WalT: A Reproducible Testbed for Reproducible Network Experiments

Pierre Brunisholz, Etienne Duble, Franck Rousseau, Andrzej Duda

WalT: a Reproducible Testbed for Reproducible Network Experiments

Pierre Brunisholz, Etienne Dublé, Franck Rousseau, Andrzej Duda
Grenoble Alps University
Grenoble Institute of Technology, CNRS Grenoble Informatics Laboratory, Grenoble, France
Email: firstname.lastname@imag.fr

***Abstract*—Testing wireless networks is a challenging task, so many research papers limit their evaluation section to simulation. At the same time, several large-scale publicly shared testbeds such as ORBIT, w-iLab.t, or IoT-LAB propose valuable means for validating new protocols. They make experimenting possible and support some level of experiment reproducibility. However, they offer operating conditions that are fixed and significantly differ from real-world deployments. As the performance of wireless networks strongly depends on the platform, topology, and operating conditions, validating protocols in artificial conditions may not lead to meaningful results. In this paper, we describe WalT, a reproducible platform to run reproducible experiments. WalT nodes are single-board computers on which users can deploy their OS (filesystem, kernel) packaged as a *docker* image for easy customization and sharing. With low-cost small-sized standard components and free software, researchers can easily reproduce their own WalT platform to validate results in real-world conditions. WalT can support mobile demos that you can bring around in your backpack. The total control on WalT nodes allows setting up diverse experiment scenarios ranging, for instance, from Wi-Fi handover measurements to evaluating routing protocols in wireless sensor networks.**

I. Introduction

Validating new protocols for wireless networks is a challenging task for which we can mainly use simulations or perform experiments. Simulations are quite limited and can only give us the first insight into the behavior of a chosen protocol under simplified assumptions [1], [2]. Usually, simulations do not take the underlying topology into account sufficiently well so the validation may lead to interesting results and comparisons at first, but eventually, the results may be different in real deployments.

Performing real-world experiments is considered as the prerequisite for a true validation of a protocol proposal and it requires both *repeatability* and *reproducibility* [3]. *Repeatability* requires that the same experiment is run in the same conditions several times by the investigator, producing similar results, and it is necessary to guarantee that experimental results are sound. *Reproducibility* is guaranteed if an experiment can be replicated under differing conditions, while providing sufficiently similar results. This is essential to prove that the scientific proposal is robust across various environments and not only in a particular setup. It is also crucial for allowing researchers to reproduce experiments, build upon, and compare their results with the previous work.

Experiments involving wireless networks are difficult to run. We have a choice between using specialized platforms (like ORBIT [4], w-iLab.t [5], IoT-LAB [6], and others) or setting up some ad hoc testbeds. Specialized platforms make the experimentation task easier and offer good support for experiment repeatability. However, they only offer operating conditions that are far away from real-world deployments: nodes are usually distributed on a regular grid in one large room, and reproducibility in varying environmental conditions is very limited. As the performance of wireless networks strongly depends on their topology, environment, and operating conditions, validating protocols in artificial conditions may not lead to meaningful results, and reproducibility is de facto limited. Moreover, researchers do not have physical access to the equipment so debugging is tedious and some measurements like fine-grain energy consumption may be impossible. Ad hoc testbeds can help validating a given protocol, but they do not scale and it is difficult to reproduce results since most of the time a precise specification is missing.

In this paper, we propose WalT, a reproducible platform to run reproducible experiments. Our goal is threefold: (i) easily setup a platform to develop, debug and make preliminary validation, (ii) allow deployment and control of experiments on a larger scale, (iii) provide a way for others to deploy exactly the same experiment in a different environment, to challenge reproducibility and also guarantee a suitable starting point and adequate repeatability when comparing with other proposals. It follows an approach that lies somewhere in between specialized platforms with rigid and limited topology, and ad hoc testbeds. WalT nodes are single-board computers on which users can deploy their OS (filesystem, kernel) packaged as a *docker* image for easy customization and sharing.

A WalT platform provides:

- *full remote control over nodes*: rebooting, remote shell sessions, deploying OS images,
- *management of node OS images*: clone from the docker hub, modify locally, and publish images,
- *log management*: means to collect, store, and query experiment logs and event traces are provided,
- *automated discovery* of the platform topology.

