Design Trade-offs for the Wireless Management Networks of Constrained Device Testbeds

Justin Cinkelj, Marjan Sterk
XLAB LLC
Ljubljana, Slovenia
Email:{justin.cinkelj, marjan.sterk}@xlab.si

Adnan Bekan, Mihael Mohorcic and Carolina Fortuna
Jozef Stefan Institute
Ljubljana, Slovenia
Email: {adnan.bekan, miha.mohorcic, carolina.fortuna}@ijs.si

***Abstract*—A relatively small number of testbeds based on constrained devices use a wireless management network, mostly because of the unreliable communication it enables. However, in some cases, such management networks are the only option due to the target location of such testbeds: outdoors, on light posts, buildings, etc. In this paper, we analyze the design trade-offs encountered when designing a wireless management network for testbeds based on constrained devices. First, we identify two use cases and the functionality needed by the management network in supporting them. Next, we discuss ways of providing the desired functionality and illustrate the decisions we took for designing and implementing the management network for the extension of the LOG-a-TEC testbed together with an initial evaluation. The analysis and the adopted decisions resulted in a management network that is separated from the experimental network providing improved application throughput, together with smaller application level updates/reconfiguration size that significantly shorten the time required to set up a new experiment.**

Keywords: wireless management network; dual stack Contiki; LOG-a-TEC experimental testbed; VESNA platform

I. Introduction

Increasing emphasis is being put on experimentally validated research and a fair number of sensor based testbeds such as Motelab [1], w-iLab.t [2], TWIST [3] and WISEBED [4], just to name a few, exist. According to a recent survey [5], existing sensor based testbeds are located mostly indoor and nodes are controlled through a wired management network such as USB or Ethernet. A number of sensor based testbeds located outdoor such as SmartSantander [6], CitySense [7] and LOG-a-TEC [8] also exist.

The outdoor testbeds typically use a wireless management network (see Table I) since the costs of wiring in the target locations are typically high. SmartSantander uses two IEEE 802.15.4 PHY/MAC interfaces on 2.4 GHz on each node, one for the management and one for the experimental network; and Digimesh[1] for realizing multi-hop functionality on the management network [6]. CitySense uses a Ubiquiti SR9 802.11b/g radio operating on 900 MHz for the management network and a Wistron CM9 802.11 a/b/g operating on 2.4 GHz for the experimental network. The management network also runs a multi-hop mesh routing protocol for providing multihop connectivity [7]. LOG-a-TEC uses one IEEE 802.15.4 PHY/MAC interface (Atmel ATZB-900-B0) on 868 MHz for management purposes on each node and multihop functionality is achieved using BitCloud[2]. The LOG-a-TEC experimental network uses TI CC1101, CC2500 or NXP TDA18219HN silicon tuner [8].

TABLE I. Wirelessly managed outdoor testbeds.

Name	Management network	Experimental network
SmartSantander	IEEE 802.15.4 PHY/MAC on 2.4 GHz Digimesh for mult-hop	IEEE 802.15.4 PHY/MAC on 2.4 GHz
CitySense	Ubiquiti SR9 802.11 b/g 900 MHz, multi-hop mesh	Wistron CM9 802.11 a/b/g on 2.4 GHz
LOG-a-TEC	IEEE 802.15.4 PHY/MAC on 868 MHz and BitCloud	TI CC1101, CC2500 and NXP TDA18219HN

Often, the decision regarding the wireless management network could be better supported by empirical evaluations and trade-off analysis rather then just adopting off-the-shelf components. For instance, when building LOG-a-TEC, our focus was on functionality and we used off-the-shelf components for the wireless management network and custom firmware on the nodes, therefore the system is application specific for spectrum sensing and cognitive radio [8]. As a result, we typically see data rates in the range of 300 bytes/sec. However, much prior work exists on estimating and evaluating IEEE 802.15.4 compatible links, multihop communication and dynamic linking of applications that can be considered as a starting point when designing a wireless management network for a sensor based testbed.

