



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

1.	Intro 1.1.	o ductio Motiva	on ation and Contributions	1 . 1
2.	Rela	ted Wo	ork	3
3.	Bac l 3.1.	kgroun Wirele	d ess Sensor Networks	4
4.	Desi	gn		5
	4.1.	FlockI	Lab	. 5
		4.1.1.	Overview	. 5
		4.1.2.	Webserver	. 5
		4.1.3.	Database	. 6
		4.1.4.	Test Management Server	. 7
		4.1.5.	Observers	. 7
		4.1.6.	Test Cycle	. 8
	4.2.	Test S	$\tilde{\operatorname{System}}$. 9
	4.3.	Conce	pt	. 10
		4.3.1.	Resource Constraints	. 10
		4.3.2.	Performance Considerations	. 12
	4.4.	Schedu	uling Algorithm	. 13
		4.4.1.	Absolute Time Mode	. 13
		4.4.2.	ASAP Mode	13
5.	Impl	ementa	ation	15
	5.1.	Datab	pase	. 15
	5.2.	Test N	Management Server and Observers	. 15
	5.3.	Webse	erver	. 16
	5.4.	Schedu	uling Algorithm	. 16
		5.4.1.	General	. 16
		5.4.2.	Get Resource Usage	. 17
		5.4.3.	Merge Resource Arrays	. 18
		5.4.4.	Scheduling	. 19
		5.4.5.	Add Test to Database	. 21
6.	Eval	uation		23
	6.1.	Schedu	uling Algorithm	. 23
		6.1.1.	Different Observers	. 23
		6.1.2.	Same Observers	. 24
	6.2.	FlockI	Lab	. 27

	6.3. Past Tests	28
7.	Outlook	30
8.	Conclusion	31
Α.	Appendix A.1. Result Rescheduling of Past Tests A.2. Declaration of Originality	34 34 41

List of Figures

1.	FlockLab - Number of Tests 1
2.	Test Cycle
3.	Model of an Observer
4.	Scheduling Algorithm - Absolute Time
5.	Scheduling Algorithm - ASAP 14
6.	Resource Usage
7.	Example Merge Resource Array 19
8.	Scheduling Performance - Different Observers
9.	Scheduling Performance - Same Observer
10.	Result Rescheduling of Past Tests

List of Tables

1.	Example of a Resource Usage Table.	12
2.	Result Rescheduling of Past Tests.	28

Listings

1.	Scheduling version one	20
2.	Scheduling version two	21
n		റെ
J.	lest Configuration - Different Observers	23
4.	Test Configuration - Same Observers Test 1	26
5.	Test Configuration - Same Observers Test 2	26

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]. Indriva [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 concept 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 file¹.
- **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 flock-lab_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 *flock-lab_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 repository². 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

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 \Rightarrow slots numbers
- architecture \Rightarrow frequencies
- used services + start/stop + setup/cleanup \Rightarrow 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 table³ 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*.

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

Table 1.: Example of a resource usage table.

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.



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



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 Flock-Lab 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 *flock-lab_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 lds 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)
Toot Start	Setup	used	free	used	used
	Start	used	used	used	used
	Run	used	used	free	used
Test Find -	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.



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.

т		-1	A 1	1 1.	•	
L	isting	1:	Sche	duling	version	one

```
def schedule([List]resArrays):
1
       \# repeat until the test is scheduled and manually exited
2
       while not scheduled:
3
            isSchedulable = True
4
           # Fetch resource arrays from db which overlap in time
5
            dbResArray = fetch_overlapping_resArrays_from_db()
6
            for new in resArrays:
7
                for old in dbResArray:
8
                    if timeIntervalsOverlap(new, old):
9
                        \# check if they use different resources (bitwise or)
10
                         if any ([x&y for (x,y) in zip(new[2:], old[2:])]):
11
                             isSchedulable = False
                             timeShift = old[1] + 1 - new[0]
13
                             break
14
                if not isSchedulable:
16
17
                    break
18
            if not isSchedulable:
1\,9
                if ASAP:
20
                    resArrays = increaseTimes(resArray, timeShift)
21
                    continue
22
                else:
23
                    return False
24
            else:
25
                return True
26
```

Listing 2: Scheduling version two

```
def schedule([List]resArrays):
1
       \# first get ALL resource arrays from the db
2
       dbResArray = fetch_resArrays_from_db()
3
        while not scheduled:
4
            timeShift = 0
5
6
            isSchedulable = True
7
           # loop through all old resource arrays
8
            for old in dbResArray:
                # compare them with the first one from the new test (+timeShift
9
                if timeIntervalsOverlap(resArrays[0] + timeShift, old):
10
                     if any ([ x&y for (x,y) in zip(new[2:],old[2:])]):
                        \# if the collide and mode is not ASAP -> return
12
                        if not ASAP:
                             return False
14
15
                        \# Else, add the necessary timeshift and continue
16
17
                         else:
                             timeShift += old [1] + 1 - new [0]
18
                             continue
19
20
                \# if the first resource array didn't collide apply timeShift to
21
                     all intervals and check the others
                resArrays = increaseTimes(resArray, timeShift)
22
                isSchedulable, timeShift = checkOtherArrays(resArrays, old)
24
25
                if not isSchedulable:
                    \# continue with the new timeshift if the other did collide
26
                    continue
27
                else:
28
                    return True
29
```

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

```
<generalConf>
        <name>FlockLab Test Scheduling</name>
2
        < description> Generated xml to test scheduling.<\!\!/ description>
3
        <scheduleAsap>
 4
            < durationSecs>600</durationSecs>
5
        </ scheduleAsap>
6
        < emailResults>yes </emailResults>
7
   </generalConf>
8
9
10
   <targetConf>
        < obsIds > < /obsIds >
11
        <voltage>3.3</voltage>
12
        <dbImageId>41</dbImageId>
13
   </targetConf>
14
15
   <serialConf>
16
        < obsIds > < /obsIds >
17
        < port > serial < / port >
18
   </ serialConf>
19
```

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

1	<generalconf></generalconf>
2	< name $>$ FlockLab Test Scheduling $< /$ name $>$
3	$< {\tt description} > {\tt Generated \ xml}$ to test ${\tt scheduling.} < / {\tt description} >$
4	<scheduleasap></scheduleasap>
5	$<\!{ m d}{ m urationSecs}\!>\!600<\!/{ m d}{ m urationSecs}\!>$
6	$$
7	< emailResults> yes
8	
9	
0	< argetConf>
1	< obsIds > / obsIds >
2	< m voltage > 3.3 < / m voltage >
3	< dbImageId > 41 < / dbImageId >
4	$$
5	
6	< serialConf>
.7	< obsIds > / obsIds >
8	< m port> usb < /port>
9	

Listing 5: Test Configuration - Same Observers Test 2

```
<generalConf>
1
        <name>FlockLab Test Scheduling</name>
2
        < description > Generated xml to test scheduling.</ description >
3
4
        <scheduleAsap>
             < durationSecs>600</durationSecs>
5
6
        </scheduleAsap>
7
        < emailResults>yes </emailResults>
   </ generalConf>
8
9
   <targetConf>
10
        <\!\!o\,b\,sI\,d\,s\!\!>\!\!<\!\!/\,o\,b\,sI\,d\,s\!\!>
11
        <voltage>3.3</voltage>
12
        <dbImageId>42</dbImageId>
13
   </targetConf>
14
   <serialConf>
16
        < obsIds > < /obsIds >
17
        < port > usb < / port >
18
    </ serialConf>
19
```

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 tests¹ 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 them². 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.

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	$11896608 \ { m s}$	$10185745 { m \ s}$	14.381~%
2015	7223	$13585914 { m \ s}$	$10977309 { m \ s}$	19.201~%
2016	1183	2424814 s	$2173069 { m \ s}$	10.382~%

Table 2.: Result Rescheduling of Past Tests.

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

 $^{1 \}qquad \text{From 2012 up to the 15. March 2016.}$

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 Commu*nication 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

. # 2012 # GENERAL: | Architecture | Total | scheduled | scheduled [%]
 Tmote
 134
 134
 100.000

 Mica2
 1
 0
 0.000

 Opal
 1
 0
 0.000
 Mica2 | 1 | Opal | 1 | TinyNode | 21 | 0 1 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| 1 - - -100.000 . 241140 | 134 | 100.000 | 241140 | TOTAL | 134 | 100.000 | 100.000 1 OBSERVER / ARCHITECTURE USAGE (TIME): -----Obs | T Tmote || TOTAL 99.054 || 98.588 || 98.588 || 96.413 || 001 | 002 | 99 054 98.588 98.588 004 1 96.413 006 1 96.836 || 98.873 || 007 1 96.836 008 98.873 010 99.158 || 99.158 99.339 || 97.924 || 97.924 || 011 99.339 97.924 97.924 013 014 97.138 || 97.785 || 96.836 || 97.138 97.785 96.836 015 016 017 018 95.515 || 96.836 || 95.515 96.836 019 020 96.836 || 96.836 97.552 || 96.836 || 022 97.552 96.836 023 96.836 || 99.158 || 96.836 || 96.836 024 99.158 96.836 025 026 027 028 95.515 || 97.552 || 95.515 97.552 96.335 || 96.050 || 031 96.335 032 96.050 033 96.335 II 96.335 95.144 || 95.144 200 201 95.144 95.144 202 95.144 || 95.144 1 204 95.144 95.144

TOTAL	104.078	l I	l					
OBSERVER /	ARCHITECTUR	E USAGE (# TES	STS):					
Obs	Tmote	TOTAL	-					
001	85.075	85.075						
002	84.328	84.328						
004	84.328	84.328						
006 1	75.373	1 75.373						
007 1	(3.134 94 329	1 73.134						
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						
	73 134	1 79.001						
018	70.896	70.896						
019	73.134	73.134						
020	73.134	73.134						
022	76.866	1 76.866						
023 1	73.134	73.134 73.134						
025	81.343	81.343						
026	73.134	73.134						
027	70.896	70.896						
028	76.866	1 76.866						
031	73 881	73.881						
033	74.627	1 74.627						
200	70.149	70.149						
201	70.149	70.149						
202	70.149	1 70.149						
TOTAL	100.000	H]					
			•					
# ########### GENERAL:	* * * * * * * * * * * * *	*********	*****	******	******	*****	******	# # # # # # #
# Tests	schedule	d scheduled	[%]					
4095	253	9 62.	.002					
Archite	cture Tot	al scheduled	l sche	duled [%]	- 			
i I	Tmote 25 Iris 1	68 2539 58 (98.871	, 			
Tin	yNode 5	20 0		0.000	1			
I.	Ópal 3	79 (0.000	1			
Wi	smote 2	91 (0.000	1			
 	Mica2 1	50 ()	0.000	1			
					1			
Time Se	quential [s] Time Parall	.el [s]	Time Redu	ced [s] Tim	e Reduced [%]		
	462144	9 4	4425051		178944	3.872		
ARCHITECT	URE STATIST	ICS:						
Archi	tecture #	Tests # Tests	[%] D	uration Te	sts # no Mu	x # no Mux [%]	Duration no Mux	Duration no Mux [%]
	Tmote	2539 100	0.000	4621	449 150	1 59.118	3095172	66.974
	TOTAL	2539 100	0001	4621	449 150	1 59.118	3095172	66.974
OBSERVER/	ARCHITECTUR	E USAGE (TIME)						
0bs	Tmote	TOTAL	-					
001	93.886	93.886						
002	95.353	95.353						
004	95.268	95.268	l					
1 006 1	93 334	93 334	1					

