless Sensor Networks (WSN) [11] are networks of small, low-power sensor nodes capable of wireless communication. These small embedded systems typically consist of one or more sensors, a data processing unit and communication components. The sensor nodes build a mobile ad-hoc network and send the processed sensor data to a sink.

Examples for WSN applications are:

- Permafrost data sensing [12]
- Structural monitoring [13]

Development of WSN applications is a non-trivial task. They usually have high requirements in terms of low power consumption and resource usage. Furthermore, the debugging possibilities of WSN are limited. While simulations are helpful for debugging, they lack precise measurements. WSN testbeds are experimentation platforms that can be used to evaluate WSNs on real hardware and often provide multiple measurement possibilities along with useful services for debugging.

4. Design

In this section, we first provide detailed information about the parts of FlockLab that are affected by this project. Next, we describe the setup of our test system, followed by a conecpt on how to alter FlockLab for parallel tests.

4.1. FlockLab

Before explaining the modifications of FlockLab, some components and their functioning is outlined. In the following, we first provide a brief overview of FlockLab and then explain all parts that are important for our project in detail.

4.1.1. Overview

FlockLab is a testbed for WSN applications. It supports various sensor node platforms and provides multiple services to debug and evaluate WSN applications. These services are GPIO tracing, GPIO actuation, power profiling, adjusting the supply voltage and serial I/O.

FlockLab currently consists of more than 30 observers. They contain a Gumstix XL6P COM, slots for up to four target nodes and additional components for measurements and for controlling the sensor nodes.

The front-end is a publicly accessible web interface where users can submit tests, get the results and, if necessary, abort them. Furthermore it provides information about the observer deployment, link qualities between the different observers, etc.

Several servers build the back-end infrastructure: The NTP server is used for time synchronization. The Web server provides the user interfaces and to schedule tests. The test management server is responsible to copy the target binaries to the right observers, starting and stopping of tests and collecting the test results. The database server hosts a MySQL database for data storage and the monitoring server is used to detect malfunctions of FlockLab.

4.1.2. Webserver

The webserver provides the user interface to administer tests and supplies the user with information about the state of FlockLab. To submit a new test, the user creates an XML file for the test configuration and submits it via the web interface. The possible configuration blocks in the XML file can be found below. Only the relevant parts of each block are described.

General Configuration Each test needs exactly one general configuration block. It defines when the test should be started. The two options are as soon as possible

(*ASAP*) or at a predefined, absolute time (*absolute time*). The test duration has to be specified for *ASAP* tests, while the end time is needed for *absolute time* tests.

Target Configuration One or more target configuration blocks define which image for the target architecture is used on each observer. The images can either be previously uploaded via the web interface or embedded in the configuration file[1].

Embedded Image Configuration For each in a target configuration referenced embedded image, there has to be an embedded image configuration block, that holds an image for the target architecture.

Serial Configuration In serial configuration blocks, the user can configure the use of the serial I/O service for one ore more observers. Additionally, the port that is used for the service can be configured. The two options are *serial* or *usb*.

GPIO Tracing Service Configuration The GPIO tracing service block holds the configuration for the GPIO tracing service. It's possible to configure it once for multiple observers if the use the same configuration or to use multiple blocks.

GPIO Actuation Service Configuration This configuration part is similar to the GPIO tracing configuration, but for the GPIO actuation service.

Power Profiling Service Configuration This configuration part is similar to the GPIO tracing configuration, but for the power profiling service.

After the user uploads a new test configuration file the webserver first checks if the configuration file is valid and schedules the test if possible. If the user configured the test to run *ASAP*, the scheduling calculates the next possible time for the test and stores it in the database. For *absolute time*, the webserver only submits the test to the database if no other test is planned at the desired time, else it's declined.

4.1.3. Database

The database contains tables with information about FlockLab's status, components, users and tests. In the following, all tables that are important for this project are described.

tbl_serv_observer This table has entries for all observers in the Network. For each observer there are fields for networking information, observer status and the ids of the target adapters connected to each of the four slots.

tbl_serv_tg_adapt_list Each target adapter used in FlockLab is listed in this table and linked to the corresponding target adapter type.

tbl_serv_tg_adapt_types The different target adapter types are listed in this table. Each of this entries is linked to the corresponding target architecture in the next table.

[1] Mote Runner is another option, but it's not covered in our project and therefore left out.

tbl_serv_architectures As mentioned before, each observer has four slots for target adapters that can be connected to a target platform. All possible target architectures are listed in this table.

tbl_serv_tests If the webserver schedules a new test, it is saved in this table. The title and description, start and end time, the configuration file and status of each test are stored along with other information.

tbl_serv_map_test_observer_targetimages An entry for each used observer in a test is created in this table in additional to the entry in the previous table. It holds the information about the target image id used in a test for each observer.

tbl_serv_targetimages The target images for the target platforms are stored in this table. This can be done either manually via webinterface or the webserver will do it automatically if a test is submitted with an embedded image in the configuration file.

4.1.4. Test Management Server

The test management server is responsible for the test management. It periodically fetches tests from the database, prepares all target images and configurations for the observers, starts and stops the tests and fetches the results. Finally, it provides the results for the user. For this task, the server uses multiple python scripts. The most important ones for our project are the following:

flocklab_scheduler.py The scheduler script runs periodically on the test management server. It checks if tests have to be started, stopped or aborted. If so, it starts the dispatcher for this tests accordingly.

flocklab_dispatcher.py The dispatcher is called by the scheduler for a single test with different modes for starting, stopping and aborting a test. To start a test, it fetches the test configuration files and target images, prepares them for all used observers and copies them to the observers. Eventually, it calls the script *flocklab_starttest.py* on the observers and invokes the fetcher. To stop or abort a test, the script *python_stoptest.py* is invoked on all used observers and then the dispatcher waits for the fetcher to finish.

flocklab_fetcher.py The fetcher collects all test results from the different observers and copies them to the test management server. Additionally, it processes the fetched test results.

4.1.5. Observers

Each observer is equipped with an embedded computer to control the observer, four slots for the target nodes, an SD card to buffer the test results and additional components for control, measurement or communication. When the script *python_starttest.py* is called, the observer prepares the target architecture, starts the test on it and controls configured services like power profiling or GPIO tracing. To stop a test, the script *flocklab_stoptest.py* is called.

At the moment, FlockLab supports the following target platforms:

- TinyNode
- Tmote Sky
- Opal
- Iris
- Mica2
- Wismote
- CC430
- Asynchronous communication module (ACM2)
- OpenMote
- Dual Processor Platform (DPP)

4.1.6. Test Cycle

A single test is proceeded as follows:

1. The user submits a test configuration file.
2. The webserver schedules the test.
3. The test is then stored in the database.
4. The scheduler on the test management server fetches the test and calls the dispatcher.
5. The dispatcher prepares the test files, copies them to the used observers and starts the test. Additionally, it starts the fetcher.
6. The test runs on the observers and test data is generated.
7. The fetcher periodically collects test data from the observers.
8. The scheduler gets the stop time of the test from the database and calls the dispatcher to stop the test.
9. The dispatcher stops the test on the observers and tells the fetcher to clean up.
10. The fetcher collects the last test data and processes it.
11. The user gets the test results.

A schematic overview of this process is shown in figure 2.

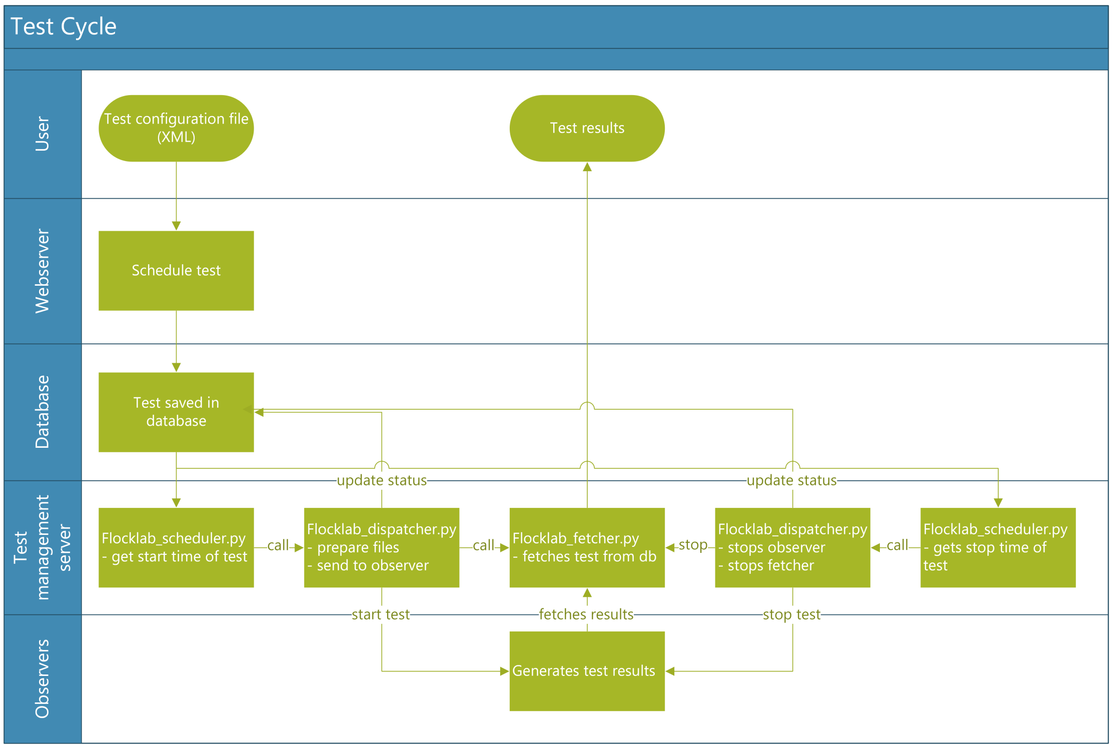


Figure 2: Test Cycle: Proceeding of a single test.

4.2. Test System Setup

In this section, we describe the setup of our test system. We used a notebook with Ubuntu 14.04 LTE and installed each component needed on this system. All of the necessary files can be found in the FlockLab SVN repository[2]. In the following we describe the setup of all components.

Database We installed a MySQL server on the system and created the database *flocklab*. To add all tables and insert the content we used the script *flocklab_server_db.sql*, which is a dump from database of FlockLab.

Webserver Apache2 was installed as webserver. We copied the data needed for the website from the SVN repository and configured apache accordingly.

Test Management Server The test management server is basically a collection of scripts and automated tasks. Therefore, we just copied the needed scripts to our system and changed minor parts of them to suit our setup. These changes were for example the paths to the different scripts in the configuration files. Additionally, we set up cronjobs to run scripts, like the scheduler, periodically.

Observer For our purpose, it was sufficient to create an additional user profile that was accessible via ssh to simulate an observer. We used the home directory of this user on the notebook to create folders for the test images and configurations as well as

[2] `https://svn.ee.ethz.ch/flocklab/`

for the test results and the scripts. Furthermore, we modified the configuration files again to suit our simulated observer.

4.3. Concept

In order to modify FlockLab for running tests in parallel, we first introduce our concept: We model the resource constraints and use this model for the test scheduling. To do so, we define a set of exclusive resources, as well as a mapping between these resources and a test configuration. Exclusive resources are resources which can only be used by one test at a time. To find them, we first examine the different components of FlockLab. The next step is to find the relation between a test configuration and the resources. For example an architecture or service that uses a specific hardware component in a test.

When a new test is submitted, we calculate time intervals during which the same exclusive resources are used. This resource usage time intervals are then compared to previously scheduled tests in the database to find the next free time slot for *ASAP* tests or to decide if the test can be accepted or has to be declined for *absolute time* tests.

4.3.1. Resource Constraints

To define the exclusive resources in FlockLab we have to find all components of FlockLab which can not be shared by different tests. Each component has to be examined separately to get all exclusive resources. First, we analyze the hardware of a single observer: There is an embedded computer, four sensor node platforms, a multiplexer, an USB hub and some test independent parts that are not affected by parallel running tests (see figure 3).