A study of the design choices for the existing wireless management networks of such outdoor testbeds along a thorough analysis of the performance of these networks with respect to various use cases is yet to be published. In this paper, we are taking a first step in this direction by providing an analysis of the design space and trade-offs in designing and implementing a wireless management network for testbeds formed of embedded devices. In this respect this paper provides three contributions. First, we identify two generic use cases for wireless experimental facilities and identify the required functionality for these generic use cases. Second, we identify and discuss state of the art models, developed within various fields of computer science, that can be used for designing and implementing the wireless management network. Although these models have been well investigated individually, their implementation and performance evaluation as a holistic working system is not yet available. Finally, we provide the design

[1]The Digimesh Networking Protocol, http://www.digi.com/technology/digimesh/

[2]BitCloud - ZigBee PRO http://www.atmel.com/tools/bitcloud-zigbeepro.aspx

978-1-4799-5863-4/14/$31.00

choices, implementation and initial evaluation for LOG-a-TEC 2.0. This work may give valuable insights to existing testbed operators and future testbed developers, thus having a notable contribution to the community.

The paper is structured as follows. Section 2 identifies two major use cases and a set of common functionalities required by the management network supporting a testbed. Section 3 discusses the existing models for reprogramming, reconfiguration and updating the nodes of the testbed while Section 4 provides an overview of the speed and reliability of the wireless management network. The discussion on the design, implementation and initial evaluation of the LOG-a-TEC testbed is covered in Section 5. Finally, Section 6 summarizes the paper.

II. Use cases and required functionality

We identify two generic use cases for testbeds consisting of constrained devices such as sensor nodes. The first use case, the monitoring use case (Monitoring-UC), refers to sensor based testbeds that enable monitoring of some phenomena such as energy consumption, humidity, temperature, motion, sound, gases, etc. These kinds of testbeds are typically used by researchers to automatically acquire some data about the phenomena under study. Examples of such testbeds are SmartSantander and CitySense. The second use case, the experimentation use case (Experimentation-UC), refers to sensor based testbeds that support the development of new communication and networking technology by enabling experimentation with new algorithms and protocols. Motelab, TWIST and LOG-a-TEC are examples of such testbeds.

In spite of this division in two major groups, from the point of view of the wireless management network, these testbeds have a set of common functionalities.

a) Need for software upgrades: From the perspective of software upgrades, we identify three types of required updates: OS/firmware upgrades, driver updates and application updates. OS/firmware upgrades are expected to occur when new versions of these software are released or when a major flaw is discovered and needs immediate fixing. The frequency of these upgrades is expected to be of 3-4 per year at most for both use cases. From the perspective of the size of the code to be transfered over the air to the nodes of the testbed, these upgrades tend to be large.

The driver updates are also expected to be required at most few times per year for Monitoring-UCes and for most instances of the Experimentation-UCes. However, for experimental setups that involve MAC layer experiments, the need for upgrades might be more frequent. In many cases, these upgrades can be achieved using dynamic linking, thus avoiding the need for realizing an OS/firmware upgrade. In such cases, the file sent to the nodes of the testbed is relatively small compared to the full OS/firmware image.

The expected application updates vary a lot across testbeds. The more flexible and generic the testbed is, the higher the number of expected application updates. These updates are best performed using dynamic linking and in most cases their expected size is relatively small. Performing a full OS/firmware upload for each application tends to be uneconomical.

b) Need for data collection: Testbeds from the Monitoring-UCes category require sensor measurement data collection while the ones from the Experimentation-UCes category require the collection of the experimental results. Here we distinguish non-time critical data collection and time-critical data collection. When the data collection is time critical, the measured phenomenon or the experimental results have to be sent to the consumer within a small predefined time period from when they were produced and can also be referred to as (near-)real time data collection. This kind of data collection is encountered in sense-act type of systems that base their actuating decision on the sensed value. When the data collection is not time critical, it is typically saved locally on the node and transmitted all at once in batch mode. This collection mechanism is found in scenarios where the data is being post processed.

We also distinguish reliable and unreliable data collection. In the first case, the loss of the measurement data is undesired while is the second case loss of data is not considered a major issue.

c) Need for remote reconfiguration and control: Remote reconfiguration is a desired feature for both use cases. For the Monitoring-UC, the user of the testbed might want to change the sampling rate of the sensor or might want to change the size of the buffer that stores the measurements. For the Experimentation-UC, the user might vary several parameters such as the frequency at which packets are sent, the number of retransmissions, transmitting power, receive channel filter bandwidth, carrier sense indicator etc. In some cases, using remote reconfiguration, the entire experiment can be remotely reconfigured. For instance, using run-time reconfiguration, the entire protocol stack can be reconfigured without flashing the node. Using a fully modular implementation such as CRime [9], an experiment that uses a gossip based algorithm for sending data can be easily reconfigured to use another type of algorithm.