Construction Construction	010 011 013 014 015 016 017 018 019 022 023 024 025 026 027 028 031 032 033 200 201 2024	92.396 93.639 97.974 93.792 93.792 93.792 94.053 92.374 94.053 92.374 93.707 93.707 92.380 97.866 92.766 96.624 92.811 92.811 92.811	92.396 1 92.396 1 97.974 1 95.725 1 93.297 1 93.792 1 94.053 1 92.374 1 94.053 1 92.374 1 94.487 1 93.707 1 92.380 1 97.266 1 92.766 1 99.624 1 96.821 1 92.811 1		
011 9.3.639 93.639 013 9.7.974 93.797 014 95.725 95.725 015 93.792 93.792 016 93.792 93.792 017 90.347 93.792 018 94.053 91.347 018 94.053 91.347 020 96.138 92.374 0221 94.687 92.374 0231 93.707 92.374 0232 94.687 92.380 0234 92.286 92.380 0235 97.766 92.766 031 96.621 92.766 0321 92.611 92.766 0331 96.621 92.766 0331 96.621 92.766 0331 96.621 92.766 0331 96.621 92.766 0331 96.621 92.766 0311 96.621 92.766 0321 92.019 83.019 0322 92.019 83.019 034 87.754	011 013 014 015 016 017 018 019 022 023 024 025 026 027 032 032 032 032 032 200 201 2024	93.639 97.974 95.725 93.792 93.792 94.053 94.053 94.6138 94.847 95.3707 92.380 97.866 97.866 97.866 92.766 95.624 96.821 95.855	93.639 97.974 95.725 93.297 94.053 92.374 94.053 92.374 94.138 94.847 93.707 92.380 99.245 97.866 99.624 96.821 92.811		
014 97.974 97.974 015 93.297 93.297 016 93.297 93.792 017 90.347 93.792 018 94.053 99.795 017 90.347 90.347 018 94.053 99.453 021 94.053 91.92 022 94.139 92.374 023 93.707 92.451 024 92.380 99.245 025 99.245 99.245 026 97.866 92.766 027 92.766 92.766 031 96.621 92.811 032 92.811 92.811 033 95.855 99.023 031 96.821 90.93 032 92.811 90.93 033 95.855 91.89.03 034 92.811 90.93 035 95.91 95.903 036 7.554 97.91 037 7.554 97.91 038 96.91	013 014 015 016 017 018 019 020 022 023 024 025 026 027 026 027 026 027 028 026 027 028 026 027 028 029 020 020 020 020 020 022 023 024 026 027 026 027 026 027 026 027 026 027 026 027 028 026 027 028 029 028 029 028 029 028 029 029 029 020 00	97.974 95.725 93.792 94.053 94.053 94.053 95.374 95.138 95.438 92.380 93.707 92.380 93.707 92.380 97.866 97.866 97.866 99.624 99.624 95.825	$\begin{array}{cccccc} 97 & 974 & \\ 95 & 725 & \\ 93 & 297 & \\ 90 & 347 & \\ 94 & 053 & \\ 94 & 053 & \\ 94 & 347 & \\ 96 & 138 & \\ 94 & 847 & \\ 93 & 707 & \\ 92 & 380 & \\ 99 & 245 & \\ 97 & 866 & \\ 92 & 766 & \\ 99 & 624 & \\ 96 & 821 & \\ 92 & 811 & \end{array}$		
014 95.725 015 93.297 93.297 016 93.792 93.297 017 90.347 93.297 018 94.053 94.053 019 92.374 92.374 022 94.647 94.651 023 93.792 024 94.647 94.651 025 99.624 99.624 026 97.666 97.624 027 99.624 99.624 031 96.621 96.621 032 93.610 032 98.061 041 17.475 051 7.755	014 015 016 017 019 020 022 023 024 025 024 026 027 028 027 028 023 023 023 023 021 020 022 023 024 023 024 032 033 032 033 032 033 032 033 032 033 032 033 032 033 032 033 033 032 033 032 033 032 033 033 033 033 033 032 033 032 033 032 033 033 033 032 034 032 033 032 033 032 033 032 033 032 032 033 033 032 033 034 03	95.725 93.792 90.347 94.053 96.138 94.847 95.707 92.380 93.707 92.380 92.380 95.245 96.624 96.821 95.855 95.855	$\begin{array}{cccccc} 95.725 & \\ 93.297 & \\ 93.792 & \\ 90.347 & \\ 94.053 & \\ 94.347 & \\ 96.138 & \\ 94.847 & \\ 93.707 & \\ 93.707 & \\ 93.245 & \\ 97.866 & \\ 97.866 & \\ 92.766 & \\ 99.624 & \\ 96.821 & \\ 92.811 & \end{array}$		
015 93.797 93.792 017 90.347 90.347 018 94.053 94.053 020 96.138 92.374 021 94.647 94.847 022 94.847 94.847 023 93.792 93.792 024 93.896 92.396 025 93.496 92.396 026 93.496 92.696 027 94.847 94.653 028 93.655 95.655 020 93.010 92.811 031 94.624 98.053 204 93.010 98.053 2021 43.635 98.053 2021 43.635 87.554 0021 7.554 87.554 0021 7.554 87.554 0021 83.051 89.053 0031 87.475 87.554 004 7.554 87.554 005 84.653 <td>015 016 017 018 020 022 023 024 025 026 027 028 031 032 033 200 201 2024 </td> <td>93.297 93.792 94.347 94.053 92.374 96.138 93.707 93.707 92.380 92.45 97.866 92.766 96.624 92.811 92.815 92.815 </td> <td>93.297 93.792 90.347 94.053 92.374 96.138 94.847 93.707 99.245 97.866 99.624 99.624 96.821 92.811 </td> <th></th> <td></td>	015 016 017 018 020 022 023 024 025 026 027 028 031 032 033 200 201 2024	93.297 93.792 94.347 94.053 92.374 96.138 93.707 93.707 92.380 92.45 97.866 92.766 96.624 92.811 92.815 92.815	93.297 93.792 90.347 94.053 92.374 96.138 94.847 93.707 99.245 97.866 99.624 99.624 96.821 92.811		
010 0.3.47 0.3.47 0.3.47 011 0.3.47 0.3.47 0.3.47 012 0.3.47 0.3.47 0.3.47 013 0.4.05 0.3.47 0.3.47 013 0.4.05 0.3.47 0.3.47 014 0.4.05 0.3.47 0.3.47 015 0.4.05 0.4.05 0.3.47 014 0.4.05 0.4.05 0.4.05 015 0.4.05 0.4.05 0.4.05 015 0.4.05 0.4.05 0.4.05 015 0.4.05 0.4.05 0.4.05 016 0.4.25 0.4.05 0.4.05 017 0.4.25 0.4.05 0.4.05 011 0.4.27 0.4.05 0.4.05 011 1.4.7.75 0.4.05 0.4.05 011 1.4.7.75 0.4.05 0.4.05 011 1.4.7.75 0.4.05 0.4.05 011 1.4.02.8 0.4.05 0.05 011 1.4.02.8 0.4.05 0.05 012 1.8.05.64	016 1 017 1 018 1 019 1 020 1 022 1 023 1 024 1 025 1 026 1 027 1 028 1 031 1 032 1 033 1 200 1 201 1 2024 1	90.347 92.374 92.374 96.138 93.707 93.707 92.380 97.866 92.766 96.624 92.811 95.855	90.347 90.347 94.053 96.138 94.847 93.707 99.245 97.866 99.624 99.624 96.821 92.811		
11 0	017 1 018 1 019 1 020 1 022 1 023 1 025 1 026 1 027 1 028 1 031 1 032 1 033 1 203 1 201 1 2021 1 2024 1	94.053 92.374 96.138 93.707 92.380 92.380 92.380 92.45 97.866 92.766 96.624 96.821 92.811 95.855	90.347 92.374 96.138 94.847 93.707 92.380 99.245 97.866 92.766 99.624 96.821 92.811		
010 92.374 92.374 020 94.487 94.383 022 94.487 94.387 023 93.707 93.707 024 92.380 92.380 025 99.4245 99.4245 026 97.666 97.866 027 92.766 92.766 031 99.6241 96.624 99.6245 92.766 92.766 032 99.6241 96.624 91.43735 18.624 032 92.811 96.624 92.855 95.655 200 89.610 18.010 33 99.6245 18.010 2021 43.735 18.010 2021 89.010 18.010 2021 89.023 18.019 2021 89.061 89.023 2021 88.061 88.061 001 87.475 18.7475 0021 88.061 88.061 0041 87.475 18.4761 012 88.064 88.0791	019 020 022 023 024 025 026 027 031 032 033 200 201 2021 2024	92.374 96.138 94.847 93.707 92.380 97.866 92.766 96.624 96.821 92.811 95.855	92.374 96.138 94.847 93.707 92.380 99.245 97.866 99.624 96.821 92.811		
202 96.138 96.138 203 93.707 93.707 204 92.380 92.380 205 99.245 92.380 205 99.245 92.766 207 92.766 92.766 203 93.024 92.766 203 93.035 95.655 203 93.036 92.766 203 93.039 94.624 303 95.655 92.766 201 83.039 94.624 303 95.655 92.766 202 83.040 89.040 201 83.050 89.040 202 83.040 89.040 204 89.093 1 104 83.056 1 005 Tmote II TOTAL	020 022 023 024 025 026 027 028 031 033 200 201 2021 2024	96.138 94.847 93.707 92.380 97.866 97.866 96.821 92.811 92.811 92.811 92.811	96.138 94.847 93.707 92.380 99.245 97.866 92.766 99.624 96.821 92.811		
023 94.847 94.70 024 92.380 93.707 024 92.380 92.380 025 99.425 99.245 026 97.666 97.666 027 99.245 99.245 028 99.624 99.245 027 99.624 99.245 028 99.624 99.245 028 99.624 99.245 031 96.621 92.766 032 92.811 92.855 032 92.811 96.021 033 95.655 95.000 204 89.003 1 204 89.003 1 204 89.003 1 204 89.003 1 204 89.003 1 204 89.003 1 204 87.755 1 001 87.755 1 0021 87.654 1 0021 87.654 1 0021 87.755 1 0021 <t< td=""><td>022 023 024 025 026 027 028 031 032 033 200 201 2021 2024 </td><td>94.847 93.707 92.380 97.866 97.866 99.624 96.821 92.811 95.855 </td><td>94.847 93.707 92.380 99.245 97.866 92.766 99.624 96.821 92.811 </td><th></th><td></td></t<>	022 023 024 025 026 027 028 031 032 033 200 201 2021 2024	94.847 93.707 92.380 97.866 97.866 99.624 96.821 92.811 95.855	94.847 93.707 92.380 99.245 97.866 92.766 99.624 96.821 92.811		
023 93.707 93.707 024 92.380 92.380 025 99.245 92.766 027 92.766 92.766 027 92.766 92.766 028 99.624 92.611 031 96.821 92.654 032 92.756 92.611 033 95.655 95.651 030 83.010 89.010 201 83.013 89.003 2024 89.093 89.093 204 89.093 89.066 204 89.056 107L 104 87.554 87.554 002 88.066 88.066 004 87.475 86.476 005 86.851 86.476 011 86.530 86.851 013 87.121 85.979 014 85.979 85.979 015 85.979 85.979 016 86.351 86.437 017 <td>023 024 025 026 027 028 031 032 033 200 201 202 204 </td> <td>93.707 92.380 97.245 97.866 92.766 99.624 96.821 92.811 95.855 89.010 </td> <td>93.707 92.380 99.245 97.866 92.766 99.624 96.821 92.811 </td> <th></th> <td></td>	023 024 025 026 027 028 031 032 033 200 201 202 204	93.707 92.380 97.245 97.866 92.766 99.624 96.821 92.811 95.855 89.010	93.707 92.380 99.245 97.866 92.766 99.624 96.821 92.811		
024 92.380 1 92.380 025 99.245 1 99.245 026 97.866 1 97.866 028 99.624 1 99.624 031 96.624 1 96.821 032 92.611 1 96.821 033 92.611 1 96.821 033 92.611 14.96.821 030 92.610 1 89.010 031 92.611 14.37.35 0204 1.89.093 14.43.735 0204 1.89.093 14.37.35 0204 1.89.093 19.093 0211 1.43.735 1.43.735 0204 1.89.093 19.093 0214 1.87.554 11 70.74.1 011 87.554 11 77.554 012 1.87.554 11 77.554 012 1.87.654 187.654 012 1.87.654 187.654 013 1.87.122 1 014 65.979 11 85.379 </td <td>024 025 026 027 031 032 033 200 201 202 202 </td> <td>92.380 99.245 97.866 92.766 99.624 96.821 92.811 95.855 89.010 </td> <td>92.380 99.245 97.866 92.766 99.624 96.821 92.811 </td> <th></th> <td></td>	024 025 026 027 031 032 033 200 201 202 202	92.380 99.245 97.866 92.766 99.624 96.821 92.811 95.855 89.010	92.380 99.245 97.866 92.766 99.624 96.821 92.811		
225 1 99.245 1 226 1 92.766 1 92.766 227 1 92.766 1 92.766 331 96.821 1 92.811 1 92.811 331 96.855 1 92.811 1 92.811 1 331 95.855 1 92.811 1 92.910 1 2001 89.010 1 89.010 1 93.910 1 201 43.735 1 43.735 1 89.093 1 2021 89.0193 1 89.093 1 93.010 1 2024 89.093 1 89.093 1 93.010 1 204 89.063 1 89.093 1 93.010 1 204 89.063 1 89.093 1 93.010 1 204 89.063 1 70.754 87.475 1 87.475 001 1 87.475 1 87.475 1 87.475	025 026 027 028 031 032 033 200 201 202 204	99.245 97.866 92.766 99.624 96.821 92.811 95.855 89.010	99.245 97.866 92.766 99.624 96.821 92.811		
026 97.866 1 97.866 027 92.766 1 92.766 031 96.821 95.855 1 032 92.811 95.855 1 030 95.855 155 1 030 95.855 143.735 1 020 1 89.039 1 021 43.735 1 39.039 022 1 104.028 143.735 020 1 89.039 1 021 143.735 1 145.735 022 1 104.028 11 1 011 147.524 1 77.554 021 143.066 1 77.554 021 183.066 1 86.069 022 183.066 1 86.065 022 183.066 1 86.065 031 187.121 1 87.476 031 187.121 1 87.476 031 187.121 1 87.121 041 85	026 027 028 031 032 033 200 201 202 202 204	97.866 92.766 99.624 96.821 92.811 95.855 89.010	97.866 92.766 99.624 96.821 92.811		
028 92.766 92.766 031 96.821 96.624 031 96.821 96.624 032 92.611 92.611 033 95.855 95.855 200 89.010 89.010 201 43.735 43.735 202 89.019 89.019 204 43.9.031 89.093 204 89.093 89.093 204 43.9.093 89.093 204 1.99.019 89.019 204 1.99.019 89.019 204 1.99.019 1.00 204 1.99.019 1.00 204 1.99.019 1.00 204 1.99.019 1.00 205 Tmote 1 0101 87.554 1.97.554 0102 18.066 88.066 004 187.475 1.87.475 0101 84.049 1.84.876 0102 84.049 1.84.876 011 86.850 1.85.979 111 85.979	027 028 031 032 033 200 201 202 202 204	92.766 99.624 96.821 92.811 95.855 89.010	92.766 99.624 96.821 92.811		
031 98.024 98.024 032 92.011 92.011 032 92.010 92.011 033 95.055 95.055 200 99.010 10 99.010 201 43.735 43.735 2024 99.093 14 99.093 TOTAL 104.028 189.093 001 189.093 1 0024 190.093 1 005 1 Taote 1 001 187.554 187.475 002 88.066 1 88.066 002 88.066 188.066 004 87.475 187.475 005 86.885 188.066 004 87.475 184.049 005 86.885 188.066 010 184.029 185.979 011 86.830 186.830 012 86.931 86.933 013 86.931 86.951 014 85.979 1 015 86.979 1	020 031 032 033 200 201 202 204	99.824 96.821 92.811 95.855 89.010	99.824 96.821 92.811		
032 032.1 032.811 92.811 033 95.855 1 95.855 200 89.010 89.010 89.010 201 43.735 1 89.735 202 189.019 189.019 189.019 204 189.093 11 89.093 204 189.093 11 89.093 204 169.093 11 89.093 204 169.093 11 89.093 204 169.093 11 89.093 204 169.093 11 89.093 204 169.093 11 89.093 205 Tmote 11 ToTAL 001 187.554 11 87.554 002 188.066 188.066 004 187.475 188.475 015 184.075 184.128 016 184.745 184.128 017 184.128 184.128 018 185.979 11 114 185.979 185.979 115	032 033 200 201 202 204	92.811 95.855 89.010	92.811		