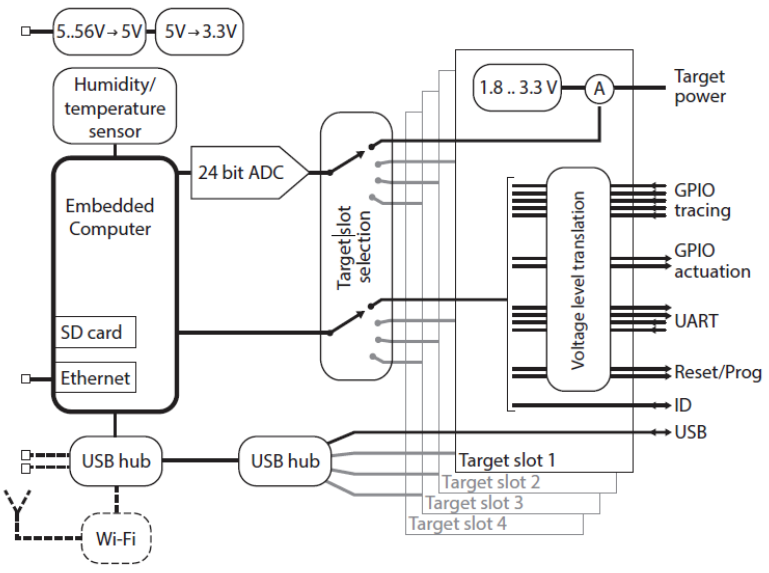


Figure 3: Model of an observer [1].

We start with the target slots. They are exclusive resources because the target platforms can only run one test image at a time. The multiplexer is used to send and receive information either directly from a sensor node or from one of the different services like power profiling. The multiplexer can only select one target slot at the same time, thus we model it as an exclusive resource.In contrast to the multiplexer, the USB hub can access all slots simultaneously. Hence, we neglect it in the model of the resource constraints. The embedded computer is capable of multitasking and therefore not modeled as an exclusive resource. However, whether the embedded computer can manage the additional load or not, is a different aspect and is covered in the section 4.3.2.

After the observer hardware is examined, there is still another factor which has to be considered on a single observer: The frequency used by the targed node's wireless communication system. As there are three different frequencies used by the different target platforms, 2.4 GHz, 868 kHz and 433 kHz, we model each frequency as a single exclusive resource. The other components, the webserver, the database and the test management, are capable to run parallel tasks and are therefore not exclusive resources.

In conclusion, we define the following exclusive resources for each observer of FlockLab:

- Platform slot 1
- Platform slot 2
- Platform slot 3
- Platform slot 4
- Frequency 2.4 GHz
- Frequency 868 kHz
- Frequency 433 kHz
- Multiplexer

After the above definition of the exclusive resources in FlockLab, the next step is to find a method to map a test configuration to this resources. Additionally, we can use information from the database about the observers and the target architectures. In the following, we examine all exclusive resources and specify the information needed for the mapping.

First, we examine the platform slots of the observers. As mentioned before, we can only run one test on a single target at the same time. We can combine the information from the configuration file about the used observer and platforms with data from the database to get the used slots. With the target architecture determined, the frequency is given as well.

To get the multiplexer usage on each observer, we first have to find all services and tasks that use the multiplexer. These services are: GPIO tracing, GPIO actuation, power profiling, adjusting the supply voltage and serial I/O tracing over the serial interface. Furthermore, the multiplexer is used for operating the Reset/Prog pin. The observers start and stop a test on the target platform with this pin. During the setup and the cleanup of a test, the multiplexer is used as well. The information if and when these services are used and the time intervals for the start, stop, setup and cleanup phase, are contained in the configuration file.

Summarized, we need the following information for each test and observer:

• observer Id + architecture	⇒ slots numbers
• architecture	⇒ frequencies
• used services + start/stop + setup/cleanup	⇒ multiplexer

With this mapping between test configurations and the exclusive resources we finished the model the resource constraints. The next step is to find a representation of the resource usage that can be used for the scheduling of new tests.

We chose to format the resource usage as a list of used resources in a time interval. Since a test normally doesn't use all resources at all times of a test, we first calculate all time intervals in which the resource usage is the same. Then, to store this information in a table of the database, we generate arrays which have the start time of the interval as the first and end time as the second element, followed by bits which represent the resource usage. An example of a resource table[3] is shown in table 1. In the following we will reference to one line of this table as a *resource array* and to the table itself as the *resource table*.

Table 1.: Example of a resource usage table.

Start	**End**	**Res. 1**	**Res. 2**	**Res. 3**	**...**	**Res. n**
12:00:00	12:03:00	1	0	0	...	0
12:03:01	12:06:01	1	1	0	...	1
12:06:02	12:09:02	1	0	0	...	0

When a new test is submitted, we can now calculate the resource usage of the test and compare it to the *resource arrays* from the database for the scheduling. The scheduling algorithm is explained in detail in section 4.4.

4.3.2. Performance Considerations

In this section, we cover possible performance issues caused by parallel running tests. Two components of FlockLab are handled: The embedded computer on an observer and the bandwidth needed by the test management server for the test data collection. In the following, we explain why we did not consider them in our model. However, if tests prove our assumption as wrong, it would be necessary to extent our model to cover those resources as well.

Both, the bandwidth used for the data collection and the load on the embedded computer of an observer are proportional to the amount of produced data by a test. We assume, that all components of FlockLab can handle any arbitrary, single test without running on it's limits. A test that runs on all observers and uses all monitoring services clearly produces the highest amount of data, but at the same time only allows parallel tests that do not use the multiplexer on any of the observers. Therefore, an additional test can only use the serial I/O service over USB which generates significantly less data. We assume, that this relatively small amount of additional data will neither

[3] The start and stop times in the example are changed to a readable form. For all calculations seconds from the epoch are used.

push the embedded computer to it's limit, nor exceed the available bandwidth for the data collection.

4.4. Scheduling Algorithm

In this section, we explain our scheduling algorithm which uses the previously built model to schedule new tests. As mentioned before, the user can choose between two modes: *ASAP* and *absolute time.* We first go through the *absolute time* mode and then extend the algorithm to support *ASAP* mode.

4.4.1. Absolute Time Mode

A flow chart of the scheduling algorithm for *absolute time* mode is depicted in figure 4. It gets a test configuration file as an input, has access to the data in the database and outputs a boolean that indicates if it's possible to schedule the test at the desired time or not.

First, the algorithm calculates the usage of each exclusive resources from the configuration file. Then it calculates time intervals in which the resource usage of the test is the same and generates *resource arrays.* Next, the algorithm fetches the *resource arrays* which overlap in time from the *resource table.* Afterwards, the resource usage in the time intervals is compared. The new test is accepted and added to the database if the resource usage in overlapping time intervals is distinct, else it's declined.

4.4.2. ASAP Mode

We slightly change the previous algorithm to support *ASAP Mode* as well. Compared to the *absolute time* mode, the algorithm doesn't have to accept or decline a test. It has to return the next possible time slot to execute the submitted test. We extended the previous algorithm to get a first time slot by using the actual time plus the setup time, which is needed to prepare a test, as start time. The end time is simply the start time plus the duration of the test. Then we use a slightly modified algorithm for *absolute time* tests and increase the starting time each time the algorithm returns *decline test* until we get a time slot that is accepted. A schematic of the extended algorithm is provided in figure 5.

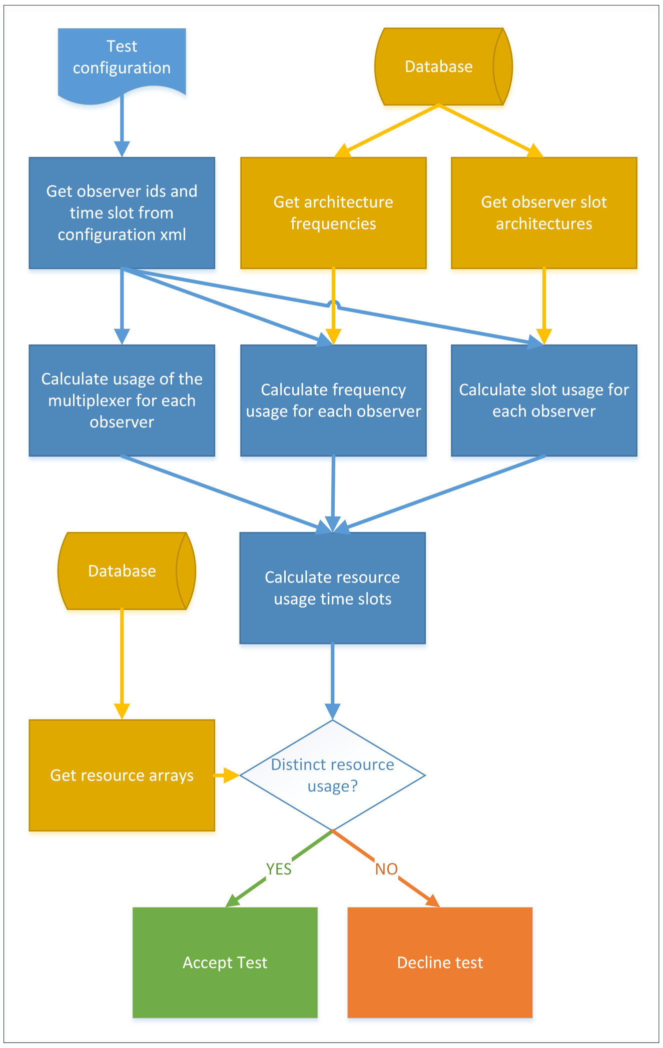

Test configuration
Database
Get observer ids and time slot from configuration xml
Get architecture frequencies
Get observer slot architectures
Calculate usage of the multiplexer for each observer
Calculate frequency usage for each observer
Calculate slot usage for each observer
Database
Calculate resource usage time slots
Get resource arrays
Distinct resource usage?
YES
NO
Accept Test
Decline test

Figure 4: Flow chart of the scheduling algorithm for absolute times.

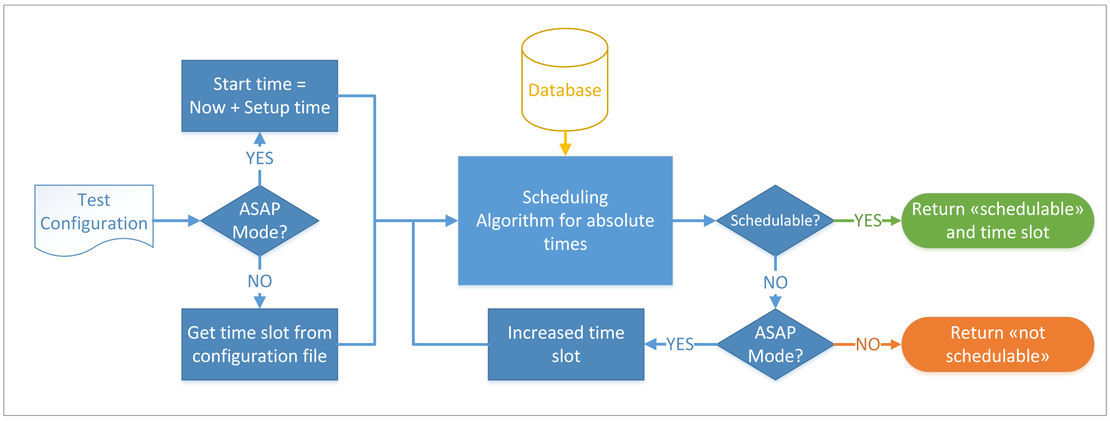

Start time = Now + Setup time
YES
Test Configuration
ASAP Mode?
NO
Get time slot from configuration file
Database
Scheduling Algorithm for absolute times
Schedulable?
YES
Return «schedulable» and time slot
NO
Increased time slot
YES
ASAP Mode?
NO
Return «not schedulable»

Figure 5: Schematic of the extended scheduling algorithm that supports ASAP mode.

5. Implementation

In this section, we first discuss the necessary modifications of the different parts of FlockLab to support parallel tests. Then we show the implementation of the scheduling algorithm. The scripts are explained either directly or with the help of pseudo code.