Remote control is a desired feature for sense-act scenarios that can appear in both use cases. For instance, in a Monitoring-UC, a light can be dimmed in response to the sensed values of luminance and presence. In an Experimentation-UC, a node can be controlled to start a transmission after another node sensed a free channel.

III. Reprogramming and reconfiguration models

The basic reprogramming model involves the use of a bootloader to replace the whole image on the node (i.e. flash the node) [10], [11] and is the most suitable for OS/firmware upgrades. With multiple images it is possible to support multiple applications or implement failure recovery. This approach is used by all the wired testbeds to upload new experiments. The main drawback is large transfer size which is less suitable for wireless management networks. One way to reduce the transfer size is to use image compression [12].

More granular upgrades, that may sometimes be more suitable for wireless management networks, can be achieved by implementing functionality as applications, ran by the core OS. Virtual machine interpreters such as Mate [13] or Java based VM [14] have high overhead due to interpreted execution. This can be avoided by deploying applications as modules in native

code. Pre-linked modules have near zero size overhead compared to monolithic application. Dynamically linked modules can be deployed to nodes with different OS images [15], [14]. However the overhead of additional metadata can be relatively large compared to the application code/data size, so the users should check if the overhead is acceptable.

As discussed in Section II, for some application updates it is not needed to update the application program code. Component based development requires splitting a monolitic OS image with application(s) into multiple components, which communicate via well defined interfaces [16], [17]. For instance, RemoWare permits, beside reconfiguration, also the addition of new components via dynamic linking [17].

IV. Speed and reliability of the wireless management network

The wireless management network of the testbed will have to provide multi-hop communication to reach all the nodes and ensure a well connected network so that all nodes can be reached for control, reconfiguration, data collection and software upgrade purposes as discussed in Section III. According to the existing literature, characterizing a multi-hop wireless network and designing the desired management network is not a trivial task. First, it has been shown [18] that the reception probability of the nodes in the network has three regions. In the first region, also called the *effective region*, the packet success rate is above 90%. In the second region, also referred to as *transitional region*, the packet success rate falls off smoothly but exhibits high variation. In the third region, also referred to *clear region*, the packet success rate is below 10%. The boundaries of the three regions and the fall of the success rate are determined by several factors, among which are the frequency band and the used transceiver.

With respect to the transitional region, it has been shown [19] that (1) the link quality is not correlated with distance, and (2) the extent of the transitional region seems to depend on the environment (e.g. outdoor, indoor, presence of obstacles), and the radio hardware characteristics. Additional observations made by the same authors that are particularly relevant for the purpose of this paper are: (1) link quality is anisotropic; (2) links with very low or very high average PRRs (Packet Reception Ratio) are more stable than links with moderate average; (3) over short time spans, links may experience high temporal correlation in packets reception, which leads to short periods of 0% PRR or 100% PRR; (4) the co-location of 802.15.4 and 802.11b networks affects transmission in both networks due to interference, but the transmission in 802.11b networks is less affected; (5) the co-location of IEEE 802.15.4 and 802.15.1 (Bluetooth) networks affects mostly the transmissions in the IEEE 802.15.4 network; and (6) the co-location of IEEE 802.15.4 networks and domestic appliances can significantly affect the transmission in the IEEE 802.15.4 networks.

In order to realize a wireless management network that services well the testbed by being able to reach all nodes at any time, providing good throughputs to enable configuring, controlling and upgrading the network, it would be highly desirable that as many as possible of the links forming the network are in the effective region. For the ones belonging to the transitional region, it is desirable that they are connected to the core of the network via more than one (highly varying) link. In order to achieve this, the candidate transceivers forming the network have to be evaluated and the operating environment and topology also need consideration.