WalT presents the following advantages:

- it can be *easily replicated* at the required deployment places,
- it is *cost-effective*, built from inexpensive components run-

ning open source software[1],
– it is *lightweight*, it can serve for a portable demo or as a development platform in an office without a tedious deployment in a whole building,
– it is *adaptable*, you can use it for sensor networks or experiments with IEEE 802.11,
– it is *easy to install and use*.

Besides supporting reproducible experiments, WalT also enables the emergence of reproducible platforms—researchers can set up their WalT platforms to validate the results of others at different places or environments.

The rest of the paper discusses the initiatives for wireless network experiments (Section II), describes the WalT architecture (Section III), presents some example experiments with WalT: a comparison of NTP/PTP temporal synchronisation and measurements of the 802.11 handover of smartphones (Section IV), and proposes a demo for the workshop (Section V).

II. Testbeds for Wireless Network Experiments

The wireless medium presents complex characteristics as it is sensitive to many external factors and varying conditions difficult to model. Thus, the entire protocol stack must be robust with respect to them and should be evaluated in realistic situations that involve all factors that may influence robustness. Moreover, research on wireless networks requires a wide area of expertise since many aspects of the protocol stack up to applications rely on specific functionalities: the physical layer, medium access control, neighbor maintenance and discovery, routing, just to name a few. Recent research activities in wireless sensor networks have redefined an entire body of standards to meet even tighter constraints found in the context of the *Internet of Things (IoT)*: low power, scarce ressources (energy, processing power, and memory), and lossy environments. IEEE 802.15.4 at the physical and link layers, IETF 6LoWPAN at the network layer, RPL for routing, and CoAP at the application layer are some of the resulting standards for which many aspects still need to be studied.

A. Wireless Network Experimentation

Once it gets to implementation, there are two main ways to validate network and protocol proposals: simulation and experiments. For obvious reasons, simulation, with or without emulation, has been one of the preferred ways to validate protocols at the large scale: easy development, cost, repeatability, controlled environments to name a few advantages. In this case, the validity of the results depends a lot on the accuracy of the model of the entire system to study. This is where simulation finds its breaking point: the electromagnetic propagation in a real environment is almost impossible to simulate accurately since models are complex and computation intensive, and there is a huge variety of environments to take into account according to planned deployment scenarios.

[1]All WalT resources are publicly available: http://walt.forge.imag.fr, https://github.com/drakkar-lig, https://hub.docker.com/u/waltplatform/

Also, when studying wireless protocols, implementation raises many issues: a lot of work has to take place at the lower layers, and sometimes debugging and validating implementations requires to use oscilloscopes and digital analyzers. This is particularly the case for IoT-related studies in which the proposed solutions have to be implemented on highly constrained devices [7] and protocols need to be optimized to be energy efficient with tight synchronization contraints of the order of μs for example.

As we can see, there is a need for experimentations in real conditions with testbeds spanning from a small size in the development phase up to larger deployments for experiments in conditions close to the real situation of utilization.

B. Repeatability and Reproducibility

In this context, repeatability and reproducibility of experiments are two crucial challenges. First of all, to debug efficiently and validate implementations, it is of prime importance to be able to run the same test suites in exactly the same conditions. Second, to be valuable for wireless networks, scientific proposals must show their robustness under different operating conditions found in real deployments. Shared large-scale testbeds are great tools for experimenting with new protocols and algorithms, however they have some limitations: the artificial regular high density deployments for a large part of the nodes, the stable conditions in which they operate, and the limited choice of devices. In addition to such infrastructures, we need a way to deploy testbeds in real conditions and replicate these testbeds in different places to show the reproducibility of results.

We argue that validating wireless networking experiments does not only rely on showing that the same experiment repeated in exactly the same situation always gives the same results. We must also show that new protocols and mechanisms achieve good results in a wide range of real-world conditions, which means that we must be able to deploy and run experiments easily in a wide variety of situations. That is what WalT is designed for, because once a new protocol has been developed on a given topology, it can be run on any other WalT platform. As each WalT platform is deployed in a real world environment with a unique node topology, when a given experiment is reproduced, we can conclude on the robustness of the tested protocol.