5.1. Database

First, we have to store the information about which frequencies are used by the different target platforms. As there is already a table *tbl_serv_architectures* in the database that contains information about the architectures, we simply add columns for all different frequencies, 2.4 GHz, 868 MHz and 433 Mhz. We fill this colums with either True if the architecture uses this frequency or False if it doesn't. Compared to just use one column with the used frequency as integer, this solution supports architectures which use multiple frequencies.

In FlockLab, each previously scheduled test is stored in the database in the table *tbl_serv_tests*. This table contains the start and stop times, the configuration file itself and some other information. Additionally, there is the table *tbl_serv_map_test_observer_targetimages* which contains a mapping between observers and tests. We keep this two tables as they are, but we also need information about the resource usage of the planned tests. Therefore, we create a new table *tbl_serv_test_resource* in the form of the previously explained *resource table* to store the *resource arrays*.

We chose the name of the resources and therefore the table columns to be in the form: *obs_<observer id>_<resource>* (e.g. *obs_007_mux* or *obs_202_slot_2*). The reason for this is that we can easily generate this names from the configuration and use a mapping table between the resource names and their positions in the *resource arrays*.

5.2. Test Management Server and Observers

On the test management side, we have to modify some of the scripts to support parallel tests. In the following, we explain the changes in the different scripts described in section 4.1.4, including the *flocklab_starttest.py* script on the observers from section 4.1.5.

flocklab_scheduler.py Compared to the sequential test management, it's now possible that multiple tests have to be started, stopped or aborted at the same time. We modify the script that it first fetches all tests which have to be started from the database. Then it starts separate threads for each of those tests. Afterwards, it does the same for the tests that have to be stopped or aborted. Additionally, we change the start thread to first alter the status of the test in the database to *planned* to prevent another scheduler from trying to start the same test again. Previously, this was done by the dispatcher. Finally, we add a delay to the start thread to prevent a test to be started before the planned time. This was no problem before, but with parallel tests, an early started test would falsify the *resource table*.

flocklab_dispatcher.py We change two parts in the dispatcher script. First, the dispatcher now saves the test configurations and images on the observer in a sub folder for this test and submits the test id as an argument when calling *flocklab_starttest.py*. Second, when scheduling a scheduler to stop the test, the dispatcher first checks if there is already a scheduler scheduled at that time. If there is already one it's not necessary to schedule another.

flocklab_starttest.py We change this script to use the image and configurations form the test id subfolder mentioned before. Furthermore, the observer now stores the test results in a subfolder for the test.

flocklab_fetcher.py The fetcher gets the test results from the corresponding test data folder on the observers. There is no other modification necessary as it already stores the test results in separate folders on the test management server.

With this modifications we make sure that neither the test configurations and images nor the test results of different tests are mixed up.

5.3. Webserver

On the webserver, PHP scripts used to schedule new tests when they are submitted. We changed this part to call a python script instead. This script schedules tests by using the concept explained in section 4.3.

5.4. Scheduling Algorithm

In this section, we show the new scheduling algorithm for parallel tests in detail. As already mentioned, the algorithm gets a new test configuration file as input and outputs if the test is schedulable. If so, the test is scheduled and stored in the database. The algorithm can be split in five parts: A general part, getting the resource usage from the configuration, merge the resource arrays, the actual scheduling of the test and finally adding the test to the database.

5.4.1. General

In the general section we execute tasks that are used by the other parts of the script. Besides standard tasks, like connecting to the database and creating a logger, the following is done:

Create Resource Name to Position Mapping The order of the different exclusive resources from the *resource table* in the database and in the *resource array* has to be the same, therefore, we first build a mapping between those. The algorithm fetches the titles of the columns and stores them in a python dictionary with the name of the resource as key and position as value to keep this mapping dynamic for name changes or additional resources. The dictionary is called *resourcePosition*.

Parse XML File The test configuration XML file is the main source to get the resource usage of a test. We parse it right at the beginning to make the different configuration parts easy accessible for the following tasks.

Get Observer Ids and Architectures All of our exclusive resources are dependent on the used observers. Hence, it is appropriate to get a list of them right at the beginning. Further, we store the information about which architecture is used on each observer. We need the architecture later to get the slot and frequency resources.

Get Test Times We need to fetch the start and end time of the test to get the time intervals in which the resources are used. This information can be found in the configuration for *absolute time* tests. For *ASAP test* on the other hand, we use the current time plus setup time as start time and use the test duration from the configuration to calculate the end time.

In the evaluation of our system we noticed that in some cases it's beneficial to round up the start time to the next full minute. The advantages and disadvantages of the rounding are explained in section 6.

5.4.2. Get Resource Usage

We use separate functions to get the usage of each of the different resource types frequency, slot and multiplexer. Each of this functions returns *resource arrays* with the usage of this resources. The reason for this is to make it as simple as possible to adapt this script for additional resources. To extend our algorithm for further resources, it is sufficient to add columns for the new resources in the *resource table* and write a function that generates *resource arrays* for them.

The functions have in common that they first generate a list with the names of the used resources together with the time intervals in which these resources are used. If the usage of the resources change throughout the test, multiple time intervals with the corresponding resource lists are calculated. To generate *resource arrays* out of the resource usage lists the functions utilize a helper function *getResourceArray*. *GetResourceArray* gets time intervals and a list of the used resources as input. It uses the *resourcePosition* dictionary to translate this list of resources to an ordered array with the start time as first and end time as second element, with the following elements indicating if the resource at this position is used.

An overview of the resource usage throughout a test is depicted in figure 6. In the figure, the multiplexer resource is split, since the time intervals can differ for varying service configurations. Multiplexer with and without service means with and without the use of services that use the multiplexer.

Resource: Slot We already stored the observer ids and the used architectures in the general section. Therefore, we can use this information together with data from the database to get the slot number of the used architecture on each observer and generate the resource usage list.

To get the time interval, we need to consider the setup und cleanup times of a test. The setup time is used to prepare the test on an observer, while the cleanup time is used

		Slot	Frequency	Multiplexer (w/o Services)	Multiplexer (w/ Services)
	Setup	used	free	used	used
Test Start					
	Start	used	used	used	used
	Run	used	used	free	used
	Stop	used	used	used	used
Test End					
	Cleanup	used	free	used	used

Figure 6: Overview of the resource usage throughout a test.

to fetch the last results and clean up on the observer. In both of this phases, the target architecture is already used. In conclusion, this resource generates a single time interval between *start time - setup time* and *end time + cleanup time.*

Resource: Frequency We can fetch the used frequencies from the database with the previously stored information about the used architectures.

Compared to the slot resource, the frequency is only used during the test, but not during setup and cleanup. Therefore, we get a single time interval from the start time to the end time of the test.

Resource: Multiplexer This functions scans through the XML files and fetches all service configurations. As mentioned in section 4, the multiplexer is used for the following services: GPIO tracing, GPIO actuation, power profiling and serial I/O service over the serial interface.

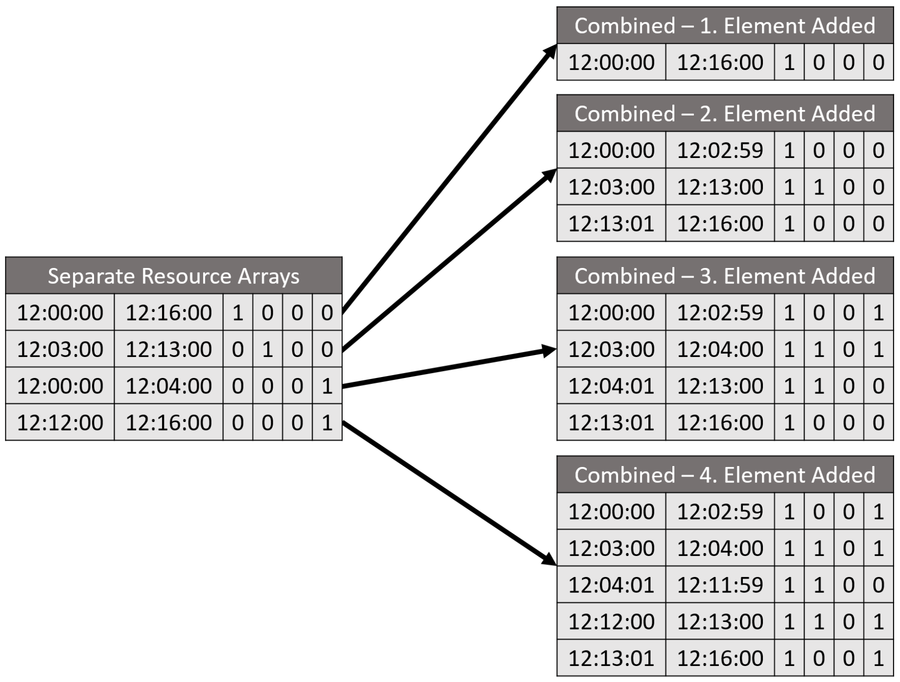
Apart from those services, the multiplexer is used to setup and cleanup a test, as well as to start and stop a test. Thus, each observer which uses one or more of the above services is used from *start time - setup time* until *end time + cleanup time.* If an observer is not configured to use those services, it is still needed for the setup and start phase as well as for the stop and cleanup phase of the test.

In conclusion, this function can produce up to three time intervals with different resource usage.

5.4.3. Merge Resource Arrays

Now we have multiple, overlapping *resource arrays* with distinct resources. The next step is to merge them in non-overlapping, timely ordered *resource arrays* before we can schedule the test. We do this by inserting them one by one to the right position of a new list and combining them if necessary. This process is best shown with the example depicted in figure 7.

The table on the left side shows four example *resource arrays* we want to merge. On the right side, we see the resulting table after adding each of the *resource arrays.* We search entries which overlap in time and merge the resource usage with an OR operation to add a new array. For the remaining time we insert new time intervals at the right position to keep the timely ordering. After this step, the *resource arrays* are ready for the scheduling algorithm.



Separate Resource Arrays					
12:00:00	12:16:00	1	0	0	0
12:03:00	12:13:00	0	1	0	0
12:00:00	12:04:00	0	0	0	1
12:12:00	12:16:00	0	0	0	1

Combined – 1. Element Added					
12:00:00	12:16:00	1	0	0	0

Combined – 2. Element Added					
12:00:00	12:02:59	1	0	0	0
12:03:00	12:13:00	1	1	0	0
12:13:01	12:16:00	1	0	0	0

Combined – 3. Element Added					
12:00:00	12:02:59	1	0	0	1
12:03:00	12:04:00	1	1	0	1
12:04:01	12:13:00	1	1	0	0
12:13:01	12:16:00	1	0	0	0

Combined – 4. Element Added					
12:00:00	12:02:59	1	0	0	1
12:03:00	12:04:00	1	1	0	1
12:04:01	12:11:59	1	1	0	0
12:12:00	12:13:00	1	1	0	1
12:13:01	12:16:00	1	0	0	1

Figure 7: Example of the merging of multiple, overlapping resource arrays.

5.4.4. Scheduling

In the course of this project, we developed two slightly different scheduling algorithms. They generate the same output, but access the database in different ways. In the following, we show the different variants of the scheduling and discuss the advantages and disadvantages of them.

The basic principle of both algorithms is same: They both consult the *resource table* in the database and check if the used resources are available in the given time intervals. If they are, the test can be scheduled. If not, for *ASAP* tests, the start and stop times of all resource arrays are increased and for *absolute times*, the test is declined. The time increment for the resource arrays is calculated as follows: We first locate the two *resource arrays* that overlap in time and use partly the same resources. This is always one from the new test and one from the database. Then we choose the time increment to be the minimum that still ensures this two *resource arrays* to not overlap in time in the next iteration. This method ensures that the start time of the test is minimally increased and does not lead to unnecessary long waiting times.

The procedure of the two algorithms can be found in pseudo code in listing 1 and 2. The difference between the two algorithms is, that the scheduling version one fetches only resource arrays from the database that overlap in time with the new test, while the scheduling version two fetches the whole database at the beginning. The advantage of scheduling version one is that it doesn't fetch more data from the database than necessary for one iteration, the disadvantage that it has to fetch data in each iteration. The scheduling version two once loads the whole resource table, but doesn't have to repeat it each iteration, which results in much faster iterations. Therefore, scheduling version one is better to minimize the memory usage, while version two is significantly faster if there are a lot of previously scheduled tests. The evaluation of the different versions can be found in section 6.