V. Design, implementation and initial evaluation of the LOG-a-TEC cognitive networking testbed

A. Design choices

When building a new testbed from scratch, one is faced with selecting a desired hardware platform and a supported operating system. Heterogeneous testbeds may chose several different such platforms. The use cases and functionality discussed in Section II as well as the considerations with respect to reprogramming, reconfiguration, speed and reliability discussed in Sections III and IV should be taken into account when selecting the hardware and OS. For instance, with some OSes will be impossible to support dynamic linking and/or run time reconfiguration. The resulting combination of hardware/OS selected based on the considerations discussed in this paper will then need to be evaluated and optimized similar to the example provided in this section.

When upgrading an already existing testbed, the guidelines presented in the previous sections still hold, however, there may be already existing constraints on the choice of hardware and software. For instance, the starting points for the extension of the already existing LOG-a-TEC testbed was the VESNA sensor node [8] and the ProtoStack tool [9]. The VESNA platform is already being used in the existing testbed for spectrum sensing and cognitive radio experimentation [8] and is the supporting block of most of our research activities. The VESNA core board contains a 32-bit ARM Cortex-M3 microcontroller running at up to 72 MHz CPU clock. It has 96 kB of RAM and 1 MB of Flash program memory. Additional non-volatile memory is provided by a micro SD card. The ProtoStack tool has been developed to support modular protocol development that would enable easy experimentation with multi-hop routing algorithms using also learned link characteristic for the routing decision - thus enabling cognitive networking experimentation. As ProtoStack relies on the Contiki OS, the extension has to use this operating system.

Starting with these constraints, answers for the following three main issues have to be found:

- How to enable two wireless - experimental and management - networks running in parallel on VESNA with Contiki (subsection B)?
- How should experiment reconfiguration, control and software upgrade be performed (subsection C)?
- Which transceiver should be used for the management network in order to achieve the best possible throughput (subsection D)?

B. Dual-stack Contiki on VESNA

The VESNA platform had a pre-existing dual radio extension board with the following options: TI CC1101 (868 MHz), TI CC2500 (2.4 GHz), AT86RF230 (2.4 GHz) and AT86RF212 (868 MHz) transceivers. Note that the extension

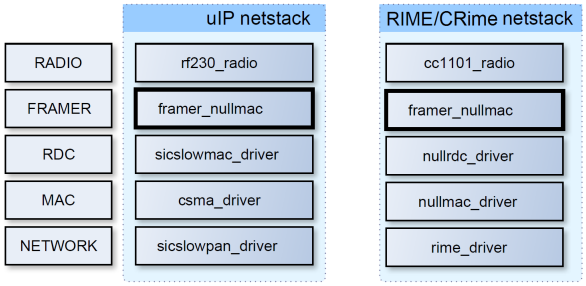

Fig. 1. Dual stack Contiki using 6LowPAN for the management network.

board can support an Atmel and a TI trasceiver at the same time, however it cannot support two Atmels or two TIs at the same time. So while the hardware set-up already supported dual stack (one for management and one for experimentation), a solution for a dual-stack OS had to be developed. Contiki OS includes two protocol stacks, one based on uIPv6 that can be configured as 6LowPAN/uIPv6/UDP/CoAP and the second protocol stack is custom and is referred to as Rime. The two stacks can run one at the time and not in parallel. 6LowPAN assumes a IEEE802.15.4 compatible transceiver and since only the Atmel transceivers comply to this, the most natural decision was to consider the Atmel transceivers for the management network and the TI transceivers for the experimental network.

Next, we extended the Contiki OS with dual stack operation. The original code uses compile-time defined network layers. Some layers are used by both Rime and uIP at the same time (see framer_nullmac in Figure 1), so we modified the networking code to explicitly pass information about which network stack the current packet belongs to. It should be noted that in a single stack Contiki, Rime uses 2 bytes for node network address, while uIP requires 8 bytes. To keep Rime packet small, thus maintaining the low power consumption of the Rime stack, we modified Contiki to permit different network address size for Rime and uIPv6 packets respectively.

Finally, we integrated the new, Composable Rime [9] network stack that enables reconfigurable protocol stacks in the Contiki OS and configured the operating system to support the 6LowPAN based management network and the CRime based experimental network in parallel as depicted in Figure 1.

C. Software upgrades, reconfiguration and control

As discussed in Sections II and III, software upgrades require a bootloader running on the VESNA platform and a large image to be sent to the node over the management network. In the case under investigation, the full image of the monolithic dual stack is in the range of 150 kB as shown in Table II. This image corresponds to a particular application (i.e. single experiment) and needs to be changed should another application be needed (i.e. flash the node).