Finally, we must also guarantee that our fellow researchers will be able to run state of the art experiments and compare their outcome with the results of the new proposals under the same conditions to prove their improvements through repeatability of the comparison. This is only possible if all other parameters except the tested element are similar.

WalT aims to ease this process based on two main features: (i) it relies on off-the-shelf low-cost hardware, (ii) it uses docker to build and run experiments, which allows to distribute a snapshot of the entire software environment used during the experiment. These two features offer the conditions for repeatability, since the same software will be executed for every run in exactly the same conditions, the same images

being restarted from scratch over and over in a genuine environment, as well as for reproducibility, since the same platform can be deployed in any other place, running the same experiment in a flash.

III. Architecture of WalT

A. Overview

Figure 1 presents the functional structure of a WalT platform. From the user point of view, WalT consists of a set of remotely managed nodes. They are low-cost SBCs—*Single-Board Computers* such as Raspberry Pi on which users can deploy a given operating system.

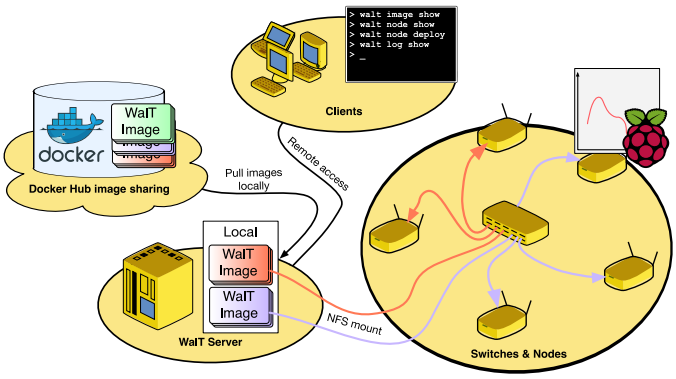

Figure 1. Functional structure of WalT

The *brain* of WalT is a GNU/Linux based server that controls the other components of the platform, interacts with the client-side software, and provides log management (collecting, storing, querying). The client-side software, which delegates all complex tasks to the server, consists of a lightweight and portable tool for exploring the topology, interacting with nodes, deploying a given operating system image on them, customizing such an image, and exploring logs.

The network is composed of cost-effective switches providing a small set of features, most notably remote management and Power-over-Ethernet (*PoE*). *PoE* simplifies deployment by providing power and data communication up to each node using the same Ethernet cable, allowing for reusing any existing cabling (e.g. in a building).

WalT internally relies on the *docker* virtualization platform. *Docker* offers efficient packaging and easy publishing of the operating systems deployed on nodes.

The ultimate goal of the WalT design was *platform reproducibility*—instead of running experiments on a publicly shared testbed, WalT proposes to reproduce the platform for any experimentation need. WalT achieves platform reproducibility because the platform is composed of low cost, standard components, free software, and it is easily deployable thanks to *PoE*. The software running on nodes is packaged into a *docker* image allowing other researchers to deploy it instantly, rerun images of other experiments in different conditions, or perform new experiments. To wrap up, the networking community can easily reproduce the WalT platform and WalT experiments.

The rest of this section will provide more details on the key components of WalT and describe our uses of WalT at the LIG laboratory.

B. WalT Images

WalT introduces the concept of a *WalT image*—it contains an operating system ready to be deployed on a WalT node. The operating system is composed of a *filesystem* and a *kernel*. A *WalT image* is internally packaged as a *docker image*, which presents two immediate benefits. First, WalT reuses docker image management features. For instance, the client command `walt image shell <image-name>` works by running a shell in a *docker container*. This allows modifying the image in a very convenient way. Adapting an image to a given experience (e.g. installing more software packages or experiment scripts) or to a given platform setup (e.g. installing a driver for a Wi-Fi device connected to the node) is just a few commands away. The second benefit is the WalT ability to use the *docker hub*, the public repository of *docker images*. Since a WalT image is actually a *docker image*, WalT images may be published and shared on the *docker hub*. The WalT client allows searching for the published images and cloning them locally.