Listing 1: Scheduling version one

```
def schedule([List]resArrays):
    # repeat until the test is scheduled and manually exited
    while not scheduled:
        isSchedulable = True
        # Fetch resource arrays from db which overlap in time
        dbResArray = fetch_overlapping_resArrays_from_db()
        for new in resArrays:
            for old in dbResArray:
                if timeIntervalsOverlap(new,old):
                    # check if they use different resources (bitwise or)
                    if any([ x&y for (x,y) in zip(new[2:],old[2:])]):
                        isSchedulable = False
                        timeShift = old[1] + 1 - new[0]
                        break

            if not isSchedulable:
                break

        if not isSchedulable:
            if ASAP:
                resArrays = increaseTimes(resArray,timeShift)
                continue
            else:
                return False
        else:
            return True
```

Listing 2: Scheduling version two

```
def schedule([List]resArrays):
    # first get ALL resource arrays from the db
    dbResArray = fetch_resArrays_from_db()
    while not scheduled:
        timeShift = 0
        isSchedulable = True
        # loop through all old resource arrays
        for old in dbResArray:
            # compare them with the first one from the new test (+timeShift
                )
            if timeIntervalsOverlap(resArrays[0] + timeShift,old):
                if any([ x&y for (x,y) in zip(new[2:],old[2:])]):
                    # if the collide and mode is not ASAP -> return
                    if not ASAP:
                        return False

                    # Else, add the necessary timeshift and continue
                    else:
                        timeShift += old[1] + 1 - new[0]
                        continue

            # if the first resource array didn't collide apply timeShift to
                all intervals and check the others
            resArrays = increaseTimes(resArray,timeShift)

            isSchedulable,timeShift = checkOtherArrays(resArrays,old)
            if not isSchedulable:
                # continue with the new timeshift if the other did collide
                continue
            else:
                return True
```

5.4.5. Add Test to Database

Finally, if the scheduling returned a valid time slot, the only task left is adding the test to the database. While the entries in the tables *tbl_serv_tests* and *tbl_serv_map_test_observer_targetimages* are done exactly the same as in the original FlockLab, the *resource table* entries are a bit trickier. The reason for this is that we have to merge the entries with existing ones if they overlap in time. We do this by fetching all *resource arrays* from the *resource table* which overlap with the new ones and delete them in the database. Then we feed all *resource arrays*, the ones we fetched from the database and the ones from the new test, to the merging algorithm we used before. Finally, we can add the now non-overlapping arrays to the *resource table*.

This is exactly the point where the rounding of the start time to the next full minute can make a difference. While the whole algorithm works in exactly the same manner for both cases, the resulting *resource table* can be bigger if we don't round the start time. Assume two tests with the same configuration, but with different observers. We submit them in the same minute, but a few seconds apart. When we don't round the start time of the two tests, all time intervals will be shifted a few second. Thus, when we add the second test to the database, we have to split all of them. This results in the doubled number of *resource arrays* in the database. On the other hand, if we round the start time to the next full minute, the intervals are exactly the same and we only have to update the existing ones when adding the second test. The impact of this behavior on the time needed to schedule a new test is discussed in section 6.

6. Evaluation

To test our modifications we executed multiple experiments. We first verified the correct scheduling of our algorithm and measured it's performance on the test system with artificially generated tests. Then we tested our modifications with the real observers of FlockLab. Finally, we fetched all past tests from the FlockLab database and rescheduled them on our test system. The exact setup of our tests and their results are shown below.

6.1. Scheduling Algorithm

The first task of our evaluation was the verification and performance measurement of our algorithm. We used scripts to generate artificial test configuration files in a certain way and scheduled them. Afterwards, we fetched the scheduled test from the database and used different checks to test if the scheduling was correct. We tested two specific cases: Parallel test on different observers and parallel test on the same observer.

6.1.1. Different Observers

To verify the scheduling for tests on different observers, we used the XML in listing 3 as a base. The *dbImageId* points to an existing target image in the database with Tmote architecture and the *obsIds* are added later.

Listing 3: Test Configuration - Different Observers

```xml
<generalConf>
    <name>FlockLab Test Scheduling</name>
    <description>Generated xml to test scheduling.</description>
    <scheduleAsap>
        <durationSecs>600</durationSecs>
    </scheduleAsap>
    <emailResults>yes</emailResults>
</generalConf>

<targetConf>
    <obsIds></obsIds>
    <voltage>3.3</voltage>
    <dbImageId>41</dbImageId>
</targetConf>

<serialConf>
    <obsIds></obsIds>
    <port>serial</port>
</serialConf>
```

We randomly separated all observers in three equally sized groups. For the first and second of these groups we modified the base configuration to use observer from the respective groups. We submitted each of these two test configuration files n-times in random order to our scheduler. Each time we measured the time needed by the scheduling algorithm to schedule a new test.

After all tests are scheduled, we executed the following automatic checks to verify the scheduling:

Observer Mapping Each of the observers in the first two groups are used in exactly n tests. None of the ones in the last group is used in any of the tests.

Resource Arrays Time Interval None of the *resource arrays* in the *resource table* overlap in time.

Parallel Tests The test are scheduled in parallel, if it's possible, not sequential.

To compare the two different scheduling algorithms and the results with and without rounding of the start time, we repeated this procedure for each combination of them. All of this combinations successfully passed all tests without any errors. We repeated this whole process multiple times to get more reliable results. The performance of the different scheduling algorithms is depicted in figure 8. We ran the test 50 times with a total of 200 scheduled tests per run (100 tests per observer group). The top four graphs show the time needed (y-axis) to schedule the n-th test (x-axis). The blue lines show the performance for each individual run whereas the red line represents the mean of all runs. As expected, the scheduling algorithm version two is clearly faster and the time increase to schedule an additional test is significantly smaller than in the version one.

The difference between rounded and not rounded start times can be seen in the top right graph. The blue lines can clearly be separated into two groups. The bottom group shows the same result as for the rounded start times while the top one is significantly slower. This happens when the first of the test on one of the observers groups is scheduled not at exactly at the same time as the first test of the other group. The reason for this is that the first test of the second group is scheduled a minimum of one second later than the first one of the first group. This causes all tests of the different groups to be shiftet one or more seconds and therefore the number of *resource arrays* in the *resource table* to double. Since the scheduling algorithms does an iteration for each *resource array* in the *resource table* that collides with the test, the number of iterations doubles as well. The same explanation holds for the graph for the scheduling algorithm version two without rounding, but without the noticeable split, because the gradient of the curve of the scheduling algorithm version two is much smaller.

The graphs at the bottom of the figure show the comparison of the different variants. The combination with the best performance is the scheduling algorithm version two with rounding. However, since the rounding can cause a user to wait up to a minute more for his test to start and the performance without rounding is only affected if multiple tests are submitted in the same minute on different observers, we recommend to using it without rounding. The use of version one is recommended if memory usage is an issue.

6.1.2. Same Observers

We used a similar approach to verify the scheduling algorithm for parallel tests on the same observer. The two different test configurations in listing 4 and 5 forced the scheduling algorithm to schedule the test in parallel on the same observer. Note the different

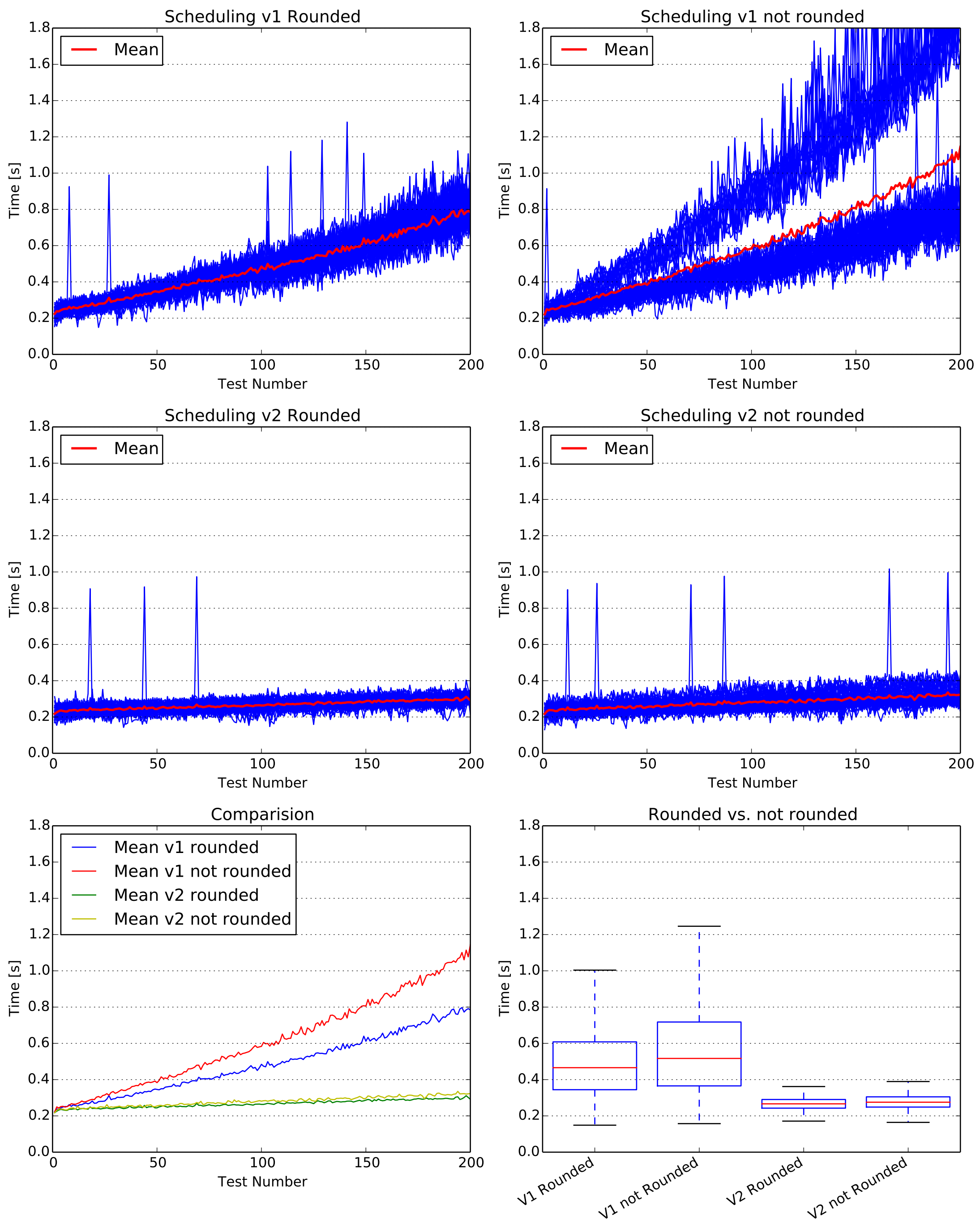


Figure 8: Performance of the scheduling algorithm for parallel test on different observers.

observer images in the test configuration. The first one is for Tmote and the second one for TinyNode targets, which use different communication frequencies. The *obsIds* are added later.

Listing 4: Test Configuration - Same Observers Test 1

```xml
<generalConf>
    <name>FlockLab Test Scheduling</name>
    <description>Generated xml to test scheduling.</description>
    <scheduleAsap>
        <durationSecs>600</durationSecs>
    </scheduleAsap>
    <emailResults>yes</emailResults>
</generalConf>

<targetConf>
    <obsIds></obsIds>
    <voltage>3.3</voltage>
    <dbImageId>41</dbImageId>
</targetConf>

<serialConf>
    <obsIds></obsIds>
    <port>usb</port>
</serialConf>
```

Listing 5: Test Configuration - Same Observers Test 2

```xml
<generalConf>
    <name>FlockLab Test Scheduling</name>
    <description>Generated xml to test scheduling.</description>
    <scheduleAsap>
        <durationSecs>600</durationSecs>
    </scheduleAsap>
    <emailResults>yes</emailResults>
</generalConf>

<targetConf>
    <obsIds></obsIds>
    <voltage>3.3</voltage>
    <dbImageId>42</dbImageId>
</targetConf>

<serialConf>
    <obsIds></obsIds>
    <port>usb</port>
</serialConf>
```

For this test we fetched all observers that have both, a Tmote and a TinyNode, connected to one of their slots and added them in the configuration. Then we alternately submitted the two configurations n-times each. This results in a scheduling that starts the first test on the observer and then, while the first test still runs, the second one. The third one can only be started after the second test finishes because of the multiplexer resource, which is needed for the setup, start, cleanup and stop phase and because of the frequency resources. So basically, the algorithm always scheduled the tests in packets of two, shifted by the setup plus start time.