In order to support minor updates of drivers and applications, we used dynamic loading through the Contiki ELF (Executable and Linkable Format) loader module. Our main interest was to enable dynamically reprogrammable network stacks - in other words, we investigated the possibility of transferring new stack compositions, each representing a new experiment. This requires splitting the application into two parts.

TABLE II. TRANSFER SIZE.

Approach	transfer size [B]	useful size [B]	overhead [%]
monolitic dual stack image	151540	151540	0
hello-world ELF app	1752	399	78
trickle RIME ELF app	11316	3930	65
monolitic dual stack config packet	1635-7937	1635-7937	0

The first part, called core, is responsible for loading the minimal Contiki OS with added ELF loader functionality. This part of the node firmware is not changed during reprogramming. It is responsible for downloading the ELF application through the management network and to dynamically link it with the core OS. To implement it, we had to include the base Contiki image with uIPv6, TCP/UDP and CoAP, also support for: (1) SD card driver and Contiki Coffee FS; (2) utility application to receive ELF file from the network and write it to a file; (3) ELF loader (generic and CPU architecture specific part) to do actual ELF file relocation; and (4) the symbol table stores the addresses and names of all core OS functions, which might be called by the ELF application.

The second part is the ELF application. The application calls functions exported by the core OS, and is compiled as a standard ELF file. When splitting the previous monolithic dual stack image into core OS (with uIP management network) and ELF application (with CRime experimental network) we have the option to leave some code parts, used only by CRime, in the core OS. In particular, we decided to leave the TI CC radio driver in the core OS. As that particular piece of code is already stable, we expect it will not require frequent updates. This resulted in about 50% smaller ELF application file.

The automatically generated symbol table contains the address and the name of each function in the core. Many of them are not even supposed to be used by the application (low level hardware initialization, static functions, ARM CMSIS library functions). Thus we minimized the core OS image size by excluding unneeded function entries from the symbol table.

We looked at the size of the file to be transferred to the nodes over the air for the very simple hello-world application and for a more complex trickle stack. The size of the hello-world ELF file is 1.7 kB while the size for the trickle ELF file is 11 kB as listed in Table II. The trickle ELF application is small compared to the full OS image, but it still does have a significant overhead due to the ELF file metadata.

This observation led us to look at run-time reconfiguration options, where all the CRime modules are loaded on the node using a monolithic system image and then a stack composition message, which describes and configures the experiment, is sent. We used an unoptimized JSON format for the configuration message whose size can vary between 1.6 kB for a simple experiment to 8 kB for a more complex experiment as shown in Table II. The code required for parsing the JSON and generating the experiment added additional 7.5 kB to the size of the system image.

D. Transceiver selection

In Section V-B, we narrowed down the candidate transceivers supporting the management network to two Atmel transceivers: AT86RF230 (2.4 GHz) and AT86RF212 (868 MHz). The first step in evaluating these transceivers was to

determine the three operating regions described in Section IV. The experiment was carried out on a 55 meter corridor of a long building where several WiFi access points are also active. Figure 2 plots the three regions empirically determined in our experiments for the AT86RF230 (2.4 GHz) transceiver. The tests show that the effective region goes up to 22 m in the line of sight conditions (no obstacles on the corridor). Additionally, we performed experiments to understand how the throughput is affected by the packet rate as shown in Figure 3. The results show that rates exceeding 70 packets per second lead to packet losses. The plots for the AT86RF212 (868 MHz) transceiver are similar (omitted for lack of space), with the effective region ending at 28 m and the optimal application rate being also at 70 packets per second.

After determining the three regions for the two transceivers under consideration, we looked at the application transfer rates and various settings that influence these. For instance, by using the header compression enabled by 6LoWPAN, the application payload can be increased thus maximizing the data rate. Figure 4 presents the experimental set-up used for measuring the uplink/downlink throughput. The CoAP client was mimicking reprogramming functionality by sending large files for reprogramming the nodes running CoAP server and data collection functionality by requesting (randomly generated) data from the nodes. The CoAP clients are located on a wired IPv4/IPv6 network, then use a gateway towards the border router which has a wireless management interface for the nodes. The links between the nodes and the border router were within the effective regions, hence reliable.