An advanced feature of the WalT server allows *mounting* a *docker* image and making it available as a NFS (*Network File System*) share, which allows WalT nodes to boot the system contained in a WalT image. Being able to instantly turn a virtualized operating system into a real deployment is one of the most remarkable features of WalT.

WalT images contain a thin middleware layer providing tools to generate experiment logs. Logs are sent to the server for storage and processing.

C. WalT Nodes

WalT nodes are PoE-powered cost-effective SBCs. Currently, WalT supports Raspberry Pi boards. They do not support PoE so we use an external PoE splitter.

The robustness of cost-effective SBCs is an issue. Our experience has shown that their main point of failure is the SD card. We have solved this issue by keeping the SD-cards read-only. We did not observe a single node failure in more than two years of usage since this change.

The bootup procedure of WalT nodes is a two-step process. First, a node boots a read-only and minimal operating system from the SD-card. Then, the operating system stored in the target WalT image is accessed using NFS and booted. The *kexec* Linux kernel feature allows us to switch from one kernel to the other.

All filesystem changes are stored in the RAM of the node (thus discarded when the node reboots). This ensure that a WalT node associated with a given WalT image will always boot exactly the same operating system.

Having such a level of control about the operating system running on nodes makes the platform very versatile. For instance, to run experiments on routing protocols in Wireless

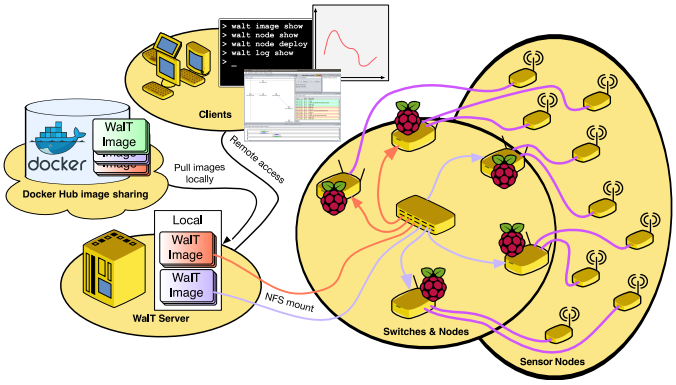

Figure 2. WalT architecture specialized for WSN

Sensor Networks, we have specialized the WalT architecture to obtain the one shown in Figure 2.

In this setup, each WalT node connects and monitors one or more Sensor Nodes. The selected WalT image contains tools to flash the sensor board and monitor sensor traces on USB-serial links.

D. WalT Client

Users interact with the platform by using a command line tool called `walt`. It provides five categories of commands: `image`, `node`, `log`, `device`, and `advanced`.

The `image` category provides commands to list WalT images, duplicate, copy, rename, remove, or modify them (using a virtualized shell), transfer files between the client machine and WalT images, search images on the docker hub, and clone an image from the docker hub to the local platform.

The `node` category allows to run a command or a shell on a node, to list nodes, transfer files between the client machine and WalT nodes, deploy an image on a set of nodes, and perform debugging tasks such as power cycling a node, making a led blink for physical identification of a node, etc.

The `log` category allows the user to explore experiment logs. They are saved in a database at the server. The user may query the historical data at any time. Another option allows the user to view the logs in pseudo real-time.

The `device` and `advanced` categories are oriented towards platform management, customization, and maintenance. For instance, one may want to rename nodes or network switches in a more user-friendly way.

In spite of this extensive platform control offered to the user, the command line tool remains lightweight and easy to install, since the server handles complexity. The tool is published on the Python Package Index (PyPI) and can be installed in just one command using the `pip` package management tool.

We have also developed a visual tool for Wireless Sensor Networks called VizWalT, implemented as a Cooja plugin. Cooja is a WSN simulator: it allows to run a simulation by emulating each sensor node. When loaded with VizWalt plugin, this behavior is modified: each node seen on the user interface reflects the behavior of a real node deployed in the WalT testbed. Figure 3 presents a screenshot of Cooja presenting WalT traces.