The checks we executed afterwards were the same as in the last section, but with modified parameters for the calculations. The same holds for the performance differences for the different algorithm variants. Therefore, we only show the results of the algorithm version one with rounding in figure 9. The zig-zag-pattern in the graph draws attention. The reason for this is because for every other test, the algorithm has to update the *resource arrays* from the previous test, which takes more time than simply add new *resource arrays* at the end of the table.

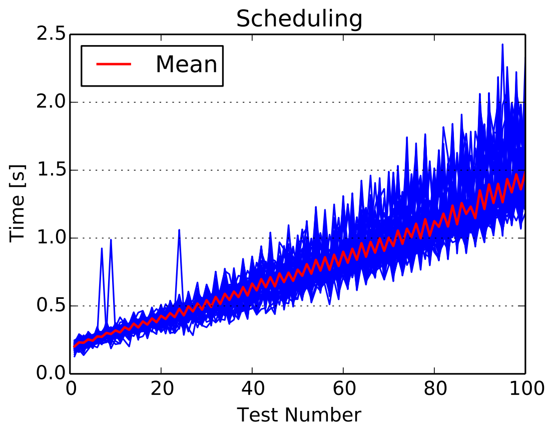


Figure 9: Performance of the scheduling algorithm for parallel test on the same observer.

6.2. FlockLab

After the verification of our scheduling algorithm, the next step was to execute parallel tests on real observers. We tested this by connecting our test system to the FlockLab network and scheduled tests, that were then executed on the real observers. Again, we tested two different cases: Parallel tests on different observers and parallel test on the same observer.

For the parallel tests on different observers, we used the FlockLab Tutorial 4: *GPIO Actuation* test configuration from the FlockLab website [2] and changed them to use different observer. We ran up to five test on different observers at the same time without errors and receiving the expected test results.

To test parallel tests on the same observer, we used the *Hello World* test configuration from the FlockLab Tutorial 2: *Getting Started and Serial Service* as the test which only uses the serial I/O service over USB. We extended the test duration in the configuration to be ten minutes so that we could schedule another test between the starting and the stopping of this test. For the test we ran in between, we used the CC430 architecture with an image that lets the LEDs on the observer blink. Both test were scheduled correctly and ran in parallel without errors and generating the expected test results.

6.3. Past Tests

We performed another experiment to examine the advantages of parallel test scheduling on FlockLab compared to sequential scheduling: We rescheduled the tests from the previous years and compared the total duration needed to run all tests for the different scheduling methods.

First, we fetched all past tests[1] from the FlockLab database, including the test configurations. Additionally, we made a dump of the *tbl_ serv_ targetimages* table and loaded it into our test system's database. This was necessary because it's the only way to get the target architectures used in a test. Then we changed the configurations from *absolute time* to *ASAP*, with the duration calculated form the start and end time of the test and sorted out the test whose target images were no longer available in the database. After this preparation, we submitted all tests of each year and scheduled them[2]. For many of the tests, the scheduling was not successfull. This was because on most of the observers, the target architectures on the different slots changed and were no longer available on the observers. After all tests were scheduled, we calculated the time difference between the start of the first to the end of the last test in the database and added the setup time for the first test and cleanup time of the last test to get the total time used to run all tests. To get the total time for sequential scheduling, we simply added the duration of each successfully scheduled test including the setup and cleanup time. The comparison of the total times for sequential and parallel scheduling are shown in table 2 and depicted in figure 10. The whole statistics for each year can be found in the appendix.

Table 2.: Result Rescheduling of Past Tests.

Year	Scheduled Tests	Time Sequential	Time Parallel	Reduction
2012	134	241140 s	231691 s	3.918 %
2013	2539	4621449 s	4442505 s	3.872 %
2014	5332	11896608 s	10185745 s	14.381 %
2015	7223	13585914 s	10977309 s	19.201 %
2016	1183	2424814 s	2173069 s	10.382 %

The tiny improvement in the years 2012 and 2013 is, because for this years only tests that use Tmote Sky targets were schedulable. That, and the fact that most of the observers where used in more than 90 % of the total duration made a better improvement impossible. In the years 2014 to 2016, the results were considerably better. For those years, the variety of the target architectures made it possible to schedule more tests in parallel. But still, about 80 % of all tests used Tmotes and many observers, which are limiting factors for the improvement.

Event though the time reduction is not outstanding, the users can still highly profit from the parallelization. For example when a test does not use all observers, it's possible to run an arbitrary test on other observers. Or when a long test that only uses the serial I/O service over USB runs on all observers, another user can still run his own test if he

[1] From 2012 up to the 15. March 2016.

[2] The XML configuration syntax has changed in March 2013. This causes our scheduling algorithm to not detect the multiplexer usage prior to the change. But since only Tmote architectures were schedulable in the years 2012 and 2013 this did not affect the results.

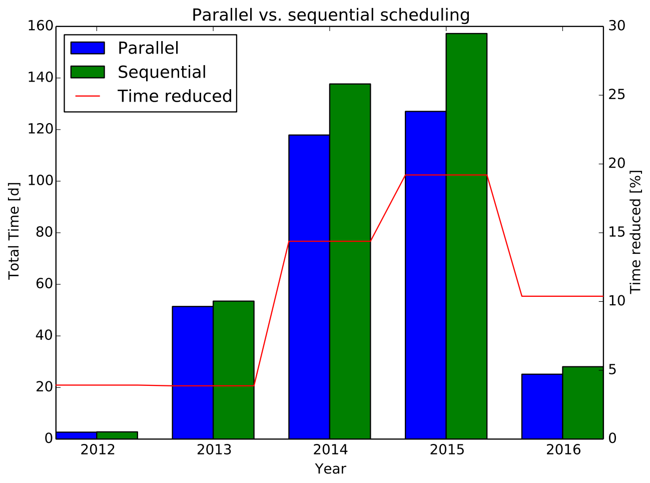


Figure 10: Total time reduction by parallel scheduling compared to sequential scheduling.

uses other target nodes. Therefore, we still expect the possibility to run tests in parallel to be highly beneficial.

7. Outlook

This project lays the foundation to modify FlockLab for running tests in parallel. However, there is still work to do before our solution can be implemented on FlockLab. In this section, we propose some tasks that should be accomplished before bringing parallel tests to the live system.

Missing Functions All changes and algorithms necessary for scheduling and running parallel tests on FlockLab are implemented and explained in this report, yet some important functions are missing. Probably the most important one is the option to remove a scheduled test or abort it. It is still possible to use the original function to delete a scheduled test, but it will only remove it from the tables *tbl_serv_tests* and *tbl_serv_map_test_observer_targetimages*. This will cause the test to not start. But as the resource usage in the *tbl_serv_test_resource* are not modified, the resources will still be blocked. Therefore, the function to remove tests has to be extended to clean up the *resource table* as well.

Extended Testing Although, we implemented scripts to automatically check our scheduling, we had only time to test a limited amount of different cases. We recommend further testing with a higher variety of different test configurations, before using our modifications on the live system.

Additional Configuration Options For some users it may falsify the results of their tests if other tests are running in parallel (e.g. a running test on neighboring observers with the same frequency). Therefore, we suggest adding configuration options in the test configuration XML to prevent parallel tests.

Modify Webinterface Without modifying the web interface as well, a user can not see which resources will be used in planned tests. Therefore, the user will assume that the whole testbed will be blocked during all planned test. For the user to take advantage of the parallelization of the testbed, it's necessary to modify the web interface as well.

8. Conclusion

The goal of our project was to modify FlockLab for running tests in parallel. To accomplish this, we first setup FlockLab in a test environment and then modify it to run parallel tests. We model the resource constraints by defining a set of exclusive resources and a mapping between test configurations and this resources. The scheduling algorithm provided is based on this model to correctly schedule new tests. We outline the modifications for FlockLab necessary to use the new scheduling algorithms and support parallel tests.

The evaluation includes performance measurements of the different scheduling algorithm variants and their verification. On the basis of this results, we show the advantages and disadvantages of the different variants. Furthermore, we show that our solution works by scheduling and executing tests on real observers of FlockLab. To compare parallel and sequential test scheduling, we rescheduled past tests from previous years and compared the time needed to run all tests for parallel and sequential scheduling. The evaluation shows, that the test throughput of FlockLab can be significantly higher with parallel running tests.

In conclusion, we provide a working extension for FlockLab to run tests in parallel. Yet, there is still room for improvement in terms of functionality.

Bibliography

[1] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Information Processing in Sensor Networks (IPSN), 2013 ACM/IEEE International Conference on*, pages 153–165, April 2013.

[2] ETHZ. `https://www.flocklab.ethz.ch`, 2016. [Online; accessed 22-March-2016].

[3] E. Ertin, A. Arora, R. Ramnath, M. Nesterenko, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, and H. Cao. Kansei: a testbed for sensing at scale. In *Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on*, pages 399–406, 2006.

[4] V. Handziski, A. Köpke, A. Willig, and A. Wolisz. Twist: A scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *Proceedings of the 2Nd International Workshop on Multi-hop Ad Hoc Networks: From Theory to Reality*, REALMAN '06, pages 63–70, New York, NY, USA, 2006. ACM.

[5] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: a wireless sensor network testbed. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 483–488, April 2005.

[6] X. Ju, H. Zhang, and D. Sakamuri. Neteye: a user-centered wireless sensor network testbed for high-fidelity, robust experimentation. *International Journal of Communication Systems*, 25(9):1213–1229, 2012.

[7] M. Dyer, J. Beutel, T. Kalt, P. Oehen, L. Thiele, K. Martin, and P. Blum. *Wireless Sensor Networks: 4th European Conference, EWSN 2007, Delft, The Netherlands, January 29-31, 2007. Proceedings*, chapter Deployment Support Network, pages 195–211. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[8] T. Dimitriou, J. Kolokouris, and N. Zarokostas. Sensenet: A wireless sensor network testbed. In *Proceedings of the 10th ACM Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, MSWiM '07, pages 143–150, New York, NY, USA, 2007. ACM.

[9] A. Achtzehn, E. Meshkova, J. Ansari, and P. Mahonen. Motemaster: A scalable sensor network testbed for rapid protocol performance evaluation. In *Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009. SECON Workshops '09. 6th Annual IEEE Communications Society Conference on*, pages 1–3, June 2009.

[10] M. Doddavenkatappa, M. Chan Choon, and A. L. Ananda. *Testbeds and Research Infrastructure. Development of Networks and Communities: 7th International ICST Conference,TridentCom 2011, Shanghai, China, April 17-19, 2011, Revised Selected*

Papers, chapter Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed, pages 302–316. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[11] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002.

[12] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuecel. Permadaq: A scientific instrument for precision sensing and data recovery in environmental extremes. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, pages 265–276, Washington, DC, USA, 2009. IEEE Computer Society.

[13] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 13–24, New York, NY, USA, 2004. ACM.