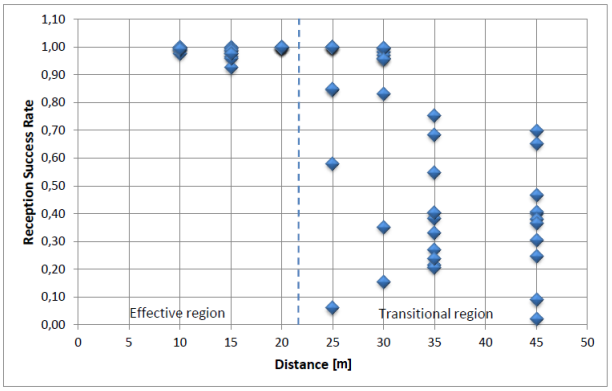

Fig. 2. Reception success rate as a function of distance for the AT86RF230 (2.4 GHz) transceiver.

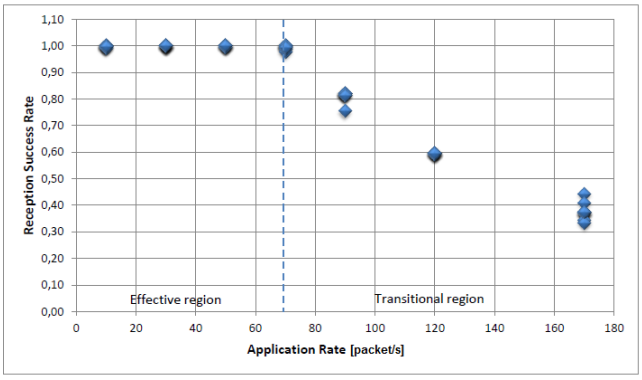

Fig. 3. Reception success rate as a function of application packet rate for the AT86RF230 (2.4 GHz) transceiver.

TABLE III. EVALUATION OF AT86RF212.

Size [bytes]	Download time [s]	Upload time [s]	Download throughput [kbs]	Upload thoughput [kbs]
128	0.185	0.167	5.405	5.988
1280	1.705	1.655	5.865	6.042
12800	16.614	16.666	6.019	6.000
128000	176.324	168.800	5.671	5.924
256000	346.699	339.327	5.768	5.894

TABLE IV. EVALUATION OF AT86RF230.

Size [bytes]	Download time [s]	Upload time [s]	Download throughput [kbs]	Upload thoughput [kbs]
128	0.086	0.069	11.627	14.492
1280	0.686	0.694	14.577	14.409
12800	6.673	6.965	14.985	14.357
128000	68.191	68.899	14.664	14.513
256000	139.696	139.241	14.316	14.363

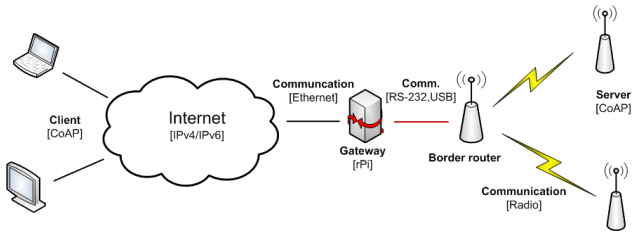

Fig. 4. Measurement set-up for determining uplink and downlink datarates.

We performed two different types of experiments, upload and download for two different sets of radio transceivers Atmel AT86RF212 (Table III) and AT86RF230 (Table IV). Each type of the experiment had five different steps and with each step we were increasing the size of the data (i.e. file) to be transmitted between 128 and 256000 bytes. To get more reliable results we repeated each step 10 times and then we calculated the average throughput value. With respect to the upload and download we performed 100 measurements per radio transceiver. From the results in III and IV, it can be seen that AT86RF230 is achieving higher data throughput and the links are more stable.

In our evaluation we only considered packets that contain payload data, avoiding acknowledgments messages that are sent for each packet and that are not relevant for the application data rate. We decided to use HC01 and HC02 compression for 6LoWPAN because if the CoAP client is accessing the testbed from a different network subnet, the IPv6 address will not be fully compressed in any case. Packet fragmentation was disabled. MAC header compression was not used, because it is supported only by the AT86RF212 MAC. As a note, there are several configurations and tunings that can be performed with such an evaluation. Configurations in the Contiki OS, the used drivers and the point from which the client is accessing the node influence the final performance of the wireless management network. An upper and lower bound as a function of configurations, are indicated by Figures 5 and 6 where the application payload is 0 - 53 and 0 - 95 bytes respectively.