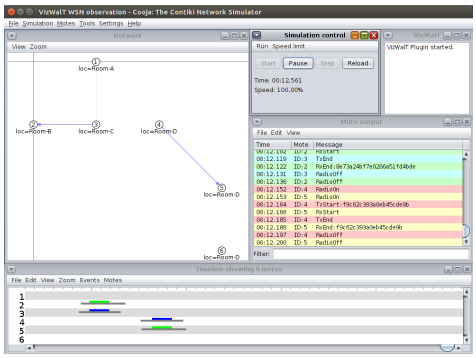

Figure 3. Screenshot of Cooja presenting WalT traces

E. WalT Experiments

One of the design goals of `walt` tool was to support the development of *experiment scripts* with any programming language the user is used to. The script can handle both the deployment and the control of an experiment. Since the complexity is handled in the WalT middleware, writing such a script is straightforward. Sharing an experiment script together with a description of the physical setup (network topology, Wi-Fi devices connected to nodes, etc.) makes the experiment reproducible, both in a similar or in a different physical environment.

We still work on the improvement of the `walt` tool with new options to ease the management of *WalT experiments*. In a near future, the tool will embed options to record, publish, and replay experiments, and to publish or download experiment results.

F. Walt Server

The WalT server controls the platform, interacts with the docker hub, the nodes, and clients. One of the main server functions is log management. The server receives logs coming from nodes, saves them in a local database, and optionally forwards them to clients for real-time viewing. When the user requests historical data, the server just queries the database.

The server also handles communication with the *docker* backend, and mounts or unmounts WalT images as NFS shares depending on the client requirements.

The server maintains the platform topology data by querying switches using the SNMP protocol. The switches use the LLDP protocol to discover their immediate neighbors.

The platform network and the external network (allowing to reach the docker hub) are isolated on dedicated VLANs to avoid any network disturbance during experiments. This network setup is fully automated.

Installing a WalT server is just a matter of minutes. A bootable image was created with `debootstick`[2] for this purpose.

[2]`debootstick` is a tool we developed to turn a filesystem tree into a bootable image. It has been part of the Debian system since July 2015.

G. WalT Platforms at LIG Laboratory

Although WalT was still in an intensive development phase a few months ago, we started working with early versions more than two years ago. As a consequence, our main deployment in the lab building has already been used for several research projects. Since WalT is also well suited to small size testbeds, we have set up a WalT-based mobile demo platform with a small number of nodes, an Intel NUC mini-PC as a server, and a dozen of STMicroelectronics 802.15.4 sensor motes. The mobile demo showed multi-hop topology creation in Cooja through the VizWalT interface at the CALIPSO final review[3]. We have also been using WalT for debugging smart object protocols with temporary on-desk setups or by simply extending the main platform in a given office (unplug one of the nodes deployed in the office, replace it with a switch, connect a few nodes to it, and run `walt device rescan` for topology update).

WalT users especially appreciate being able to use the same platform for debugging, for experiments, and for demonstrations.

IV. Example Experiments with WalT

A. Comparison of NTP/PTP Temporal Synchronisation

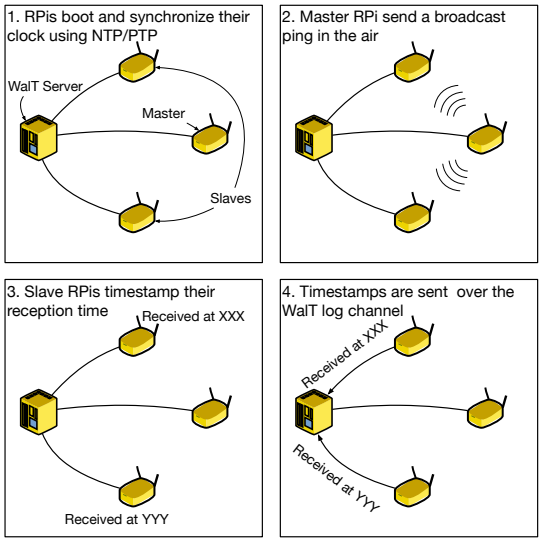

Figure 4. Clock drift measurement scenario

Synchronizing all WalT nodes is quite challenging because Raspberry Pis do not have a dedicated clock and every time a node reboots, it looses synchronization. We have decided to use the Network Time Protocol (NTP) and the Precision Time Protocol (PTP) for time synchronization. Evaluating the clock drift between nodes was a good test case to show how WalT can be used.