033 95.855 95.855 200 89.010 90.010 201 43.735 43.735 202 89.019 99.019 2024 89.028 99.019 2024 89.028 99.019 2024 89.028 99.019 2024 89.028 99.019 2024 89.028 99.019 2024 89.028 99.019 204 89.028 99.019 204 104.028 99.019 007 104.028 11 001 87.554 87.6554 002 88.066 88.066 004 87.475 87.475 006 84.049 84.049 007 84.128 84.128 008 86.851 86.851 010 87.511 87.475 011 86.530 86.373 012 86.051 86.979 015 86.979 86.373 014 85.979 85.979 015 86.981 8	033 200 201 202 204	95.855 89.010			
200 89.010 1 89.010 201 43.735 43.735 202 89.019 1 89.019 204 68.093 1 89.019 204 68.093 1 89.019 204 104.028 1 90.019 204 104.028 1 90.019 205 1 104.028 1 005 1 Tmote 1 TOTAL 001 1 87.554 1 87.554 002 186.066 1 80.066 004 87.475 1 87.475 006 184.049 1 84.128 007 84.128 1 84.501 010 84.876 1 84.976 011 86.530 1 86.545 013 87.121 87.121 014 85.979 1 85.979 015 85.979 1 85.979 016 86.381 86.381 022 86.058 85.900	200 201 202 204	89.010	95.855		
201 43.735 43.735 202 89.093 89.093 89.093 204 89.093 89.093 89.093 204 89.093 89.093 89.093 204 89.093 89.093 89.093 204 104.028 1 90.093 TOTAL 104.028 1 104.028 D05 Tmote 1 TOTAL 001 87.554 1 87.554 002 88.066 88.066 004 87.475 87.475 006 84.049 84.049 007 84.128 86.885 008 86.885 86.885 011 86.576 86.799 015 85.979 85.979 016 86.987 85.979 017 83.064 83.064 018 85.990 85.979 016 86.294 83.084 022 86.058 86.058 023 83.931 83.931 024 83.537 83.537	201 202 204		89.010		
202 89.039 89.039 TOTAL 104.028 89.039 BSERVER/ARCHITECTURE USAGE (# TESTS): BSERVER/ARCHITECTURE USAGE (# TESTS): 001 87.554 1 80.049 002 88.066 1 82.066 004 87.554 1 87.455 001 87.554 1 87.475 004 87.475 1 87.475 005 84.049 84.049 006 84.049 84.049 007 84.128 1 86.385 011 86.385 86.385 012 86.385 86.385 013 87.979 1 85.979 014 85.979 1 85.979 015 185.979 1 85.979 016 86.373 1 83.931 017 83.054 1 83.931 022 86.055 1 86.373 023 83.931 1 83.931 024 83.931 <td>202 204 </td> <td>43.735 </td> <td>43.735</td> <th></th> <td></td>	202 204	43.735	43.735		
204 89.093 89.093 TOTAL 104.028 1 TOTAL 104.028 1 Dbs Tmote 1 Totat Tmote 1 D01 80.066 1 001 87.475 88.066 002 88.066 88.066 004 87.475 87.475 006 84.028 84.128 007 84.128 84.856 010 86.855 86.855 011 86.530 86.530 013 87.121 87.579 014 85.979 85.979 015 86.373 85.900 016 86.371 85.900 017 83.064 83.931 018 85.900 85.900 019 84.049 84.049 022 86.058 85.91 023 83.931 83.931 024 83.537 83.537 025 90.941 90.547 031 87.554 85.70 <	204	89.019	89.019		
TOTAL 104.028 BSERVER/ARCHITECTURE USAGE (# TESTS): BSERVER/ARCHITECTURE USAGE (# TESTS): OD1 87.654 001 87.654 002 88.066 004 87.475 006 84.049 007 84.128 008 86.885 001 87.574 003 86.885 011 86.530 011 86.530 011 86.531 011 86.531 011 86.537 012 81.979 015 85.979 016 86.985 017 83.064 018 86.973 019 84.049 020 86.958 021 86.958 022 86.058 023 83.931 024 85.577 025 90.941 026 87.003 027 84.167		89.093	89.093		
3SERVER/ARCHITECTURE USAGE (# TESTS): Obs I Tmote I TOTAL 001 87.554 II 87.554 002 88.066 II 88.066 004 87.475 II 87.475 006 84.049 II 84.049 007 84.128 II 84.128 008 86.885 II 86.885 011 86.501 86.850 013 87.121 II 87.121 014 85.979 II 85.979 015 85.979 II 85.979 016 86.373 II 86.373 017 83.064 II 83.064 018 85.900 II 86.931 020 86.058 II 86.058 023 83.931 II 83.931 024 83.537 II 83.537 025 90.941 II 84.403 026 87.003 II 87.504 027 84.167 II <td>TOTAL </td> <td>104.028 </td> <td> </td> <th></th> <td></td>	TOTAL	104.028			
Obs Tmote I TOTAL 001 87.554 I 87.554 002 88.066 I 88.066 004 87.475 I 87.475 006 84.049 I 84.049 007 84.128 I 84.049 008 86.885 I 86.885 0101 84.576 I 86.530 0111 86.530 I 86.985.979 015 85.979 I 85.979 016 86.373 I 86.373 017 83.064 I 85.979 018 85.900 I 85.900 018 85.900 I 85.901 022 86.058 I 86.058 023 83.931 I 83.931 024 83.537 I 83.537 025 90.941 90.941 90.941 026 87.003 I 87.554 031 87.554 84.403 I 032 84.403<	BSERVER/AR	CHITECTURE U	JSAGE (# TES:):	
001 87.554 87.554 002 88.066 87.475 004 87.475 87.475 006 84.049 84.049 007 84.128 84.049 008 86.885 86.885 010 84.876 84.876 011 86.530 86.885 013 87.121 87.121 014 85.979 85.979 015 85.979 85.979 016 86.337 86.373 017 83.064 85.900 018 85.900 85.910 019 84.049 84.049 022 86.058 86.294 023 83.931 86.294 024 83.537 83.537 025 90.941 90.941 026 87.003 87.054 027 84.167 84.167 033 86.530 86.530 033 86.530 86.530 033 86.530 86.530 033 86.530 86.530	Obs	Tmote	TOTAL		
002 08.066 01.054 01.054 002 88.066 01.84.049 01.054 006 84.049 81.046 01.054 007 84.128 81.84.049 01.054 008 86.885 84.049 01.054 0101 84.128 86.885 01.054.01 0101 84.876 84.876 01.01 011 86.530 86.885 01.01 013 87.121 87.121 87.121 014 85.979 11.014 85.979 015 85.979 11.055.979 016 86.373 86.373 017 83.064 85.900 018 85.900 85.900 019 84.049 84.049 020 86.058 86.294 0221 86.058 86.058 0231 83.931 83.931 024 83.537 83.537 025 90.941 90.547 031 87.554 87.554 032 84.403 84.403 <	0.01	97 554 11	97 664		
004 87.475 1 87.475 006 84.049 1 84.049 007 84.128 1 84.049 008 86.885 1 86.885 010 84.876 1 84.876 011 86.851 1 86.885 010 84.876 1 84.876 011 86.530 1 86.885 013 87.121 1 87.121 014 85.979 1 85.979 015 85.979 1 85.979 016 86.373 1 86.373 017 83.064 83.064 018 85.900 1 85.979 020 86.294 1 85.91 021 86.058 1 86.058 022 86.058 83.931 1 024 83.931 1 90.941 025 90.941 90.547 90.547 031 87.554 84.403 86.530 032 84.403 86	002 1	88 066 11	88 066 I		
006 84.049 84.049 007 84.128 84.128 008 86.885 84.876 010 84.876 84.876 011 86.851 86.885 011 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 86.833 86.373 017 83.064 85.900 018 85.900 85.900 019 84.049 84.049 020 86.294 86.058 022 86.058 86.058 023 83.931 83.931 024 83.537 83.537 025 90.941 90.547 026 87.003 87.003 031 87.554 86.530 032 84.403 86.530 033 86.530 86.530 033 86.530 86.530 204 3.843 81.843<	004	87.475	87.475		
007 84.128 84.128 008 86.885 86.885 86.885 010 84.876 84.876 084.876 010 86.530 011 86.530 013 87.121 87.121 87.121 87.121 87.121 101 85.979 85.979 85.979 85.979 85.979 86.373 101 86.373 86.373 86.373 101 85.900 101 85.900 101 85.900 102 86.058 86.058 86.105 81.93 102 83.931 83.537 102 83.537 90.941 90.941 90.941 90.941 90.941 </td <td>006 I</td> <td>84.049 </td> <td>84.049</td> <th></th> <td></td>	006 I	84.049	84.049		
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 85.900 018 85.900 85.900 019 84.049 84.049 020 86.294 86.294 021 86.058 86.931 022 86.058 86.931 023 83.931 83.931 024 83.537 83.537 025 90.941 90.941 026 87.003 87.003 027 84.167 84.167 033 86.530 86.530 033 86.530 86.530 033 86.530 86.530 204 83.863 81.883 205 90.547 86.530 84.403 86.530 86.530 202 81.843 81.	007 I	84.128	84.128		
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 85.979 017 83.064 83.064 018 85.900 85.900 019 84.049 84.049 020 86.294 86.294 022 86.058 86.058 022 86.931 83.931 024 83.537 83.537 025 90.941 90.941 026 87.003 87.003 027 84.167 84.403 031 87.554 86.530 032 84.403 86.530 033 86.530 86.530 033 86.530 86.530 200 81.843 81.843 204 82.198 81.883	008	86.885	86.885		
013 87.121 87.121 014 85.979 85.979 015 85.979 85.979 015 85.979 85.979 016 86.373 85.979 017 83.064 85.900 018 85.900 85.900 019 84.049 86.294 020 86.294 86.294 022 86.058 86.058 023 83.931 83.931 024 83.537 83.931 025 90.941 90.941 026 87.003 87.003 027 84.167 84.403 031 87.554 86.530 033 86.530 86.530 033 86.530 86.530 200 81.843 81.843 201 43.876 43.876 202 81.883 81.883 204 82.198 81.883	010	84.876	84.876		
013 01.121 01.121 014 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 85.900 020 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 84.403 031 87.554 86.530 032 84.403 86.530 033 86.530 86.530 200 81.843 81.843 201 43.876 43.876 202 81.883 81.883 204 82.198 82.198	011	80.530	80.530		
015 85.979 85.979 85.979 016 86.373 86.373 86.373 017 83.064 85.900 86.373 018 85.900 85.900 86.294 020 86.294 86.294 86.294 021 86.294 86.058 86.058 022 86.058 86.058 86.058 023 83.931 83.931 83.931 024 83.537 83.537 83.537 025 90.941 90.941 026 87.003 87.003 87.003 027 84.167 81.84.167 1 031 87.554 87.554 90.547 032 84.403 87.554 86.530 033 86.530 86.530 1 033 86.530 86.530 1 200 81.843 81.843 1 201 43.876 43.876 1 202 81.883 81.883 1 204 82.198 82.198 1 82.198 </td <td>014</td> <td>85.979 </td> <td>85.979</td> <th></th> <td></td>	014	85.979	85.979		
016 86.373 86.373 017 83.064 83.064 018 85.900 83.064 019 84.049 84.049 020 86.294 86.294 021 86.357 86.058 022 86.3537 83.931 023 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 86.530 032 84.403 86.530 033 86.530 86.530 200 81.843 81.843 201 43.876 43.876 202 81.883 81.883 204 82.198 81.883	015	85.979	85.979		
017 83.064 83.064 018 85.900 85.900 019 84.049 86.294 020 86.294 86.294 021 86.294 86.294 022 86.958 86.058 023 83.931 83.931 024 83.537 83.931 025 90.941 90.941 026 87.003 84.167 027 84.167 90.547 031 87.554 97.554 032 84.403 81.843 033 86.530 86.530 200 81.843 81.843 204 82.198 82.198 204 82.198 82.198	016	86.373	86.373		
018 85.900 1 85.900 019 84.049 1 85.900 020 86.294 1 86.058 022 86.058 1 86.058 023 83.931 83.931 1 024 83.537 1 83.537 025 90.941 90.941 90.941 026 87.003 1 87.003 027 84.167 1 90.547 031 87.554 1 90.547 032 84.403 1 86.530 033 86.530 1 86.530 200 81.843 81.843 201 43.876 1 202 81.883 81.883 204 82.198 82.198	017	83.064	83.064		
019 1 84.049 1 84.049 1 84.049 1 020 1 86.294 1 86.294 1	018	85.900	85.900		
022 86.058 1 86.058 023 83.931 83.931 83.931 024 83.537 83.537 83.537 025 90.941 90.941 90.941 026 87.003 87.003 87.003 027 84.167 84.167 84.167 028 90.547 90.547 90.547 031 87.554 84.403 87.554 032 84.403 84.403 84.530 200 81.843 81.843 81.843 201 43.876 43.876 81.843 202 81.883 81.883 81.883 204 82.198 82.198 81.883	019	84.049	84.049		
023 83.931 03.931 83.931 024 83.537 83.931 90.941 025 90.941 90.941 90.941 026 87.003 87.003 87.003 027 84.167 84.167 84.167 028 90.547 90.547 90.547 031 87.554 84.403 84.403 032 84.403 84.403 84.403 033 86.530 86.530 86.530 200 81.843 81.843 81.843 201 43.876 43.876 81.843 202 81.883 81.883 81.883 204 82.198 82.198 82.198	020 1	00.294 86.058	86 058		
024 83.537 83.537 83.537 025 90.941 90.941 90.941 026 87.003 87.003 87.003 027 84.167 84.4167 84.167 028 90.547 90.547 90.547 031 87.554 84.403 81.530 032 84.403 84.403 86.530 033 86.530 86.530 86.530 200 81.843 81.843 81.843 201 43.876 43.876 202 202 81.883 81.883 204 82.198 82.198	023	83.931	83.931		
025 90.941 90.941 026 87.003 87.003 027 84.167 84.167 028 90.547 90.547 031 87.554 84.403 032 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	024	83.537	83.537		
026 87.003 87.003 027 84.167 84.167 028 90.547 90.547 031 87.554 87.554 032 84.403 87.554 033 86.530 86.530 200 81.843 81.843 201 43.876 43.876 202 81.883 81.883 204 82.198 82.198	025	90.941	90.941		
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 81.843 201 201 43.876 43.883 202 81.883 81.883 204 82.198 82.198	026	87.003	87.003		
028 90.547 90.547 90.547 031 87.554 87.554 87.554 032 84.403 87.554 87.554 033 86.530 86.530 86.530 200 81.843 81.843 81.843 201 43.876 43.876 81.883 202 81.883 81.883 82.198	027	84.167	84.167		
031 07.054 07.054 87.554 032 84.403 84.403 1 033 86.530 84.403 1 200 81.843 81.843 1 201 43.876 43.876 1 202 81.883 81.883 1 204 82.198 81.883 1	028	90.547	90.547		
033 04.403 04.403 033 86.530 86.530 200 81.843 81.843 201 43.876 43.876 202 81.883 81.883 204 82.198 82.198	031	87.554	87.554		
200 81.843 1 81.843 1 201 43.876 43.876 1 202 81.883 81.883 1 204 82.198 82.198 1	032	04.403 86 520	04.403 86 520		
201 43.876 43.876 202 202 81.883 81.883 204 82.198 82.198	200	81.843	81.843		
202 81.883 81.883 204 82.198 82.198 	201	43.876	43.876		
204 82.198 82.198	202	81.883	81.883		
	204 	82.198	82.198 		
TOTAL 100.000	TOTAL	100.000	ا 		
TUTAL 100.000	201 202 204 TOTAL	43.876 81.883 82.198 100.000	43.876 81.883 82.198 		