A. Appendix

A.1. Result Rescheduling of Past Tests

```
start
####################################################################################################
#                                                                                                  #
# 2012                                                                                             #
#                                                                                                  #
####################################################################################################
GENERAL:
---------------------------------------
| # Tests | scheduled | scheduled [%] |
|-------------------------------------|
|     157 |       134 |        85.350 |
---------------------------------------

----------------------------------------------------
| Architecture | Total | scheduled | scheduled [%] |
|--------------------------------------------------|
|        Tmote |   134 |       134 |       100.000 |
|        Mica2 |     1 |         0 |         0.000 |
|         Opal |     1 |         0 |         0.000 |
|     TinyNode |    21 |         0 |         0.000 |
|--------------------------------------------------|

-----------------------------------------------------------------------------
| Time Sequential [s]| Time Parallel [s]| Time Reduced [s]| Time Reduced [%]|
|---------------------------------------------------------------------------|
|              241140|            231691|             9449|            3.918|
-----------------------------------------------------------------------------

ARCHITECTURE STATISTICS:
---------------------------------------------------------------------------------------------------------------------
|   Architecture| # Tests| # Tests [%]| Duration Tests| # no Mux| # no Mux [%]| Duration no Mux| Duration no Mux [%]|
|-------------------------------------------------------------------------------------------------------------------|
|          Tmote|     134|     100.000|         241140|      134|      100.000|          241140|             100.000|
|-------------------------------------------------------------------------------------------------------------------|
|          TOTAL|     134|     100.000|         241140|      134|      100.000|          241140|             100.000|
---------------------------------------------------------------------------------------------------------------------

OBSERVER/ARCHITECTURE USAGE (TIME):
----------------------------------
|  Obs  |     Tmote ||     TOTAL |
|--------------------------------|
|  001  |    99.054 ||    99.054 |
|  002  |    98.588 ||    98.588 |
|  004  |    98.588 ||    98.588 |
|  006  |    96.413 ||    96.413 |
|  007  |    96.836 ||    96.836 |
|  008  |    98.873 ||    98.873 |
|  010  |    99.158 ||    99.158 |
|  011  |    99.339 ||    99.339 |
|  013  |    97.924 ||    97.924 |
|  014  |    97.924 ||    97.924 |
|  015  |    97.138 ||    97.138 |
|  016  |    97.785 ||    97.785 |
|  017  |    96.836 ||    96.836 |
|  018  |    95.515 ||    95.515 |
|  019  |    96.836 ||    96.836 |
|  020  |    96.836 ||    96.836 |
|  022  |    97.552 ||    97.552 |
|  023  |    96.836 ||    96.836 |
|  024  |    96.836 ||    96.836 |
|  025  |    99.158 ||    99.158 |
|  026  |    96.836 ||    96.836 |
|  027  |    95.515 ||    95.515 |
|  028  |    97.552 ||    97.552 |
|  031  |    96.335 ||    96.335 |
|  032  |    96.050 ||    96.050 |
|  033  |    96.335 ||    96.335 |
|  200  |    95.144 ||    95.144 |
|  201  |    95.144 ||    95.144 |
|  202  |    95.144 ||    95.144 |
|  204  |    95.144 ||    95.144 |
|--------------------------------|
| TOTAL |   104.078 ||          |
----------------------------------

OBSERVER/ARCHITECTURE USAGE (# TESTS):
----------------------------------
|  Obs  |     Tmote ||     TOTAL |
|--------------------------------|
|  001  |    85.075 ||    85.075 |
|  002  |    84.328 ||    84.328 |
|  004  |    84.328 ||    84.328 |
|  006  |    75.373 ||    75.373 |
|  007  |    73.134 ||    73.134 |
|  008  |    84.328 ||    84.328 |
|  010  |    85.075 ||    85.075 |
|  011  |    82.090 ||    82.090 |
|  013  |    76.119 ||    76.119 |
|  014  |    76.119 ||    76.119 |
|  015  |    78.358 ||    78.358 |
|  016  |    79.851 ||    79.851 |
|  017  |    73.134 ||    73.134 |
|  018  |    70.896 ||    70.896 |
|  019  |    73.134 ||    73.134 |
|  020  |    73.134 ||    73.134 |
|  022  |    76.866 ||    76.866 |
|  023  |    73.134 ||    73.134 |
|  024  |    73.134 ||    73.134 |
|  025  |    81.343 ||    81.343 |
|  026  |    73.134 ||    73.134 |
|  027  |    70.896 ||    70.896 |
|  028  |    76.866 ||    76.866 |
|  031  |    74.627 ||    74.627 |
|  032  |    73.881 ||    73.881 |
|  033  |    74.627 ||    74.627 |
|  200  |    70.149 ||    70.149 |
|  201  |    70.149 ||    70.149 |
|  202  |    70.149 ||    70.149 |
|  204  |    70.149 ||    70.149 |
|--------------------------------|
| TOTAL |   100.000 ||          |
----------------------------------

Maximal Number of Parallel tests: 3

####################################################################################################
#                                                                                                  #
# 2013                                                                                             #
#                                                                                                  #
####################################################################################################
GENERAL:
---------------------------------------
| # Tests | scheduled | scheduled [%] |
|-------------------------------------|
|    4095 |      2539 |        62.002 |
---------------------------------------

----------------------------------------------------
| Architecture | Total | scheduled | scheduled [%] |
|--------------------------------------------------|
|        Tmote |  2568 |      2539 |        98.871 |
|         Iris |   158 |         0 |         0.000 |
|     TinyNode |   520 |         0 |         0.000 |
|         Opal |   379 |         0 |         0.000 |
|      Wismote |   291 |         0 |         0.000 |
|        Mica2 |   150 |         0 |         0.000 |
|--------------------------------------------------|

-----------------------------------------------------------------------------
| Time Sequential [s]| Time Parallel [s]| Time Reduced [s]| Time Reduced [%]|
|---------------------------------------------------------------------------|
|             4621449|           4442505|           178944|            3.872|
-----------------------------------------------------------------------------

ARCHITECTURE STATISTICS:
------------------------------------------------------------------------------------------------------------------------
|    Architecture| # Tests| # Tests [%]| Duration Tests| # no Mux| # no Mux [%]| Duration no Mux| Duration no Mux [%]|
|----------------------------------------------------------------------------------------------------------------------|
|           Tmote|    2539|     100.000|        4621449|     1501|       59.118|         3095172|              66.974|
|----------------------------------------------------------------------------------------------------------------------|
|           TOTAL|    2539|     100.000|        4621449|     1501|       59.118|         3095172|              66.974|
------------------------------------------------------------------------------------------------------------------------

OBSERVER/ARCHITECTURE USAGE (TIME):
----------------------------------
|  Obs  |     Tmote ||     TOTAL |
|--------------------------------|
|  001  |    93.886 ||    93.886 |
|  002  |    95.353 ||    95.353 |
|  004  |    95.268 ||    95.268 |
|  006  |    93.334 ||    93.334 |
|  007  |    94.178 ||    94.178 |
|  008  |    93.715 ||    93.715 |
|  010  |    92.396 ||    92.396 |
|  011  |    93.639 ||    93.639 |
|  013  |    97.974 ||    97.974 |
|  014  |    95.725 ||    95.725 |
|  015  |    93.297 ||    93.297 |
|  016  |    93.792 ||    93.792 |
|  017  |    90.347 ||    90.347 |
|  018  |    94.053 ||    94.053 |
|  019  |    92.374 ||    92.374 |
|  020  |    96.138 ||    96.138 |
|  022  |    94.847 ||    94.847 |
|  023  |    93.707 ||    93.707 |
|  024  |    92.380 ||    92.380 |
|  025  |    99.245 ||    99.245 |
|  026  |    97.866 ||    97.866 |
|  027  |    92.766 ||    92.766 |
|  028  |    99.624 ||    99.624 |
|  031  |    96.821 ||    96.821 |
|  032  |    92.811 ||    92.811 |
|  033  |    95.855 ||    95.855 |
|  200  |    89.010 ||    89.010 |
|  201  |    43.735 ||    43.735 |
|  202  |    89.019 ||    89.019 |
|  204  |    89.093 ||    89.093 |
|---------------------------------|
| TOTAL |   104.028 ||           |
-----------------------------------

OBSERVER/ARCHITECTURE USAGE (# TESTS):
-----------------------------------
|  Obs  |     Tmote ||     TOTAL |
|---------------------------------|
|  001  |    87.554 ||    87.554 |
|  002  |    88.066 ||    88.066 |
|  004  |    87.475 ||    87.475 |
|  006  |    84.049 ||    84.049 |
|  007  |    84.128 ||    84.128 |
|  008  |    86.885 ||    86.885 |
|  010  |    84.876 ||    84.876 |
|  011  |    86.530 ||    86.530 |
|  013  |    87.121 ||    87.121 |
|  014  |    85.979 ||    85.979 |
|  015  |    85.979 ||    85.979 |
|  016  |    86.373 ||    86.373 |
|  017  |    83.064 ||    83.064 |
|  018  |    85.900 ||    85.900 |
|  019  |    84.049 ||    84.049 |
|  020  |    86.294 ||    86.294 |
|  022  |    86.058 ||    86.058 |
|  023  |    83.931 ||    83.931 |
|  024  |    83.537 ||    83.537 |
|  025  |    90.941 ||    90.941 |
|  026  |    87.003 ||    87.003 |
|  027  |    84.167 ||    84.167 |
|  028  |    90.547 ||    90.547 |
|  031  |    87.554 ||    87.554 |
|  032  |    84.403 ||    84.403 |
|  033  |    86.530 ||    86.530 |
|  200  |    81.843 ||    81.843 |
|  201  |    43.876 ||    43.876 |
|  202  |    81.883 ||    81.883 |
|  204  |    82.198 ||    82.198 |
|---------------------------------|
| TOTAL |   100.000 ||           |
-----------------------------------

Maximal Number of Parallel tests: 7

#########################################################################################################
#                                                                                                       #
# 2014                                                                                                  #
#                                                                                                       #
#########################################################################################################
GENERAL:
----------------------------------------
| # Tests | scheduled | scheduled [%] |
|--------------------------------------|
|    7107 |      5332 |        75.025 |
----------------------------------------

------------------------------------------------------
| Architecture | Total | scheduled | scheduled [%] |
|----------------------------------------------------|
|        Tmote |  4383 |      4376 |        99.840 |
|         Iris |    24 |         0 |         0.000 |
|     TinyNode |   199 |         0 |         0.000 |
|         Opal |   389 |         0 |         0.000 |
|        ACM2 |   825 |       823 |        99.758 |
|       CC430 |  1149 |       133 |        11.575 |
|       Mica2 |    42 |         0 |         0.000 |
|---------------------------------------------------|

-------------------------------------------------------------------------------
| Time Sequential [s]| Time Parallel [s]| Time Reduced [s]| Time Reduced [%]|
|-----------------------------------------------------------------------------|
|            11896608|          10185745|          1710863|           14.381|
-------------------------------------------------------------------------------

ARCHITECTURE STATISTICS:
-------------------------------------------------------------------------------------------------------------------------------------