Figure 4 illustrates the scenario: we use three nodes with Wi-Fi dongles. We built WalT images so that they are connected to the same Wi-Fi ad hoc network. One node acts as a master and sends a broadcast ping to all slaves. They receive the message almost at the same time, which they timestamp locally. Finally the reception timestamps are logged on the server so that we can compute the clock offset.

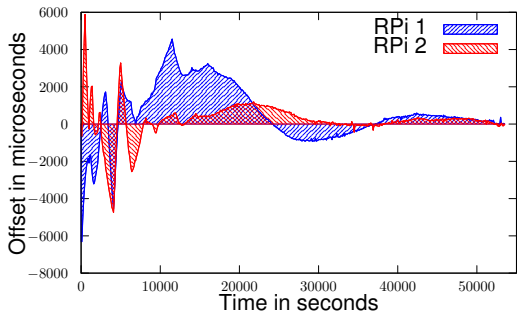

Figure 5. NTP offset variation between two nodes and the time master

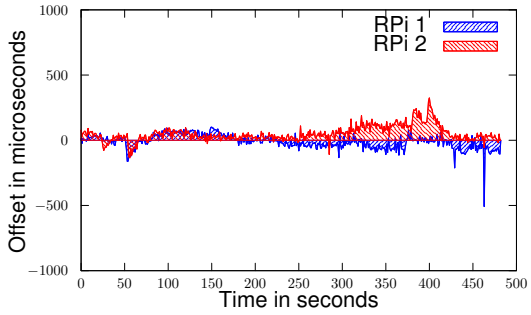

Figure 6. PTP offset variation between two nodes and the time master

WalT allows us to repeat the experiment during a long duration with three slave nodes. The results highlight that NTP offers a clock offset of tenth of milliseconds right after boot up, but precise synchronization may require a very long time as illustrated in Figure 5 —up to 13 h to achieve synchronization within 100 μs. Figure 6 shows much faster convergence for the same experiment with PTP.

B. Measuring 802.11 Handover Duration

Another example experiment was conducted with WalT to measure the duration of IEEE 802.11 handover with recent smart devices. Figure 7 presents the scenario involving two Raspberry Pi model B nodes with Wi-Fi dongles. We have built a WalT image[4] to run these nodes in AP mode —Wi-Fi access points. In the scenario, the first node starts an AP and waits for a device to connect. We consider the device connected once it is associated with the AP and they can exchange IP packets. To check IP connectivity, the device sends UDP packets periodically. After the connection is established, the first node notifies the second one to start its access point too. As soon as the second AP is up and running, we suddenly stop the first one, so that the connected device does not receive any notification about the AP going down. Then, we measure

[3]EU FP7 CALIPSO project on "Connect All IP-based Smart Objects!".

[4]walt image clone hub:brunisholz/rpi-handover-measurement

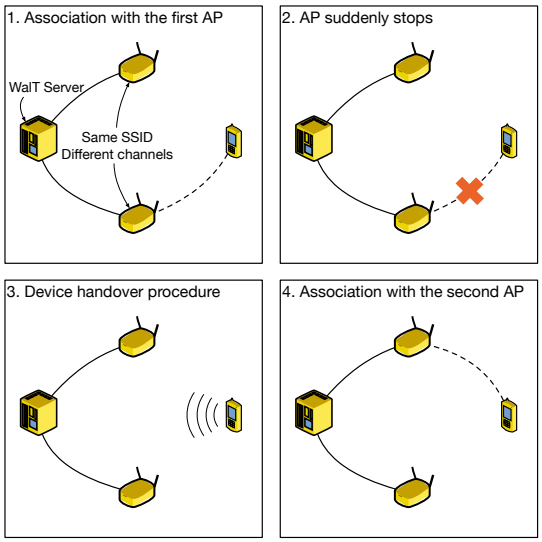

Figure 7. Wi-Fi handover measurement scenario

the time until the device restores IP connectivity through the second AP.