	ACM2 8 CC430 11 Mica2	25 823 49 133 42 0		99.758 11.575 0.000				
Time Se	equential [s] Time Parall	el [s] Tin	me Reduced [s]	Time Red	uced [%] 		
	1189660	8 10	185745	1710863	 	14.381		
		1 4 4						
CHITECT	ORE STATIST	105:						
Archi	itecture #	Tests # Tests	[%] Dura	tion Tests #	no Mux #	no Mux [%] Dur	ation no Mux Durat:	ion no Mux [
	Tmote	4376 82	.071	10775028	2031	46.412	3867085	35.8
	ACM2	823 15	.435	1018800	0	0.000	0	0.0
		133 2 	.4941	1127601			2967095	0.0 20 E
					20311			
3SERVER/ Obs	ARCHITECTUR Tmote	E USAGE (TIME) ACM2	: CC430	 TOTAL				
001	91.190 89.400	0.740	U.256 0 594	92.185 98.144				
003	29.178	1.789	0.000	30.966				
004	90.330	8.064	0.529	98.922				
006 I	98.489	1.433	0.620	100.543				
007	93.647	0.000	0.022	93.670				
010 1	89.808	7.832	0.279	97.918				
011	93.292	0.000	0.001	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				
018	93.575	0.000	0.000	93.666				
019	94.230	0.000	0.008	94.238				
020 I	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 1	96 861		0.183	96 861				
026	97.182	0.000	0.000	97.182				
027 I	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				
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 89.531				
 TOTAT	105 785	 1 10 002 l	1 009					
0bs	Tmote	E USAGE (# 1ES ACM2	CC430	 TOTAL				
001	62.228	2.926	0.638	 65.791				
002 İ	59.677	9.565	1.369	70.611				
003 I	25.525	5.814	0.000	31.339				
004	60.709	9.377	1.238	11 71.324				
000 1	53 957	1 4.0/V I	1.519	54 051				
008	60.296	9.696	0.788	11 70.780				
010 I	55.608	0.000	0.056	55.664				
011	53.788	0.000	0.000	53.788				
014	55.077		U.U55 0 038	55.133 55.970				
015	62.697	3.507	0.431	66.635				
016	64.141	0.000	1.088	65.229				
017 I	55.701	0.000	0.038	55.739				
018	54.201	0.000	0.188	11 54.389				
019	56.133 60 579		0.038	56.170 60.579				
022 1	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	11 59.959				
V21	54.295	1 0.000 l	0.131	11 54.426				