|   Architecture| # Tests| # Tests [%]| Duration Tests| # no Mux| # no Mux [%]| Duration no Mux| Duration no Mux [%]|
|-----------------------------------------------------------------------------------------------------------------------------------|
|          Tmote|    4376|      82.071|       10775028|     2031|       46.412|         3867085|              35.889|
|           ACM2|     823|      15.435|        1018800|        0|        0.000|               0|               0.000|
|          CC430|     133|       2.494|         102780|        0|        0.000|               0|               0.000|
|-----------------------------------------------------------------------------------------------------------------------------------|
|          TOTAL|    5332|     100.000|       11896608|     2031|       38.091|         3867085|              32.506|
-------------------------------------------------------------------------------------------------------------------------------------

OBSERVER/ARCHITECTURE USAGE (TIME):
-----------------------------------------------------------
|  Obs  |     Tmote |      ACM2 |     CC430 ||     TOTAL |
|---------------------------------------------------------|
|  001  |    91.190 |     0.740 |     0.256 ||    92.185 |
|  002  |    89.422 |     8.127 |     0.594 ||    98.144 |
|  003  |    29.178 |     1.789 |     0.000 ||    30.966 |
|  004  |    90.330 |     8.064 |     0.529 ||    98.922 |
|  006  |    98.489 |     1.433 |     0.620 ||   100.543 |
|  007  |    93.647 |     0.000 |     0.022 ||    93.670 |
|  008  |    89.808 |     7.832 |     0.279 ||    97.918 |
|  010  |    94.269 |     0.000 |     0.014 ||    94.283 |
|  011  |    93.292 |     0.000 |     0.000 ||    93.292 |
|  013  |    91.719 |     0.000 |     0.012 ||    91.731 |
|  014  |    94.954 |     0.000 |     0.008 ||    94.962 |
|  015  |    89.528 |     1.147 |     0.194 ||    90.869 |
|  016  |    96.870 |     0.000 |     0.485 ||    97.356 |
|  017  |    83.803 |     0.000 |     0.008 ||    83.811 |
|  018  |    93.575 |     0.000 |     0.091 ||    93.666 |
|  019  |    94.230 |     0.000 |     0.008 ||    94.238 |
|  020  |    97.000 |     0.000 |     0.000 ||    97.000 |
|  022  |    96.997 |     0.000 |     0.012 ||    97.010 |
|  023  |    94.429 |     0.000 |     0.008 ||    94.437 |
|  024  |    94.536 |     0.000 |     0.183 ||    94.720 |
|  025  |    96.861 |     0.000 |     0.000 ||    96.861 |
|  026  |    97.182 |     0.000 |     0.000 ||    97.182 |
|  027  |    93.599 |     0.000 |     0.058 ||    93.658 |
|  028  |    98.819 |     0.363 |     0.000 ||    99.181 |
|  029  |     0.435 |     0.000 |     0.000 ||     0.435 |
|  031  |    94.868 |     0.014 |     0.000 ||    94.882 |
|  032  |    97.247 |     0.410 |     0.000 ||    97.657 |
|  033  |    75.439 |     2.869 |     0.169 ||    78.477 |
|  200  |    90.424 |     0.000 |     0.104 ||    90.528 |
|  201  |    85.529 |     0.000 |     0.019 ||    85.548 |
|  202  |    59.917 |     0.000 |     0.058 ||    59.975 |
|  204  |    89.296 |     0.000 |     0.234 ||    89.531 |
|---------------------------------------------------------|
| TOTAL |   105.785 |    10.002 |     1.009 ||           |
-----------------------------------------------------------

OBSERVER/ARCHITECTURE USAGE (# TESTS):
-----------------------------------------------------------
|  Obs  |     Tmote |      ACM2 |     CC430 ||     TOTAL |
|---------------------------------------------------------|
|  001  |    62.228 |     2.926 |     0.638 ||    65.791 |
|  002  |    59.677 |     9.565 |     1.369 ||    70.611 |
|  003  |    25.525 |     5.814 |     0.000 ||    31.339 |
|  004  |    60.709 |     9.377 |     1.238 ||    71.324 |
|  006  |    65.548 |     4.670 |     1.519 ||    71.737 |
|  007  |    53.957 |     0.000 |     0.094 ||    54.051 |
|  008  |    60.296 |     9.696 |     0.788 ||    70.780 |
|  010  |    55.608 |     0.000 |     0.056 ||    55.664 |
|  011  |    53.788 |     0.000 |     0.000 ||    53.788 |
|  013  |    56.077 |     0.000 |     0.056 ||    56.133 |
|  014  |    55.833 |     0.000 |     0.038 ||    55.870 |
|  015  |    62.697 |     3.507 |     0.431 ||    66.635 |
|  016  |    64.141 |     0.000 |     1.088 ||    65.229 |
|  017  |    55.701 |     0.000 |     0.038 ||    55.739 |
|  018  |    54.201 |     0.000 |     0.188 ||    54.389 |
|  019  |    56.133 |     0.000 |     0.038 ||    56.170 |
|  020  |    60.578 |     0.000 |     0.000 ||    60.578 |
|  022  |    64.291 |     0.000 |     0.019 ||    64.310 |
|  023  |    57.052 |     0.000 |     0.038 ||    57.089 |
|  024  |    56.377 |     0.000 |     0.338 ||    56.714 |
|  025  |    59.771 |     0.000 |     0.000 ||    59.771 |
|  026  |    59.959 |     0.000 |     0.000 ||    59.959 |
|  027  |    54.295 |     0.000 |     0.131 ||    54.426 |
|  028  |    65.585 |     1.069 |     0.000 ||    66.654 |
|  029  |     1.725 |     0.000 |     0.000 ||     1.725 |
|  031  |    64.010 |     0.056 |     0.000 ||    64.066 |
|  032  |    66.279 |     1.257 |     0.000 ||    67.536 |
|  033  |    57.314 |     9.265 |     0.319 ||    66.898 |
|  200  |    50.581 |     0.000 |     0.244 ||    50.825 |
|  201  |    44.805 |     0.000 |     0.075 ||    44.880 |
|  202  |    41.335 |     0.000 |     0.188 ||    41.523 |
|  204  |    50.825 |     0.000 |     0.431 ||    51.257 |
|---------------------------------------------------------|
| TOTAL |    82.071 |    15.435 |     2.494 ||           |
-----------------------------------------------------------

Maximal Number of Parallel tests: 10

###########################################################################################################
#                                                                                                         #
# 2015                                                                                                    #
#                                                                                                         #
###########################################################################################################
GENERAL:
----------------------------------------
| # Tests | scheduled | scheduled [%] |
|--------------------------------------|
|    9006 |      7223 |        80.202 |
----------------------------------------

-----------------------------------------------------
| Architecture | Total | scheduled | scheduled [%] |
|---------------------------------------------------|
|        Tmote |  5500 |      5500 |       100.000 |
|         None |     4 |         0 |         0.000 |
|     TinyNode |   182 |       150 |        82.418 |
|     OpenMote |   335 |       331 |        98.806 |
|         Opal |   366 |         0 |         0.000 |
|         ACM2 |  1651 |      1286 |        77.892 |
|        CC430 |  1023 |        26 |         2.542 |
|      Wismote |     8 |         8 |       100.000 |
|---------------------------------------------------|

-----------------------------------------------------------------------------
| Time Sequential [s]| Time Parallel [s]| Time Reduced [s]| Time Reduced [%]|
|---------------------------------------------------------------------------|
|            13585914|          10977309|          2608605|           19.201|
-----------------------------------------------------------------------------

ARCHITECTURE STATISTICS:
-------------------------------------------------------------------------------------------------------------------------------
|   Architecture| # Tests| # Tests [%]| Duration Tests| # no Mux| # no Mux [%]| Duration no Mux| Duration no Mux [%]|
|-----------------------------------------------------------------------------------------------------------------------------|
|          Tmote|    5500|      76.146|       11460394|     3071|       55.836|         9091419|              79.329|
|       TinyNode|     150|       2.077|         129000|        0|        0.000|               0|               0.000|
|        Wismote|       8|       0.111|          10080|        8|      100.000|           10080|             100.000|
|       OpenMote|     331|       4.583|         389060|        7|        2.115|            3660|               0.941|
|           ACM2|    1286|      17.804|        1633620|        0|        0.000|               0|               0.000|
|          CC430|      26|       0.360|          17640|        0|        0.000|               0|               0.000|
|-----------------------------------------------------------------------------------------------------------------------------|
|          TOTAL|    7223|     100.000|       13639794|     3086|       42.725|         9105159|              66.754|
-------------------------------------------------------------------------------------------------------------------------------

OBSERVER/ARCHITECTURE USAGE (TIME):
-----------------------------------------------------------------------------------------------
| Obs  |     Tmote |  TinyNode |   Wismote |  OpenMote |      ACM2 |     CC430 ||     TOTAL |
|---------------------------------------------------------------------------------------------|
|  001  |    91.082 |     0.000 |     0.000 |     0.000 |    11.159 |     0.130 ||   102.371 |
|  002  |    90.117 |     0.000 |     0.000 |     0.000 |    11.220 |     0.152 ||   101.490 |
|  003  |    92.278 |     0.000 |     0.000 |     0.885 |    12.598 |     0.000 ||   105.760 |
|  004  |    93.072 |     0.000 |     0.000 |     0.000 |    11.238 |     0.146 ||   104.456 |
|  006  |    80.936 |     0.000 |     0.000 |     0.759 |    12.147 |     0.000 ||    93.842 |
|  007  |    83.648 |     1.175 |     0.000 |     0.000 |     0.000 |     0.009 ||    84.832 |
|  008  |    95.563 |     0.000 |     0.000 |     0.911 |    11.725 |     0.146 ||   108.346 |
|  010  |    93.673 |     1.175 |     0.000 |     0.000 |     0.000 |     0.000 ||    94.848 |
|  011  |    83.041 |     1.167 |     0.000 |     0.000 |     0.000 |     0.000 ||    84.208 |
|  013  |    89.627 |     1.175 |     0.000 |     0.000 |     0.000 |     0.000 ||    90.802 |
|  014  |    87.723 |     1.175 |     0.000 |     0.000 |     0.000 |     0.009 ||    88.907 |
|  015  |    94.791 |     0.000 |     0.000 |     0.713 |    11.024 |     0.048 ||   106.577 |
|  016  |    94.962 |     0.000 |     0.000 |     2.573 |    11.319 |     0.000 ||   108.854 |
|  017  |    87.110 |     1.112 |     0.000 |     0.000 |     0.000 |     0.009 ||    88.231 |
|  018  |    91.177 |     0.000 |     0.000 |     2.974 |    10.773 |     0.000 ||   104.925 |
|  019  |    88.777 |     1.167 |     0.000 |     0.000 |     0.000 |     0.000 ||    89.945 |
|  020  |    91.957 |     0.000 |     0.092 |     0.000 |     0.000 |     0.000 ||    92.049 |
|  022  |    96.126 |     0.000 |     0.000 |     2.821 |     0.000 |     0.000 ||    98.947 |
|  023  |    89.872 |     0.000 |     0.000 |     3.341 |     0.000 |     0.000 ||    93.214 |
|  024  |    86.611 |     0.000 |     0.000 |     3.229 |     0.000 |     0.000 ||    89.841 |
|  025  |    87.452 |     1.175 |     0.000 |     0.000 |     0.000 |     0.009 ||    88.636 |
|  026  |    90.450 |     1.175 |     0.000 |     0.000 |     0.000 |     0.000 ||    91.626 |
|  027  |    90.415 |     0.000 |     0.057 |     0.000 |    10.743 |     0.000 ||   101.215 |
|  028  |    96.834 |     0.000 |     0.000 |     0.000 |    11.076 |     0.000 ||   107.910 |
|  029  |     0.361 |     0.000 |     0.000 |     0.000 |     0.000 |     0.000 ||     0.361 |
| 031  |    95.080 |    0.000 |    0.000 |    0.739 |    1.213 |    0.000 ||   97.032 |
| 032  |    95.885 |    0.000 |    0.092 |    0.000 |   11.072 |    0.000 ||  107.049 |
| 033  |    96.013 |    0.000 |    0.092 |    0.000 |   12.674 |    0.048 ||  108.826 |
| 200  |    18.801 |    1.159 |    0.000 |    0.000 |    0.000 |    0.000 ||   19.961 |
| 201  |    20.352 |    1.152 |    0.000 |    0.000 |    0.000 |    0.000 ||   21.504 |