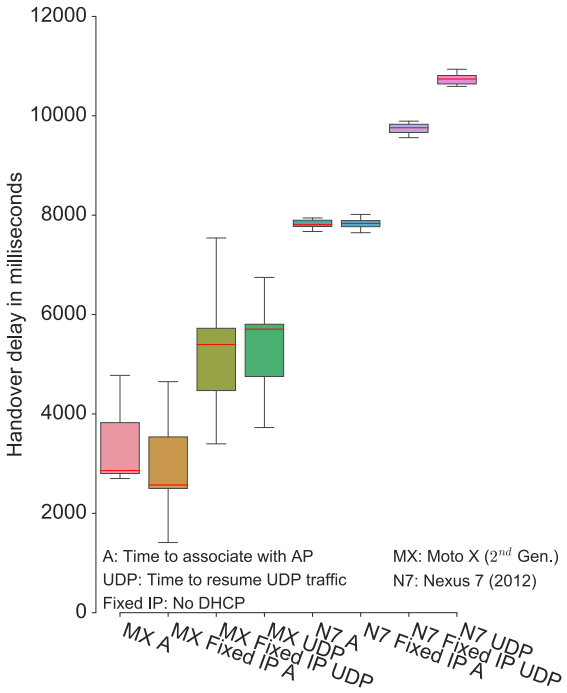

Figure 8. Measured Wi-Fi handover delay

Using WalT, we can automate the entire process and effortlessly repeat the experiment a large number of times for different devices. Here, we have collected 300 measurements for each device, a Nexus 7 (2012) (N7) running Android 4.4 and a Moto X 2nd Gen. (MX) with Android 5.1. Figure 8 shows the measured Wi-Fi handover delays: the time to associate with the other AP (A) and the time to resume UDP traffic (UDP), either using a fixed IP address or DHCP when not specified. The results show relatively long delays for Nexus 7 and important variations for Moto X.

V. Proposed Demo

To show the features of the WalT platform, we propose to bring a mobile set up consisting of one Intel NUC as a WalT server, two managed switches with associated RPis connecting a dozen of STMicroelectronics 802.15.4 sensor motes. The demo will present VizWalT, a plugin for *Cooja*, used to visualize sensor network traffic: Cooja displays traces gathered on the operational network and presents the timeline of events as well as the topology view during network discovery and topology construction.

VI. Conclusions

This paper describes WalT, a reproducible platform for running reproducible experiments. It complements existing specialized platforms with the possibility of reproducing and deploying our own experiment platform for debugging, comparisons, and testing networks in production environments under real-world conditions. WalT offers low cost and standard components, free software, and easy installation based on PoE cabling. We have described two example experiments that can be easily reproduced just by deploying WalT and running experiment images: NTP/PTP time synchronization and measurements of the 802.11 handover of smartphones.

In the near future, we will work on WalT images compatible with IoT-LAB experiments. Our goal is to be able to run on any instance of WalT the same experiments that run on IoT-LAB. We also plan to extend WalT with support for more powerfull SBCs, like RPi 2 and UDOO, and study the support for Android to run experiments with mobile applications easily.

Acknowledgments

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01), the French Ministry of Research projects IRIS under contract ANR-11-INFR-016 and DataTweet under contract ANR-13-INFR-0008-01.

References

[1] D. Kotz, C. C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott, "Experimental Evaluation of Wireless Simulation Assumptions," in *Proc. of MSWiM*, 2004.

[2] K. Tan, D. Wu, A. J. Chan, and P. Mohapatra, "Comparing Simulation Tools and Experimental Testbeds for Wireless Mesh Networks," *Pervasive and Mobile Computing*, vol. 7, no. 4, 2011.

[3] B. N. Taylor and C. E. Kuyatt, "Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results," National Institute of Standards and Technology, NIST Technical Note 1297, 1994.

[4] D. Raychaudhuri, M. Ott, and I. Seskar, "ORBIT Radio Grid Tested for Evaluation of Next-Generation Wireless Network Protocols," in *Proc. of TRIDENTCOM*, 2005.

[5] iMinds Research Institute, "w-iLab.t Generic Wireless Testbeds." [Online]. Available: http://www.iminds.be/en/develop-test/ilab-t

[6] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed," in *Proc. of IEEE World Forum on Internet of Things*, 2015.

[7] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," RFC 7228 (Informational), Internet Engineering Task Force, May 2014.