028	65.585	1.	.069	0.000	66.65	4						
029	1.725	1 0.	.000	0.000	1.72	5						
031	64.010	1 0.	.056	0.000	64.06	6						
032	57 314	I 1.	265	0.000	II 66.89	8						
200	50.581	1 0.	.000	0.244	1 50.82	5						
201	44.805	0	.000	0.075	44.88	0						
202	41.335	İ 0.	.000	0.188	41.52	3						
204	50.825	I 0.	.000	0.431	51.25	7						
TUTAL	82.071	15.	.435	2.494								
aximal N	Number of Pa	arallel t	tests: 1	. 0								
*****	********	*******	* * * * * * * *		*****	******	*****	# # # # # #	******	#####	# # #	
											#	
2015											#	
########	********	******	* * * * * * * *		******	*******	*****	# # # # #	******	####	# ###	
# Tests	s schedule	d sche		 [¥]								
9006	6 722	3	80.2	202								
Archite	ecture Tot	al sch	heduled	schedu]	led [%]							
	Tmote 55	500 I	5500		100.000							
	None	4	0	· · ·	0.000							
Tir	nyNode 1	82	150	1	82.418							
Ope	enMote 3	35	331		98.806							
	Upal 3	566 I	1286		0.000							
	CC430 10	001	1200		2 542							
Wi	ismote	8	20	i :	100.000							
Time Se	equential [s 1358591] Time 4	Paralle	91 [s] Ti 977309	ime Reduced 2608	[s] Time R 	educed [%] 19.201					
Time Se RCHITECT	equential [s 1358591 TURE STATIST] Time 4 ICS:	Paralle 109	91 [s] Ti 977309	ime Reduced 2608	[s] Time R 	educed [%] 19.201 		ation no b	·		оо Мих Г
Time Se RCHITECT Archi	equential [s 1358591 TURE STATIST itecture #] Time 4 TICS: Tests 4	Paralle 109 # Tests	01 [s] T3 077309 	ime Reduced 2608 ation Tests	[s] Time R 605 	educed [%] 	 Dur	ation no N	1ux I	Duration r	10 Mux [
Time Se RCHITECT Archi	equential [s 1358591 TURE STATIST itecture # Tmote] Time 4 ICS: Tests 4 5500	Paralle 109 # Tests 76.	01 [s] T: 077309 [%] Dura 146	ime Reduced 2608 ation Tests 11460394	[s] Time R 605 	educed [%] 19.201 # no Mux [% 55.833	Dur	ation no M 90914	1ux I	Duration r	10 Mux [
Time Se RCHITECT Archi	equential [s 1358591 TURE STATIST itecture # Tmote TinyNode] Time 4 ICS: Tests 4 5500 150	Paralle 109 # Tests 76. 2.	51 [s] T 777309 [%] Dura 146 077	ime Reduced 2608 ation Tests 11460394 129000	[s] Time R 605 # no Mux 3071 0	educed [%] 	Dur	ation no N 90914	1ux I 	Duration r	79.3 0.0
Time Se RCHITECT Archi T	equential [s 1358591 TURE STATIST itecture # Tmote TinyNode Wismote DaaMote] Time 4 TCS: Tests 4 5500 150 8 221	Paralle 109 # Tests 76. 2. 0.	1 [s] T 77309 [%] Dura 146 077 111 582	ime Reduced 2608 ation Tests 11460394 129000 10080 28060	[s] Time R 605 # no Mux 3071 0 8 7	educed [%] 	Dur 	ation no N 90914 1000	1ux I 19 0 80	Duration I	79.3 0.0 100.0
Time Se RCHITECT Archi I	equential [s 1358591 TURE STATIS1 itecture # Tmote TinyNode Wismote OpenMote ACM2] Time 4 TCS: Tests 4 5500 150 8 331 1286	Paralle 109 # Tests 76. 2. 0. 4. 17.	51 [s] T 777309 [%] Dura 146 0771 111 583 804	ime Reduced 2608 ation Tests 11460394 129000 10080 389060 1633620	[s] Time R 605 # no Mux 3071 0 8 7 0	educed [%] 	Dur Dur 	ation no M 90914 100 36	1ux 1 19 0 080 560 0	Duration r	79.3 79.3 0.0 100.0 0.9
Time Se RCHITECT Archi I	equential [s 1358591 TURE STATIST itecture # Tmote TinyNode Wismote OpenNote ACM2 CC430] Time 4 TCS: Tests 4 5500 150 8 331 1286 26	Paralle 109 # Tests 76. 2. 0. 4. 17. 0.	01 [s] Ti 077309 [%] Dura 146 077 111 583 804 360	ime Reduced 2608 ation Tests 11460394 129000 10080 389060 163620 17640	[s] Time R 605 # no Mux 3071 0 8 7 0 0	educed [%] # no Mux [%: 55.83 0.000 100.000 2.111 0.000 0.000	Dur 	ation no M 90914 100 36	1ux I 19 0 080 060 0 0	Duration r	79.3 79.3 0.0 100.0 0.9 0.0 0.0
Time Se RCHITECT Archi T	equential [s 1358591 TURE STATIST itecture # Tmote TinyNode Wismote OpenMote ACM2 CC430] Time 4 TCS: Tests 4 5500 150 8 331 1286 26	Paralle 109 # Tests 76. 2. 0. 4. 17. 0.	21 [s] Ti 277309 [%] Dure 146 077 111 583 804 360	ime Reduced 2608 ation Tests 11460394 129000 10080 389060 163620 17640	[s] Time R 605 # no Mux 3071 0 8 7 0	educed [%] # no Mux [%: 55.83(0.000 100.000 2.111 0.000 0.000	Dur	ation no M 90914 100 30	fux I 19 0 0 0 0 0	Duration r	10 Mux [79.3 0.0 100.0 0.9 0.0 0.0
Time Se RCHITECT Archi T	equential [s 1358591 TURE STATIS1 itecture # Tinote TinyNode Wismote OpenMote ACM2 CC430 TOTAL] Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223	Paralle 109 # Tests 76. 2. 0. 4. 17. 0. 100.	01 [s] Ti 077309 [%] Durs 146 077 111 583 804 360 000	ime Reduced 2608 ation Tests 11460394 129000 10080 389060 163620 17640 13639794	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086	educed [%] 	Dur 	ation no M 90914 100 36	1ux I 	Duration r	To Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi C	equential [s 1358591 TURE STATIS1 itecture # Tmote TinyNode Wismote OpenMote ACM2 CC430 TOTAL] Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223 E HEAGE	Paralle 109 # Tests 76. 2. 0. 4. 17. 0. 100.	01 [s] T 077309 [%] Dura 146 0771 111 583 804 360 .000	ime Reduced 2608 11460394 129000 10080 389060 1633620 17640 13639794	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086	educed [%] 	Dur 	ation no M 90914 100 36 91051	fux I 	Duration r	10 Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi I BSERVER/	equential [s 1358591 TURE STATIST itecture # TinyNode Wismote OpenMote CC430 TOTAL /ARCHITECTUR] Time 4 TCS: 5500 150 8 331 1286 26 7223 E USAGE	Paralle 109 # Tests 76. 2. 0. 4. 177. 0. 100. (TIME):	01 [s] T 077309 [%] Dura 146 077 111 583 804 360 000	ime Reduced 2608 ation Tests 11460394 129000 10880 389060 1633620 17640 13639794	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086	educed [%] 19.201 # no Mux [%] 55.83(0.000 100.000 2.111 0.000 42.721	Dur	ation no N 90914 100 30	fux I -19 0 0 0 0 0 	Duration r	to Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi T C BSERVER/ Obs	equential [s 1358591 TURE STATIST itecture # Tmote Wismote Wismote OpenMote CC430 TOTAL ARCHITECTUR] Time 4 Tests 4 5500 150 8 331 1286 26 7223 E USAGE Tiny)	Paralle 109 # Tests 76. 2. 0. 4. 177. 0. 100. (TIME):	D1 [s] T D77309 [%] Dura 146 077 111 583 804 360 000 Wismote	ime Reduced 2608 ation Tests 11460394 129000 10880 389060 1633620 17640 13639794 13639794	[s] Time R 605 # no Mux 3071 0 8 7 0 0 0 3086 ACM	educed [%] 19.201 # no Mux [%] 55.83(0.000 100.000 2.111 0.000 0.000 42.721 2. CC4	Dur 	ation no N 90914 100 30 91051 	fux 1 -19 0 80 860 0 	Duration r	10 Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi I BSERVER / Obs 001	equential [s 1358591 TURE STATIS1 itecture # Tmote TinyNode Wismote OpenMote ACM2 CC430 TOTAL /ARCHITECTUR Tmote 91.082	Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223 E USAGE TinyN 0.	Paralle 109 # Tests 76. 2. 0. 4. 177. 0. 1000. (TIME): Node -000	01 [s] T 077309 [%] Dura 146 0771 111 583 804 360 000 Wismote 0.000	ime Reduced 2608 ation Tests 11460394 129000 10080 389060 1633620 17640 13639794 13639794 OpenMote OpenMote	[s] Time R 605 # no Mux 3071 0 8 7 0 0 0 3086 4.000	educed [%] 19.201 # no Mux [%] 55.83(0.000 100.000 2.111 0.000 0.000 42.721 2 CC4 9 0.13	Dur Dur 	ation no N 90914 100 3(91051 	fux I 	Duration r	10 Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi BSERVER/ Obs 001 002	equential [s 1358591 TURE STATIST itecture # Tmote TinyNode Wismote OpenMote ACM2 CC430 TOTAL /ARCHITECTUR Tmote 91.082 90.117	Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223 E USAGE Tinyl 0. 0.	Paralle 109 Tests 76. 2. 0. 4. 177. 0. (TIME): Node .000 .000	<pre>b1 [s] Ti p77309 p77309 [%] Durr [%] Durr 146 077 111 583 804 360 0000 Wismote 0.000</pre>	ime Reduced 2608 2608 11460394 129000 10080 389060 1633620 17640 13639794 13639794 0penMote 0penMote 0.000	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086	educed [%] 	Dur 	ation no M 90914 100 30 91051 	Mux I 19 0 80 860 0 59 	Duration r	10 Mux [79.3 0.0 100.0 0.9 0.0 0.0 0.0
Time Se RCHITECT Archi USSERVER/ Obs 001 0 002 0 003	equential [s 1358591 TURE STATIST itecture # Tmote TinyNode Mismote OpenMote ACM2 CC430 TOTAL /ARCHITECTUR Tmote 91.082 90.117 92.278] Time 4 TCS: 5500 1500 1500 8 331 1286 26 7223 FUSAGE Tiny] 0. 0 0	Paralle 109 # Tests 76. 2. 0. 4. 17. 0. 100. (TIME): Node .000 .000 .000	<pre>b1 [s] T: p77309 </pre>	ime Reduced 2608 2608 11460394 129000 10080 389060 1633620 17640 13639794 13639794 13639794	[s] Time R 605 # no Mux 3071 0 8 7 0 3086 11.15 1 1.22 12.55	educed [%] 	Dur	ation no M 90914 100 36 91051 TOTAL 102.371 101.490 105.760	fux 1 	Duration r	To Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi T () BSERVER/ Obs 001 002 003 004 004	equential [s 1358591 TURE STATIST itecture # Tmote TinyNode Wismote OpenMote ACM2 CC430 TOTAL /ARCHITECTUR Tmote 91.082 90.117 92.278 93.072] Time 4 ICS: 5500 150 8 331 1286 26 7223 E USAGE Tiny} 0, 0 0, 0 0, 0 0, 0 0, 0 0, 0 0, 0 0,	Paralle 109 # Tests 76. 2. 0. 4. 17. 0. 100. (TIME): Node .000 .000 .000 .000	D1 [s] T D77309 (%] Dura 146 077 111 583 804 360 0000 Wismote 0.000 0.000 0.000 0.000 0.000	ime Reduced 2608 2608 11460394 129000 10080 1633620 17640 13639794 13639794 13639794 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086 11.15 11.22 12.59 11.23	educed [%] 	Dur 	ation no N 90914 100 36 91051 		Duration r	
Time Se RCHITECT Archi BSERVER/ 0bs 001 002 003 004 006 007	equential [s 1358591 TURE STATIST itecture # Tmote Wismote QpenMote ACM2 CC430 TOTAL /ARCHITECTUR Tnote 91.082 90.117 92.278 93.072 80.936 4 84] Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223 F USAGE Tiny] 0, 0 0 0 0 0 0 0 0 0	Paralle 109 # Tests 766 2. 0. 4. 177. 0. (TIME): Node .000 .000 .000 .000 .000 .000 .000	<pre>p1 [s] T: p77309 </pre>	ime Reduced 2608 2608 11460394 129000 10080 389060 1633620 17640 13639794 13639794 13639794 0.000 0.0000 0.0000 0.0885 0.0000	[s] Time R 605 # no Mux 3071 0 8 7 0 0 0 3086 11.15 11.22 12.59 11.23 12.14	educed [%] 	Dur Dur 	ation no N 90914 100 36 91051 TOTAL 102.371 101.490 105.760 104.456 93.842 24.82	ux 1 -19 0 0 0 0 0 59 - - 1 1 1 1 	Duration r	10 Mux [79.3 0.0 100.0 0.9 0.0 66.7
Time Se RCHITECT Archi BSERVER / Obs 001 1 002 1 003 1 004 0 006 0 007 0 008 0	equential [s 1358591 TURE STATIST itecture # Tmote TinyNode Wismote OpenMote ACM2 CC430 TOTAL /ARCHITECTUR Tmote 91.082 90.117 92.278 93.072 80.936 83.648 95.563] Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223 E USAGE Tinyl 00 00 00 1. 00	Paralle 109 Tests 76. 2. 0. 4. 177. 0. (TIME): Node 000 000 000 000 000 175 000	01 [s] T 077309 077309 (%] Dura 146 0771 111 583 804 360 000 000 Wismote 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000	ime Reduced 2608 211460394 129000 10080 389060 1633620 17640 13639794 13639794 13639794 0.000 0.000 0.000 0.000 0.000 0.0759 0.000 0.001	[s] Time R 605 # no Mux 3071 0 8 7 0 0 0 3086	educed [%] 	Dur 	ation no M 90914 100 36 91051 	fux I 19 0 0 0 0 0 0 0 0	Duration r	no Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi I BSERVER/ Obs 001 003 004 003 004 003 004 006 007 008 007	equential [s 1358591 TURE STATIST itecture # Tmote TinyNode Wismote OpenNote ACM2 CC430 	Time 4 ICS: Tests 4 5500 150 8 331 1266 266 7223 E USAGE Tinyl 00 00 00 00 00 00 00 0	Paralle 109 # Tests 776. 2. 0. 4. 177. 0. 100. (TIME): Node .000 .000 .000 .000 .000 .000 .175 .000 .175 .000 .175 .000 .175 .000 .000 .175 .000 .000 .175 .000	<pre>b1 [s] T; p77309 p77309 p774000 p77400 pre>	ime Reduced 2608 2608 ation Tests 11460394 129000 10080 389060 1633620 17640 13639794 0.000 0.000 0.0000 0.0885 0.0000 0.0805 0.0000 0.0759 0.0000 0.0759 0.0000 0.0911 0.000	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086	educed [%] 	Dur 	ation no N 90914 100 36 91051 TOTAL 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848	fux 1 19 0 560 0 59 	Duration r	TO Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi T C BSERVER/ Obs 001 003 004 006 007 008 007 008 001 011	equential [s 1358591 TURE STATIS1 itecture # Tmote TinyNode ACM2 CC430 CC430 ACM2 CC430 TOTAL /ARCHITECTUR 91.082 90.117 92.278 93.072 80.936 83.648 95.563 83.041] Time 4 ICS: Tests 4 5500 150 8 331 1286 26 7223 7223 FUSAGE Tiny] 0. 0 0. 0 0. 0 1. 1 0. 1 0 0. 1 0 0 0 0 0 0 0 0 0 0	Paralle 109 # Tests 76. 2. 0. 4. 17. 0. 100. (TIME): Node .000	<pre>bl [s] T; p77309 p77309 [%] Dura 146 0771 111 583 804 360 0000 .0000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000</pre>	ime Reduced 2608 2608 11460394 129000 10080 1633620 17640 13639794 13639794 13639794 13639794 0.000 0.0.000 0.0.885 1.0.000 0.0.759 0.0.000 1.0.911 0.0000	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086	educed [%] 	Dur 	ation no N 90994 100 36 91051 TOTAL 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208	fux 1 19 0 80 80 0 0 59 - - 1 1 1 1 1 1 1 1 1	Duration r	To Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi DESERVER/ Dbs 001 002 003 004 004 003 001 003 001 003 001 003 001 003 003 001 003 004 003 005 00	equential [s 1358591 TURE STATIST TURE STATIST Tiecture # Tmote Wismote Wismote QpenMote ACM2 CC430 TOTAL TOTAL ARCHITECTUR 91.082 90.117 92.278 93.072 80.936 83.648 95.563 83.641 89.627] Time 4 ICS: 5500 150 8 331 1286 26 7223 F USAGE TinyP 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.	Paralle 109 # Tests 76. 2. 0. 4. 17. 0. 100. (TIME): Node .000	D1 [s] T D77309 D77309 [%] Dura 146 077 111 583 804 360 0000 Wismote 0.0000 0.00000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.00000 0.00000 0.00000 0.00000 0.000000 0.00000000	ime Reduced 2608 2608 11460394 129000 10080 389060 1633620 17640 13633794 13637	[s] Time R 605 # no Mux 3071 0 8 7 0 3086	educed [%] 	Dur 	ation no N 90914 100 36 91051 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802		Duration r	