| 202  |    19.891 |    1.136 |    0.000 |    0.000 |    0.000 |    0.000 ||   21.027 |
| 204  |    20.592 |    1.159 |    0.000 |    0.000 |    0.000 |    0.000 ||   21.752 |
|---------------------------------------------------------------------------------------|
| TOTAL |  104.401 |    1.175 |    0.092 |    3.544 |   14.882 |    0.161 ||          |
-----------------------------------------------------------------------------------------

OBSERVER/ARCHITECTURE USAGE (# TESTS):
-----------------------------------------------------------------------------------------
| Obs  |     Tmote | TinyNode |  Wismote | OpenMote |     ACM2 |    CC430 ||    TOTAL |
|---------------------------------------------------------------------------------------|
| 001  |    64.239 |    0.000 |    0.000 |    0.000 |    9.248 |    0.291 ||   73.778 |
| 002  |    64.018 |    0.000 |    0.000 |    0.000 |    9.442 |    0.346 ||   73.806 |
| 003  |    62.966 |    0.000 |    0.000 |    0.914 |   12.723 |    0.000 ||   76.603 |
| 004  |    66.247 |    0.000 |    0.000 |    0.000 |    9.511 |    0.332 ||   76.090 |
| 006  |    65.042 |    0.000 |    0.000 |    0.858 |   11.283 |    0.000 ||   77.184 |
| 007  |    53.067 |    2.077 |    0.000 |    0.000 |    0.000 |    0.014 ||   55.157 |
| 008  |    68.808 |    0.000 |    0.000 |    0.955 |   11.117 |    0.332 ||   81.213 |
| 010  |    62.301 |    2.077 |    0.000 |    0.000 |    0.000 |    0.000 ||   64.378 |
| 011  |    52.070 |    2.063 |    0.000 |    0.000 |    0.000 |    0.000 ||   54.133 |
| 013  |    58.604 |    2.077 |    0.000 |    0.000 |    0.000 |    0.000 ||   60.681 |
| 014  |    57.095 |    2.077 |    0.000 |    0.000 |    0.000 |    0.014 ||   59.186 |
| 015  |    67.368 |    0.000 |    0.000 |    0.858 |    8.902 |    0.111 ||   77.239 |
| 016  |    66.496 |    0.000 |    0.000 |    3.503 |    8.888 |    0.000 ||   78.887 |
| 017  |    57.954 |    1.966 |    0.000 |    0.000 |    0.000 |    0.014 ||   59.934 |
| 018  |    58.964 |    0.000 |    0.000 |    3.890 |    7.407 |    0.000 ||   70.262 |
| 019  |    56.818 |    2.063 |    0.000 |    0.000 |    0.000 |    0.000 ||   58.881 |
| 020  |    60.363 |    0.000 |    0.111 |    0.000 |    0.000 |    0.000 ||   60.473 |
| 022  |    68.213 |    0.000 |    0.000 |    3.544 |    0.000 |    0.000 ||   71.757 |
| 023  |    57.303 |    0.000 |    0.000 |    4.140 |    0.000 |    0.000 ||   61.443 |
| 024  |    55.697 |    0.000 |    0.000 |    4.043 |    0.000 |    0.000 ||   59.740 |
| 025  |    57.566 |    2.077 |    0.000 |    0.000 |    0.000 |    0.014 ||   59.657 |
| 026  |    60.280 |    2.077 |    0.000 |    0.000 |    0.000 |    0.000 ||   62.356 |
| 027  |    59.241 |    0.000 |    0.069 |    0.000 |    7.296 |    0.000 ||   66.607 |
| 028  |    69.708 |    0.000 |    0.000 |    0.000 |    8.127 |    0.000 ||   77.835 |
| 029  |     0.761 |    0.000 |    0.000 |    0.000 |    0.000 |    0.000 ||    0.761 |
| 031  |    67.894 |    0.000 |    0.000 |    0.789 |    2.810 |    0.000 ||   71.494 |
| 032  |    69.016 |    0.000 |    0.111 |    0.000 |    8.168 |    0.000 ||   77.295 |
| 033  |    69.320 |    0.000 |    0.111 |    0.000 |   12.959 |    0.111 ||   82.500 |
| 200  |    21.612 |    2.049 |    0.000 |    0.000 |    0.000 |    0.000 ||   23.661 |
| 201  |    23.619 |    2.035 |    0.000 |    0.000 |    0.000 |    0.000 ||   25.654 |
| 202  |    23.425 |    2.007 |    0.000 |    0.000 |    0.000 |    0.000 ||   25.433 |
| 204  |    24.284 |    2.049 |    0.000 |    0.000 |    0.000 |    0.000 ||   26.333 |
|---------------------------------------------------------------------------------------|
| TOTAL |   76.146 |    2.077 |    0.111 |    4.583 |   17.804 |    0.360 ||          |
-----------------------------------------------------------------------------------------

Maximal Number of Parallel tests: 8

###############################################################################################
#                                                                                             #
# 2016                                                                                        #
#                                                                                             #
###############################################################################################
GENERAL:
--------------------------------------
| # Tests | scheduled | scheduled [%] |
|-------------------------------------|
|    1333 |      1183 |        88.747 |
--------------------------------------

--------------------------------------------------
| Architecture | Total | scheduled | scheduled [%] |
|------------------------------------------------|
|        Tmote |   946 |       946 |       100.000 |
|        CC430 |   185 |       120 |        64.865 |
|         Opal |    74 |        70 |        94.595 |
|     TinyNode |    49 |        47 |        95.918 |
|          dpp |    79 |         0 |         0.000 |
|------------------------------------------------|

---------------------------------------------------------------------------
| Time Sequential [s]| Time Parallel [s]| Time Reduced [s]| Time Reduced [%]|
|-------------------------------------------------------------------------|
|             2424814|           2173069|           251745|           10.382|
---------------------------------------------------------------------------

ARCHITECTURE STATISTICS:
------------------------------------------------------------------------------------------------------------------
|    Architecture| # Tests| # Tests [%]| Duration Tests| # no Mux| # no Mux [%]| Duration no Mux| Duration no Mux [%]|
|----------------------------------------------------------------------------------------------------------------|
|           Tmote|     946|      79.966|        2187300|      372|       39.323|         1465510|              67.001|
|           CC430|     120|      10.144|         136494|        0|        0.000|               0|               0.000|
|            Opal|      70|       5.917|          60200|        0|        0.000|               0|               0.000|
|        TinyNode|      47|       3.973|          40820|        8|       17.021|            7080|              17.344|
|-----------------------------------------------------------------------------------------------------------------|
|          TOTAL|    1183|     100.000|        2424814|       380|       32.122|          1472590|             60.730|
-------------------------------------------------------------------------------------------------------------------

OBSERVER/ARCHITECTURE USAGE (TIME):
-------------------------------------------------------------------
| Obs  |     Tmote |     CC430 |      Opal |  TinyNode ||     TOTAL |
|-----------------------------------------------------------------|
| 001  |    95.275 |     0.584 |     0.000 |     0.000 ||    95.859 |
| 002  |    95.409 |     0.563 |     2.770 |     0.000 ||    98.743 |
| 003  |    93.784 |     0.000 |     0.000 |     0.000 ||    93.784 |
| 004  |    93.727 |     0.584 |     2.770 |     0.000 ||    97.082 |
| 006  |    70.961 |     1.327 |     0.000 |     0.000 ||    72.287 |
| 007  |    90.533 |     0.456 |     2.770 |     1.878 ||    95.638 |
| 008  |    96.041 |     0.563 |     0.000 |     0.000 ||    96.604 |
| 010  |    94.664 |     0.456 |     0.000 |     1.878 ||    96.998 |
| 011  |    94.650 |     0.456 |     2.770 |     1.878 ||    99.755 |
| 013  |    94.637 |     0.456 |     2.770 |     1.878 ||    99.741 |
| 014  |    94.379 |     0.456 |     2.770 |     1.878 ||    99.483 |
| 015  |    95.833 |     0.563 |     0.000 |     0.000 ||    96.396 |
| 016  |    96.840 |     3.037 |     0.000 |     0.000 ||    99.877 |
| 017  |    89.778 |     0.425 |     2.691 |     1.839 ||    94.733 |
| 018  |    95.284 |     5.489 |     0.000 |     0.000 ||   100.773 |
| 019  |    93.503 |     0.456 |     2.770 |     1.878 ||    98.608 |
| 020  |    94.169 |     0.456 |     2.770 |     0.000 ||    97.395 |
| 022  |    94.815 |     3.661 |     0.000 |     0.000 ||    98.475 |
| 023  |    94.920 |     1.963 |     2.770 |     0.000 ||    99.653 |
| 024  |    94.787 |     0.584 |     2.770 |     0.000 ||    98.142 |
| 025  |    92.067 |     0.456 |     2.770 |     1.878 ||    97.172 |
| 026  |    94.428 |     0.456 |     0.000 |     1.878 ||    96.762 |
| 027  |    95.284 |     4.134 |     0.000 |     0.000 ||    99.418 |
| 028  |    96.085 |     0.000 |     2.770 |     0.000 ||    98.855 |
| 029  |     1.281 |     0.000 |     0.000 |     0.000 ||     1.281 |
| 031  |    92.093 |     0.563 |     0.000 |     0.000 ||    92.657 |
| 032  |    94.780 |     0.563 |     0.000 |     0.000 ||    95.343 |
| 033  |    97.010 |     0.563 |     0.000 |     0.000 ||    97.573 |
| 200  |    18.168 |     4.655 |     2.691 |     1.839 ||    27.354 |
| 201  |    18.296 |     0.563 |     2.770 |     1.878 ||    23.508 |
| 202  |    15.992 |     2.347 |     2.137 |     1.562 ||    22.037 |
| 204  |    17.969 |     2.714 |     2.770 |     1.878 ||    25.332 |
|-----------------------------------------------------------------|
| TOTAL |  100.655 |     6.281 |     2.770 |     1.878 ||           |
-------------------------------------------------------------------

OBSERVER/ARCHITECTURE USAGE (# TESTS):
-------------------------------------------------------------------
| Obs  |     Tmote |     CC430 |      Opal |  TinyNode ||     TOTAL |
|-----------------------------------------------------------------|
| 001  |    70.752 |     1.606 |     0.000 |     0.000 ||    72.358 |
| 002  |    71.260 |     1.522 |     5.917 |     0.000 ||    78.698 |
| 003  |    65.004 |     0.000 |     0.000 |     0.000 ||    65.004 |
| 004  |    65.173 |     1.606 |     5.917 |     0.000 ||    72.697 |
| 006  |    56.551 |     2.959 |     0.000 |     0.000 ||    59.510 |
| 007  |    60.101 |     1.268 |     5.917 |     3.973 ||    71.260 |
| 008  |    72.189 |     1.522 |     0.000 |     0.000 ||    73.711 |
| 010  |    70.161 |     1.268 |     0.000 |     3.973 ||    75.402 |
| 011  |    70.161 |     1.268 |     5.917 |     3.973 ||    81.319 |
| 013  |    70.161 |     1.268 |     5.917 |     3.973 ||    81.319 |
| 014  |    69.484 |     1.268 |     5.917 |     3.973 ||    80.642 |
| 015  |    72.020 |     1.522 |     0.000 |     0.000 ||    73.542 |
| 016  |    71.344 |     4.818 |     0.000 |     0.000 ||    76.162 |
| 017  |    58.242 |     1.183 |     5.748 |     3.888 ||    69.062 |
| 018  |    70.414 |     8.369 |     0.000 |     0.000 ||    78.783 |
| 019  |    67.456 |     1.268 |     5.917 |     3.973 ||    78.614 |
| 020  |    68.808 |     1.268 |     5.917 |     0.000 ||    75.993 |
| 022  |    69.992 |     5.917 |     0.000 |     0.000 ||    75.909 |
| 023  |    70.245 |     3.635 |     5.917 |     0.000 ||    79.797 |
| 024  |    70.076 |     1.606 |     5.917 |     0.000 ||    77.599 |
| 025  |    68.132 |     1.268 |     5.917 |     3.973 ||    79.290 |
| 026  |    69.231 |     1.268 |     0.000 |     3.973 ||    74.472 |
| 027  |    70.414 |     7.270 |     0.000 |     0.000 ||    77.684 |
| 028  |    70.837 |     0.000 |     5.917 |     0.000 ||    76.754 |
| 029  |     4.142 |     0.000 |     0.000 |     0.000 ||     4.142 |
| 031  |    61.200 |     1.522 |     0.000 |     0.000 ||    62.722 |
| 032  |    69.484 |     1.522 |     0.000 |     0.000 ||    71.006 |
| 033  |    72.105 |     1.522 |     0.000 |     0.000 ||    73.626 |
| 200  |    20.626 |     8.030 |     5.748 |     3.888 ||    38.292 |
| 201  |    20.879 |     1.522 |     5.917 |     3.973 ||    32.291 |
| 202  |    18.090 |     3.973 |     4.565 |     3.297 ||    29.924 |
| 204  |    21.640 |     4.903 |     5.917 |     3.973 ||    36.433 |
|-----------------------------------------------------------------|
| TOTAL |   79.966 |    10.144 |     5.917 |     3.973 ||           |
-------------------------------------------------------------------

Maximal Number of Parallel tests: 5

End
```