Time Se RCHITECT Archi T C BSERVER/ Obs 001 002 003 004 006 007 003 004 006 007 003 004 006 007 003 004 006 007 003 004 006 007 003 004 006 007 003 004 006 007 003 004 006 007 006 007 007 007 007 007	equential [s 1358591 TURE STATISI Ture STATISI Tiecture # Tmote TinyNode Wismote OpenMote ACM2 CC430 TOTAL /ARCHITECTUR Tmote 91.082 90.117 92.278 93.072 80.936 83.648 95.563 93.673 83.041 89.627 87.723] Time 4 TCS: Tests 4 5500 1500 1500 1286 26 7223 E USAGE Tinyl 0.0 0.0 0.0 1.1 1.1	Paralle 109 Tests 76. 0. 4. 177. 0. 1000. (TIME): Node .000 .0	<pre>b1 [s] T: p77309 p77309 [%] Dura 146 0771 111 583 804 360 000 0000 0000 0.0000 0.000 0.000 0.000 0.000 0.00000 0.00000 0.000000</pre>	ime Reduced 2608 2608 11460394 129000 1080 389060 1633620 17640 13639794 13639794 13639794 13639794 0.000 0.000 0.000 0.000 0.000 0.010 0.0000 0.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000 0.000000 0.00000000	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086 11.15 11.22 12.59 11.23 12.14 0.000 11.72 0.000 0.000	educed [%] 	Dur 	ation no N 90914 100 36 91051 TOTAL 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 88.907	fux I 19 0 80 660 0 0 59 - 1 1 1 1 1 1 1 1 1 1 1 1 1	Duration r	no Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi T C BSERVER / BSERVER / O01 003 004 003 004 007 008 001 011 013 014 015 014 015 016 016 017 018 016 017 018 018 018 018 018 018 018 018	equential [s 1358591 TURE STATIST itecture # Tmote TinyNode Wismote OpenNote ACM2 CC430 TOTAL /ARCHITECTUR 91.082 90.117 92.278 93.673 83.648 95.563 93.673 83.041 89.627 87.723 94.791 95.751 95.751 95.751 95.751 9] Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223 E USAGE Tinyl 00 0 00 0 00 1	Paralle 109 # Tests 776. 2. 0. 4. 177. 0. 100. (TIME): Node .000 .000 .000 .000 .000 .000 .000 .175 .167 .175 .000 .175 .167 .175 .000 .000 .000 .175 .175 .000 .175 .175 .000 .000 .175 .175 .000 .175 .175 .000 .175 .175 .000 .175 .175 .175 .000 .175 .000 .175 .175 .175 .000 .000 .175 .175 .000 .000 .175 .175 .000 .000 .175 .175 .000 .000 .175 .175 .000 .000 .175	<pre>b1 [s] T; p77309 p77309 [%] Durr p77309 p77309 p7730000 p7730000 p700000 p7000000 p7000000 p7000000 p7000000 p7000000 p7000000 p70000000 p70000000 p7000000 p70000000 p70000000 p7000000000 p70000000 p70000000000</pre>	ime Reduced 2608 2608 11460394 129000 10080 389060 1633620 17640 13639794 13639794 0.0000 0.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000000	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086 11.15 11.22 112.59 112.25 112.25 112.25 112.25 112.25 112.24 12.25 112.24 0.00 0 0.00 0 0.000 0.000 0.000 0.000	educed [%] 	Dur 	ation no N 90914 100 36 91051 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 88.907 106.577 100.65577	fux 1 19 0 560 0 559 	Duration r	to Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi T C BSERVER/ Obs D C C C C C C C C C C C C C	equential [s 1358591 TURE STATISI itecture # Tmote TinyNode Wismote OpenMote ACM2 CC430 TOTAL /ARCHITECTUR 91.082 90.117 92.278 93.072 80.936 83.648 95.563 93.673 87.723 87.723 94.791 94.962 87.110] Time 4 ICS: 5500 150 8 331 1266 26 7223 FUSAGE TinyT 0. 1 0. 1 0. 1 1. 1 1. 1 1. 1 1. 1 1. 1	Paralle 109 # Tests 76. 2. 0. 4. 17. 0. 100. (TIME): Node	<pre>b1 [s] T; p77309 p77309 p773</pre>	ime Reduced 2608 2608 11460394 129000 10080 1633620 1633620 17640 13639794 13639794 13639794 13639794 0.000 0.0000 0.885 0.0000 0.0759 0.0000 0.0759 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000 0.00000000	[s] Time R 605 # no Mux 3071 0 8 71 0 3086	educed [%] 	Dur 	ation no M 90914 100 36 91051 TOTAL 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 85.907 106.577 108.854 88.907	fux 1 19 0 360 0 59 	Duration r	To Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi T C BSERVER/ Obs 001 002 003 004 006 007 008 007 008 007 008 001 003 004 006 10 007 01 01 01 01 01 01 01 01 01 01	equential [s 1358591 TURE STATIST itecture # TimyNode Wismote OpenMote CC430 CC430 TOTAL CC430 TOTAL /ARCHITECTUR 91.082 90.117 92.78 93.072 80.936 83.041 89.627 87.723 83.041 89.627 87.723 83.94 94.962 87.110 91.177] Time 4 ICS: Tests 4 5500 150 8 331 1286 26 7223 FUSAGE USAGE USAGE I Tinyl 0.0 0.0 0.0 1.0 1	Paralle 109 # Tests 76. 2. 0. 4. 17. 0. 100. (TIME): Node .000 .000 .000 .000 .000 .000 .000 .175 .175 .167 .175 .175 .175 .000 .000 .175 .175 .175 .167 .175 .175 .000 .000 .000 .175 .175 .167 .175 .167 .175 .000 .000 .000 .175 .175 .167 .175 .167 .175 .000 .000 .000 .175 .175 .167 .175 .167 .175 .000 .000 .000 .175 .175 .167 .175 .000 .000 .000 .175 .175 .167 .175 .167 .000 .000 .000 .000 .175 .175 .167 .000 .000 .000 .000 .175 .175 .167 .000 .000 .000 .000 .000 .175 .175 .167 .000 .000 .000 .000 .000 .000 .175 .175 .167 .000	D1 [s] T D77309 D77309 [%] Dura 146 0771 111 583 804 360 0000 Wismote 0.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000 0.000000 0.00000000	ime Reduced 2608 2608 11460394 129000 10080 389060 389060 1633620 17640 13639794 13639794 13639794 13639794 13639794 10000 0.00000 0.0000 0.0000 0.0000 0.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000	[s] Time R 605 # no Mux 3071 0 8 7 0 3071 0 0 3086 11.25 11.	educed [%] 	Dur 	ation no N 90914 100 36 91051 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 88.907 106.577 108.854 88.231 104.925	Fux 1 19 0 080 60 0 0 59 - - 1 1 1 1 1 1 1 1 1 1 1 1 1	Duration r	To Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi BSERVER / Obs 001 002 003 004 006 007 008 010 011 013 014 015 016 017 018 019 019	equential [s 1358591 TURE STATIS1 Ture starts Tiecture # Tmote Wismote 0penMote ACM2 CC430 TOTAL ARCHITECTUR TOTAL 01.082 90.117 92.278 93.072 80.936 83.648 95.563 93.673 83.041 89.627 1 94.962 87.110 91.177 1 94.962 1 87.712 1 88.777] Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223 E USAGE Tinyl - E USAGE Tinyl 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0	Paralle 109 # Tests 76. 2. 0. 4. 17. 100. (TIME): Wode .000 .175 .175 .175 .175 .000 .175 .175 .000 .175 .175 .175 .000 .175 .175 .175 .000 .175 .167 .175 .1	D1 [s] T D77309 D77309 [%] Dura 146 0771 111 583 804 360 000 Wismote 0.0000 0.00000 0.0000 0.00000 0.00000 0.000000 0.00000 0.000000 0.00000000	ime Reduced 2608 2608 11460394 129000 10080 389060 1633620 17640 136339794 13639794 13639794 0.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000000 0.00000000	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086 1 11.12 1 12.59 1 11.23 1 12.14 1 12.59 1 11.23 1 12.14 0.00 1 11.72 0.00 0 0.00 0 0.00 0 11.01 0 0.00 0 11.01 0 0.00	educed [%] 	Dur 	ation no N 90914 100 36 91051 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 84.208 84.208 84.208 84.201 106.577 106.577 108.854 88.231 104.925 89.945	fux 1 -19 0 80 66 0 0 0 59 - - 	Duration r	no Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi BSERVER / Obs 001 002 003 004 006 007 008 010 011 013 014 015 016 017 018 019 020	equential [s 1358591 TURE STATIST Ture Tmote TinyNode Wismote OpenMote ACM2 CC430 TOTAL /ARCHITECTUR 91.082 91.082 91.082 91.082 92.278 93.673 83.648 95.563 93.673 83.641 89.627 87.723 94.791 94.962 87.110 91.177 88.777 91.957] Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223 E USAGE Tinyl 00 1 00 1 00 1 00 1 1 1 00 1 1 1 00 1 1 1 0 1 0	Paralle 109 # Tests 776. 2. 0. 4. 177. 0. 100. (TIME): Node .000 .000 .000 .000 .000 .000 .000 .175 .167 .175 .000 .175 .000 .175 .167 .175 .000 .175 .167 .175 .000 .175 .167 .175 .000 .000 .175 .167 .175 .000 .000 .175 .167 .175 .000 .000 .000 .175 .167 .175 .000 .000 .000 .000 .175 .167 .175 .000 .000 .000 .000 .000 .000 .175 .167 .175 .000	<pre>b1 [s] T; p77309 p77309 p773</pre>	ime Reduced 2608 2608 11460394 129000 10080 389060 1633620 17640 13639794 13639794 13639794 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000 0.00000000	[s] Time R 605 	educed [%] 	Dur 	ation no N 90914 100 36 91051 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 88.907 106.577 108.854 88.231 104.925 89.945 89.945	fux 1 19 0 560 0 559 	Duration r	no Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi T BSERVER/ Obs BSERVER/ Obs 001 003 004 003 004 007 008 007 008 007 008 001 003 004 007 008 0007 008 008	equential [s 1358591 TURE STATISI Tintecture # Tmote TinyNode Wismote OpenMote ACM2 CC430 TOTAL /ARCHITECTUR 91.082 90.117 92.278 93.072 80.936 83.648 95.563 93.673 83.641 89.627 87.723 94.962 87.710 91.177 88.777 91.177 91.177 91.177 91.177] Time 4 ICS: Tests 4 5500 150 8 331 1266 26 7223 FUSAGE Tinyl 0. 1 0. 1 0. 1 0. 1 1. 1 1. 1 1. 1 1. 1	Paralle 109 # Tests 76. 2. 0. 4. 17. 0. 100. (TIME): Node	D1 [s] T D77309 D77309 D77309 D77309 D77309 D111 D12 D146 0771 1111 583 804 360 000 0000 0000 0.0000 0.0000 0.000 0.000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000	ime Reduced 2608 2608 11460394 129000 10080 1633620 1633620 17640 13639794 13639794 13639794 0.000 0.0000 0.0885 0.0000 0.0885 0.0000 0.059 0.0000 0.0100 0.0000 0.0100 0.0000 0.0110 0.0000 0.0000 0.0112 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000000	[s] Time R 605 # no Mux 3071 0 8 71 0 3086 1 11.15 1 12.24 1 2.59 1 11.22 1 2.59 1 11.23 1 2.14 0.00 1 11.77 0.00 0.	educed [%] 	Dur 	ation no N 90914 100 36 91051 TOTAL 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 88.907 106.577 106.854 88.231 104.925 89.945	fux 1 19 0 360 00 559 1 1 1 1 1 1 1 1 1 1 1 1 1	Duration r	To Mux [79.3 0.0 100.0 0.9 0.0 66.7
Time Se RCHITECT Archi T C BSERVER/ Obs 001 002 003 004 006 007 008 007 008 007 008 007 008 001 003 004 006 007 008 007	equential [s 1358591 TURE STATIST TURE STATIST Tiecture # TinyNode Wismote OpenMote ACM2 CC430 TOTAL TOTAL TOTAL ARCHITECTUR 91.082 91.082 91.082 93.072 80.936 83.648 95.563 93.673 83.041 89.627 87.723 83.041 189.627 87.723 83.041 189.627 87.723 83.041 189.627 187.723 87.723 187.723 187.723 187.723 187.723 187.723 187.723 187.723 187.723 187.723 187.723 187.723 187.723 187.723 187.723 187.723 187.723 194.962 197.710 191.777 191.957 195.262 195.263 194.791 195.263 194.791 195.263 195.263 195.263 195.263 195.263 195.263 195.263 195.2777 195.2777 195.2777 195.2777 195.27777 1] Time 4 ICS: Tests 4 5500 150 8 331 1286 26 7223 E USAGE Tiny] E USAGE Tiny] 0.0 1.0 0.1 0.1 1.1 1.1 1.1 1.1	Paralle 109 * Tests 76. 2. 0. 4. 17. 0. 100. (TIME): Node .000 .000 .000 .000 .000 .000 .175 .175 .175 .175 .175 .175 .000 .000 .175 .175 .175 .000 .000 .175 .175 .175 .000 .000 .000 .000 .175 .175 .167 .000 .000 .000 .000 .000 .000 .175 .167 .100 .000 .000 .000 .000 .175 .167 .175 .000 .000 .000 .000 .000 .000 .000 .000 .175 .175 .000 .000 .000 .000 .000 .175 .167 .100 .000 .000 .000 .000 .000 .175 .167 .100 .000 .000 .000 .000 .000 .000 .175 .167 .100 .000	D1 [s] T D77309 D77309 146 0771 111 583 804 360 0000 Wismote 0.0000 0.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000000	ime Reduced 2608 2608 11460394 129000 10080 1633620 17640 1389060 1633620 17640 13639794 13639794 13639794 13639794 0.000 0.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000000	[s] Time R 605 # no Mux 3071 0 8 71 0 0 3086 1 11.125 1 11.22 1 2.14 1 12.59 1 11.23 1 12.14 0 0.000 1 10.000 1 10.000 1 10.77 0 0.000 1 10.77 0 0.000 1 0.0000 1 0.00000	educed [%] 	Dur Dur 	ation no N 90994 100 36 91051 102.371 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 88.907 106.577 108.854 88.231 104.925 89.945 98.947 98.947 93.214	fux 1 19 0 880 660 0 0 59 - - 1 1 1 1 1 1 1 1 1 1 1 1 1	Duration r	To Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi I BSERVER/ Obs I 001 002 003 004 006 007 008 001 002 003 004 006 10 007 008 001 011 013 014 015 016 017 018 011 013 014 015 016 017 018 019 020 020 022 023 024 025 025 025 025 025 025 025 025	equential [s 1358591 TURE STATIS1 TURE STATIS1 Tiecture # Tmote Vismote 0penMote ACM2 CC430 TOTAL TOTAL ARCHITECTUR 1 Tmote 91.082 90.117 1 92.278 93.072 80.936 83.648 93.673 83.041 83.648 93.673 1 83.648 93.673 1 83.648 93.673 1 83.648 93.673 1 83.648 93.673 1 83.777 1 94.962 87.110 1 94.962 87.712 87.723 1 94.962 1 87.771 94.962 1 87.772 1 94.962 1 87.771 1 94.962 1 87.772 1 95.772 1 96.126 1 89.872 1 86.611 1 97.725 1 96.6126 1 97.725 1 97.755 1 97.7555 1 97.755 1 97.755 1 97.755 1 97.7555 1 97.7555] Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223 E USAGE Tiny) E USAGE Tiny) 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0	Paralle 109 # Tests 76.2. 0. 4. 17. 0. 100. (TIME): 000 000 000 000 000 175 175 175 175 175 167 175 167 167 000 000 167 000 167 175	D1 [s] T D77309 D77309 [%] Dura 146 077 111 583 804 360 000 Wismote 0.0000 0.00000 0.00000 0.00000 0.0000000 0.000000 0.00000000	ime Reduced 2608 2608 11460394 129000 10080 1633620 17640 13633794 13633794 13633794 13633794 13633794 0.0000 0.0000 0.0000 0.000 0.000	[s] Time R 605 # no Mux 3071 0 8 7 0 3071 0 8 7 0 0 3086	educed [%] 	Dur 	ation no N 90914 100 36 91051 102.371 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 88.907 106.577 108.854 88.231 104.925 89.945 92.049 98.947 93.214	fux I -19 0 80 56 0 0 0 55 - - 	Duration r	no Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi T C BSERVER/ Obs 001 002 003 004 006 007 008 001 002 003 004 006 007 008 010 011 013 014 015 016 017 018 019 019 020 022 023 024 025 026	equential [s 1358591 TURE STATISI Ture [Tiecture] # Tmote] DpenMote] ACM2] CC430] TOTAL] /ARCHITECTUR TOTAL] /ARCHITECTUR 10000000000000000000000000000000000] Time 4 TCS: Tests 4 5500 150 8 331 1286 26 7223 E USAGE Tinyl - CON 0 0 1 00 1 00 1 00 1 00 1 1 1 1 1 1 1 1	Paralle 109 Tests 76. 2. 0. 4. 17. 0. 100. (TIME): 000 000 000 000 000 000 175 175 175 000 175 175 000 12 000 12 000 12 000 12 000 12 000 175 1	<pre>31 [s] Ti 377309 377309 (%] Durr 146 077 111 583 840 360 0000 Wismote Wismote 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.000</pre>	ime Reduced 2608 2608 11460394 129000 10080 389060 1633620 17640 13633794 13639794 13639794 13639794 13639794 0.000 1.0.000 0.000 1.0.0000 1.0.0000 1.0.000 1.0.000	[s] Time R 605 # no Mux 3071 0 8 7 0 0 3086 1 11.15 1 11.22 1 2.58 1 11.23 1 12.24 0.000 1 11.72 1 0.000 0.11.72 0.000 1 1.000 0.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000 0.00000000	educed [%] 	Dur 	ation no M 90914 100 36 91051 TOTAL 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 88.907 106.577 108.854 88.231 104.925 89.945 89.947 93.214 89.841 88.636 91.625	fux 1 19 0 560 0 59 - - 1 1 1 1 1 1 1 1 1 1 1 1 1	Duration r	no Mux [79.3 0.0 100.0 0.9 0.0 0.0 66.7
Time Se RCHITECT Archi T C BSERVER/ C BSERVER/ C C C C C C C C C C C C C	equential [s] Time 4 ICS: Tests 4 5500 150 8 331 1286 26 7223 E USAGE Tinyl E USAGE Tinyl 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0	Paralle 109 # Tests 776. 2. 0. 4. 177. 0. 100. (TIME): Node .000 .000 .000 .000 .000 .000 .000 .175 .167 .175 .175 .000 .000 .000 .000 .000 .000 .175 .175 .000 .000 .000 .000 .175 .175 .000 .000 .000 .000 .000 .175 .175 .000 .000 .000 .175 .000 .000 .000 .000 .175 .000 .000 .000 .000 .175 .000 .000 .000 .000 .000 .000 .175 .000 .000 .000 .000 .000 .000 .000 .175 .167 .175 .000 .000 .000 .000 .000 .000 .000 .175 .175 .000 .000 .000 .000 .000 .000 .000 .175 .167 .175 .000 .175 .175 .175 .175 .000 .175 .175 .000 .000 .000 .000 .000 .175 .000 .000 .000 .000 .000 .000 .000 .000 .175 .000	<pre>b1 [s] T; p77309 p77309 p773</pre>	ime Reduced 2608 2608 11460394 129000 10080 1633620 1633620 17640 13639794 13639794 13639794 0.000 0.0.0000 0.0.000 0.0.0000 0.0.0000 0.0.0000 0.0.0000 0.0.0000 0.0.0000 0.0.0000 0.0.0000 0.0.0000 0.0.0000 0.0.0000 0.0.0000 0.0.	[s] Time R 605 # no Mux 3071 0 8 71 0 0 3086 1 1.15 1 1.22 1 12.59 1 11.23 1 12.14 0.00 1 11.72 0.0000 0.000 0.000 0.000 0.000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000000	educed [%] 	Dur I Dur I I <tr< td=""><td>ation no N 90914 100 36 91051 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 88.907 106.577 108.854 88.231 104.925 89.945 89.945 89.947 98.947 98.941 88.636 91.626 91.626</td><td>fux 1 19 0 560 0 59 </td><td>Duration r</td><td>10 Mux [79.3 0.0 100.0 0.9 0.0 0.0</td></tr<>	ation no N 90914 100 36 91051 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 88.907 106.577 108.854 88.231 104.925 89.945 89.945 89.947 98.947 98.941 88.636 91.626 91.626	fux 1 19 0 560 0 59 	Duration r	10 Mux [79.3 0.0 100.0 0.9 0.0 0.0
Time Se RCHITECT Archi T C ESERVER/ C C C C C C C C C C C C C	equential [s] Time 4 ICS: 5500 150 8 331 1266 26 7223 EUSAGE I Tinyl 0.0 0.1 0.1 0.1 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.1 1.1 1.00 1.1 1.1 1.00 1.1 1.00 1.1 1.00 1.1 1.00 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1	Paralle 109 # Tests 76. 2. 0. 4. 17. 0. 100. (TIME): Node	<pre>b1 [s] T; p77309 p77309 F(1) Dura 146 0771 111 583 804 360 .0000 .0000 0.0000 0.0000 0.0000 0.000 0.0000 0.0000 0.0000 0.0000 0.</pre>	ime Reduced 2608 2608 11460394 129000 10080 1633620 1633620 17640 13639794 13639794 13639794 13639794 13639794 0.000 0.0000 0.0000 0.0885 0.0000 0.0885 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.2974 1.2573 0.0000 0.0000 0.0000 0.0000 0.2974 1.3.341 1.3	[s] Time R 605 # no Mux 3071 0 8 71 0 3086 1 11.15 1 12.14 1 12.15 1 12.259 1 11.22 1 2.59 1 11.23 1 12.14 0.000 1 11.72 0.000 1 11.02 1 11.01 0.0000 0.00000 0.0000 0.0000 0.00000 0.00000	educed [%] 	Dur Dur 	ation no M 90914 100 36 91051 TOTAL 102.371 101.490 105.760 104.456 93.842 84.832 108.346 94.848 84.208 90.802 84.907 106.577 108.854 84.231 104.925 89.945 92.049 98.947 93.214 88.636 91.626 107.2910	fux 1 19 0 360 00 	Duration r	To Mux [79.3 0.0 100.0 0.9 0.0 66.7

200	95.885 96.013 18.801	0.000	0.092	0.739 0.000 0.000	1.213 11.072 12.674 0.000	I 0.000 I 0.000 I 0.048 I 0.000	97.032 107.049 108.826 19.961	
201 202	20.352	1.152	0.000	0.000	0.000	i 0.000	21.504 21.027	
204	20.592	1.159	0.000	0.000	0.000			
TUTAL	104.401	1.175	0.092	3.544	14.882	0.161		
BSERVER / A	ARCHITECTURE	USAGE (# TE	STS):					
Ubs 	Tmote	TinyNode	Wismote	UpenMote	A C M 2		TUTAL 	
001	64.239	0.000	0.000	0.000	9.248	0.291		
002	62.966	0.000	0.000	0.914	9.442	0.346	1 76.603	
004	66.247	0.000	0.000	0.000	9.511	0.332	76.090	
006 I	65.042	0.000	0.000	0.858	11.283	0.000	77.184	
007 I	53.067 I	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	60 681	
014	57 095	2.077	0.000		0.000	0.000	1 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	11 70.262	
019	56.818	2.063	0.000	0.000	0.000	0.000	11 58.881	
020	62.353 62.012	0.000	0.111	3 544	0.000	0.000	11 00.473 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	I 0.000 I	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	0 761	
029	67 894	0.000		0.000	2 810		1 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	
202		0 0 0 0	0 000	0.000	0.000	0.000	26.333	
202	24.284	2.049	0.000				1	
202 204 	24.284 76.146	2.049	0.111	4.583	17.804	0.360		
202 204 TOTAL 	24.284 76.146 umber of Par	2.049 2.077	0.111	4.583	17.804	0.360	 	
1 202 204 TOTAL 1 TOTAL 1 Aximal Nu 1 2 2016 2 2016 2 2 16 2 2 16 2 1 16 2 1	24.284 76.146 umber of Par ############### scheduled 1183	2.049 2.077 callel tests: ###################################	0.000 0.111 8 *********************************	4.583 *****************	17.804 ***********			* * * * * * * * * * * * *
1 202 204 1 TOTAL 1 TOTAL 1 Aximal Nu 1 2 2016 1 2 2016 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	24.284 76.146 umber of Par ####################################	2.049 2.077 callel tests: ###################################	0.000 0.111 8 *********************************		17.804			* * * * *
202 204 TOTAL 1aximal Nu ########## 2016 # ############# 1333 Architec	24.284 76.146 umber of Par ####################################	2.049 2.077 callel tests: ###################################	0.000 0.111 8 *********************************	<pre>4.583 ####################################</pre>	17.804	0.360		* # # # # # # # # # # # #
2024 2044 TOTAL 1aximal Nu ########## 2016 2016 2016 2016 2016 2016 2016 2016	24.284 76.146 ####################################	2.049 2.077 callel tests: ###################################	0.000 0.111 8 *********************************	<pre> 4.583 4.583 ################################</pre>	17.804] 0.360		* * * * * * * * * * * * *
2024 204 TOTAL faximal Nu ########## 2016 ########### ENERAL: # Tests 1333 	24.284 76.146 umber of Par ############## scheduled 1183 cture Tota Cture Yota Tmote 94 CC430 18	2.049 2.077 callel tests: ####################################	0.000 0.111 8 ************** 1.[%] 3.747 9d schedu 66 20	<pre> 4.583 ################################</pre>	17.804	0.360		* * * * * * * * * * * * * * * * * * * *
<pre>1 202 204 1 204 1 201AL 1 2016 1 2016</pre>	24.284 76.146 umber of Par ############## scheduled 1183 cture Tota Fmote 94 Cd430 18 Opal 7	2.049 2.077 2.077 2.101 tests: 4############## 4 scheduled 3 88 4 86 4 86 5 12 4 12 4 12 4 12 5 12	0	<pre> 4.583 ################################</pre>	17.804	0.360		* * * * * * * * * * * * * * * * * * * *
2024 2044 TOTAL faximal Nu ########## 2016 ########### 2016 ############ BENERAL: # Tests 1333 	24.284 76.146 umber of Par ############### scheduled 1183 cture Tota Tmote 94 Cota30 18 Opal 7 yNode 4 dpp 7	2.049 2.077 callel tests: ####################################	0 .000 0 .111 0 .1111 0 .1111 0 .1111 0 .1111 0 .1111 0 .1111	<pre> 4.583 ################################</pre>	17.804	0.360		
202 204 TOTAL 1aximal Nu ########## 2016 2ENERAL: 1333 Architec Tiny	24.284 76.146 umber of Par ####################################	2.049 2.077 2.077 callel tests: ************************************	0 0.000 0 0.111 0 0.1111 0 0.11111 0 0.1111 0 0.1111 0 0.11111 0 0.11111 0 0.1111 0 0.11111 0 0.111111 0 0.111111 0 0.111111 0 0.111111 0 0.111111 0 0.11111111111 0 0.11111111111111111111111111111111111	<pre> 4.583 ###################################</pre>	17.804] 0.360		
1 202 204 TOTAL 1 aximal Nu 1 axim	24.284 76.146 umber of Par ####################################	2.049 2.077 callel tests: ####################################	0	4.583 ####################################	17.804 ********** ***********	0.360		
2024 2044 TOTAL 1aximal Nu 1######### 2016 2016 2016 2016 2016 2016 2016 2016	24.284 76.146 umber of Par ####################################	2.049 2.077 callel tests: ####################################	0 .000 0 .111 8 ********************************	4.583 ################# ################# led [%] led [%] 100.000 64.865 94.595 95.918 0.000 ime Reduced [s] 25174	17.804 ********** ***********	Uced [X] 10.382		
<pre>4 202 204 7 204 4 aximal Nu ########### 2016 2 2016 2 /pre>	24.284 76.146 umber of Par ####################################	2.049 2.077 callel tests: ####################################	0 0.000 0 0.111 8 ********************************	4.583 ####################################	17.804 ############ ############ Time Red	0.360		
2024 2044 TOTAL 4aximal Nu ######### 2016 ########## BNERAL: # Tests 1333 Architec Time Sec	24.284 76.146 umber of Par ############## scheduled 1183 cture Tota Cd30 18 Opal 7 Node 4 dpp 7 quential [s] 2424814 JRE STATISTI tecture # 1	2.049 2.077 2.077 callel tests: ####################################	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	<pre> 4.583 ###################################</pre>	17.804 ####################################	Uced [%] 10.360	Duration no Mux	: Duration no Mux [%]
2024 2044 707AL 4aximal Nu ######### 2016 # Tests 1333 Architec Tiny Time Sec RCHITECTU	24.284 76.146 umber of Par ############## scheduled 1183 cture Tota fmote 94 Cd430 18 Opal 7 Node 4 dpp 7 quential [s] 2424814 JRE STATISTI tecture # T Tmote	2.049 2.077 2.077 callel tests: ####################################	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	4.583 ####################################	17.804 ####################################	Uced [X] 10.360 11.10 11.10 10.382		<pre>##### # # ###########################</pre>
2024 2044 TOTAL 4aximal Nu ########## 2016 2016 2016 2016 2016 2016 2016 2016	24.284 76.146 umber of Par ############### scheduled 1183 cture Tota cture Tota Cture 70ta Cd30 18 Ogal 7 VN de 4 dpp 7 quential [s] 2424814 JRE STATISTI tecture T Tmote CC430	2.049 2.077 2.077 callel tests: ####################################	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	4.583 ####################################	17.804 ####################################	U.ced [%] u.ced [%] 10.382 10.382 no Mux [%] 39.323 0.000		<pre>##### # # # # # # # # # # # # # # # #</pre>

DBSERVER/					
	ARCHITECTURE U	JSAGE (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
007	90.533	0.456	2.770	1.878	95.638
008 1	96.041	0.563	0.000	0.000 11	96.604
010	94.664	0.456	0.000 l	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		99.483
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 l	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
023 1	94.815 94.920	3.001 1.963	2 770	0.000 11	90.4/D 99 652
024	94.787	0.584	2.770	0.000 11	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 11	1.281
032	94.780	0.563	0.000	0.000 11	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 BSERVER/	100.655 	6.281 JSAGE (# TEST	2.770 S):	1.878	
Obs	Tmote	CC430	Opal	TinyNode	TOTAL
0.01	70 750	1 606	0 000 1	0 000 11	70 250
001 1	70.752	1 522	5 917 1	0.000 11	78 698
003	65.004	0.000	0.000	0.000	65.004
004 I	65.173	1.606	5.917	0.000	72.697
006	56.551	2.959	0.000 l	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 1	70.161	1.208	0.000 I	3.9/3	75.402
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 l	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
010 1	67 456	0.309	5 917 1	3 973 11	78 614
010	68.808	1.268	5.917	0.000	75.993
020	69.992	5.917	0.000	0.000	75.909
020	70.245	3.635	5.917	0.000	79.797
020 022 023	70 070	1.606	5.917	0.000	77.599
020 022 023 024	10.016	1.268	5.917	3.973	79.290
020 022 023 024 025	68.132	1 0 0 0	0.000	3.973	74.472
020 022 023 024 025 026	68.132 69.231	1.268	0.000 1	0.000 11	76 754
020 022 023 024 025 026 027 028	70.076 68.132 69.231 70.414 70.837	1.268 7.270 0.000	5 917 1	0.000	4.142
020 022 023 024 025 026 027 028 029	70.076 68.132 69.231 70.414 70.837 4.142	1.268 7.270 0.000 0.000	5.917 0.000		62 722
020 022 023 024 025 026 027 028 029 031	70.076 68.132 69.231 70.414 70.837 4.142 61.200	1.268 7.270 0.000 0.000 1.522	5.917 0.000 0.000	0.000	02.122
020 022 023 024 025 026 027 028 029 031 032	68.132 69.231 70.414 70.837 4.142 61.200 69.484	1.268 7.270 0.000 0.000 1.522 1.522	5.917 0.000 0.000 0.000	0.000 0.000	71.006
020 023 023 024 025 026 027 028 028 029 031 032 033	68.132 69.231 70.414 70.837 4.142 61.200 69.484 72.105	1.268 7.270 0.000 1.522 1.522 1.522	5.917 0.000 0.000 0.000 0.000	0.000 0.000 0.000	71.006
020 023 023 024 025 026 027 028 028 029 031 032 033 200 021 033	68.132 69.231 70.414 70.837 4.142 61.200 69.484 72.105 20.626	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	5.917 0.000 0.000 0.000 0.000 5.748	0.000 0.000 0.000 3.888	71.006 73.626 38.292
0220 0223 024 025 026 027 028 029 031 033 200 201 201	70.076 68.132 69.231 70.414 70.837 4.142 61.200 69.484 72.105 20.626 20.879 18.090	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	5.917 0.000 0.000 0.000 5.748 5.917 4.565	0.000 0.000 0.000 3.888 3.973	71.006 73.626 38.292 32.291
022 023 024 025 026 027 028 029 031 033 200 201 202 202	70.076 68.132 69.231 70.414 70.837 4.142 61.200 69.484 72.105 20.626 20.879 18.090 21.640	1.268 7.270 0.000 1.522 1.522 1.522 8.030 1.522 3.973 4.903	$\begin{array}{c ccccc} 5.917 & \\ 0.000 & \\ 0.000 & \\ 0.000 & \\ 5.748 & \\ 5.917 & \\ 4.565 & \\ 5.917 & \\ 5.917 & \end{array}$	0.000 0.000 3.888 3.973 3.297 3.973	71.006 73.626 38.292 32.291 29.924 36.433
0 2 0 0 2 2 0 2 3 0 2 4 0 2 5 0 2 6 0 2 7 0 2 8 0 2 7 0 2 8 0 2 9 0 3 1 0 3 2 0 3 3 0 3 2 0 3 3 0 3 3 0 2 0 1 0 2 0 1 0 2 5 0 2 7 0 2 8 0 2 9 0 3 1 0 2 9 0 3 1 0 2 9 0 3 1 0 2 7 0 2 8 0 2 9 0 3 1 0 2 9 0 3 1 0 2 9 0 3 1 0 2 9 0 3 1 0 3 2 0 4 0 3 1 0 3 2 0 4 0 3 1 0 3 2 0 2 1 0 3 2 0 4 0 3 1 0 3 2 0 2 1 0 3 2 0 2 1 0 3 2 0 2 1 0 3 2 0 2 1 0 3 2 0 2 1 0 3 2 0 2 1 0 3 2 0 2 1 0 3 2 0 2 1 0 3 2 0 2 1 0 3 2 0 2 0 0 0	70.076 68.132 69.231 70.414 70.837 4.142 61.200 69.484 72.105 20.626 20.879 18.090 21.640	$\begin{array}{c ccccc} 1 & .268 \\ 7 & .270 \\ 0 & .000 \\ 1 & .522 \\ 1 & .522 \\ 1 & .522 \\ 1 & .522 \\ 1 & .522 \\ 1 & .522 \\ 3 & .030 \\ 1 & .522 \\ 3 & .973 \\ 4 & .903 \\ \end{array}$	$\begin{array}{c} 5.917 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 5.748 \\ 5.917 \\ 4.565 \\ 5.917 \\ 1 \end{array}$	0.000 0.000 3.888 3.973 3.297 3.973	71.006 73.626 38.292 32.291 29.924 36.433
0220 0230 0240 0250 0260 0270 0280 0290 0310 0330 2000 2010 2020 2020 2020 707AL	70.076 68.132 69.231 70.414 70.837 4.142 61.200 69.484 72.105 20.626 20.879 18.090 21.640	1.268 7.270 0.000 1.522 1.522 1.522 1.522 8.030 1.522 3.973 4.903 10.144	5.917 0.000 0.000 0.000 5.748 5.917 4.565 5.917 5.917	0.000 0.000 3.888 3.973 3.297 3.973 3.973	71.006 73.626 38.292 32.291 29.924 36.433
0 2 0 0 2 2 0 2 3 0 2 4 0 2 5 0 2 6 0 2 7 0 2 8 0 2 9 0 3 1 0 3 2 0 3 3 2 0 0 2 0 1 2 0 2 1 2 0 4 1 0 2 5 0 2 8 0 2 9 0 3 1 0 3 2 0 3 3 2 0 0 2 0 1 2 0 2 1 0 2 6 0 2 7 0 2 9 0 3 1 0 3 2 0 4 0 4 0 7 0 4 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7	70.076 68.132 70.414 70.837 4.142 61.200 69.484 72.105 20.626 20.879 18.090 21.640 79.966	1.268 7.270 0.000 1.522	5.917 0.000 0.000 0.000 5.748 5.917 4.565 5.917 5.917	0.000 0.000 3.888 3.973 3.973 3.973 3.973	71.006 73.626 38.292 32.291 29.924 36.433