Tables III and IV are summarizing results where 64 bytes of application payload has been used. It can be seen that transferring 128000 bytes (a bit less than a full system image from Table II), 68 seconds are needed when using AT86RF230 and 177 seconds when using AT86RF212. For the dynamic loading of a simple application or it's run-time reconfiguration using a non-optimal JSON format, less than a second is needed with AT86RF230 and under 2 seconds with AT86RF212 - thus being more economic for the wireless management network.

Octets 23	1	40	8	(0-53)	2
MAC Header	**DSP byte**	**IPv6**	**UDP**	**Frame Payload**	**FCS**

Fig. 5. Data frame format - Application layer (worst case).

Octets 23	1	1	1	1	3	(0-95)	2
MAC Header	**DSP byte**	**HC1**	**HC2**	**IPv6**	**UDP**	**Frame Payload**	**FCS**

Fig. 6. Data frame format - Application layer (best case).

VI. Summary and Future Work

In this paper, we took a first step in providing an analysis of the design space and trade-offs involved in setting up a management network for testbeds formed of embedded devices. We first identified two generic use cases - the Monitoring-UC and the Experimentation-UC - that are relevant from an experimenter's point of view. Then we identified the functionality required from the management network to support these use cases, namely the software upgrade, data collection and configuration and control functionalities. We then looked at available models for reprogramming and reconfiguring embedded devices and at aspects concerning the speed and reliability of the multihop wireless management network. Finally we showed the design, implementation and initial evaluation of the new management network for the extension of LOG-a-TEC.

The implemented dual stack network enables to remotely upgrade the nodes of the testbed without the need for wired infrastructure and still benefit from a management network that is separated from the experimental network rather than piggybacking on it. The improved application throughput, together with the smaller application level updates/reconfiguration size significantly shorten the time required to set up a new experiment.

As future work, we plan to extend the evaluation for a multi-hop wireless management network in indoor scenario and then move to a multi-hop wireless management network in an outdoor scenario, similar as we did for LOG-a-TEC 1.0 [20].

Acknowledgment

The authors would like to thank all our colleagues that contributed to this work. This project was partly supported by the FP7 projects CREW (ICT-258301) and ABSOLUTE (ICT-318632) and ARRS through grant J2-4197.

References

[1] G. Werner-Allen, P. Swieskowski, and M. Welsh, "Motelab: A wireless sensor network testbed," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, ser. IPSN '05. Piscataway, NJ, USA: IEEE Press, 2005.

[2] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, and P. Demeester, "The w-ilab. t testbed," in *Testbeds and Research Infrastructures. Development of Networks and Communities*. Springer, 2011, pp. 145–154.

[3] 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*. ACM, 2006, pp. 63–70.

[4] I. Chatzigiannakis, S. Fischer, C. Koninis, G. Mylonas, and D. Pfisterer, "Wisebed: an open large-scale wireless sensor network testbed," in *Sensor Applications, Experimentation, and Logistics*. Springer, 2010, pp. 68–87.

[5] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A survey on facilities for experimental internet of things research," *Communications Magazine, IEEE*, vol. 49, no. 11, 2011.

[6] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis *et al.*, "Smartsantander: Iot experimentation over a smart city testbed," *Computer Networks*, 2013.

[7] J. Bers, A. Gosain, I. Rose, and M. Welsh, "Citysense: The design and performance of an urban wireless sensor network testbed," in *JJ Proceedings of the 2008 IEEE International Conference on Technologies for Homeland Security, Waltham, MA*. Citeseer, 2008.

[8] T. Solc, C. Fortuna, and M. Mohorcic, "Low-cost testbed development and its applications in cognitive radio prototyping," in *Visions on Cognitive Radio*. Springer, 2014.

[9] C. Fortuna, "Dynamic composition of communication systems," in *PhD dissertation*. Jozef Stefan International Postgraduate School, Ljubljana, Slovenia, 2012.

[10] A. Marchiori and Q. Han, "A two-stage bootloader to support multi-application deployment and switching in wireless sensor networks," in *Computational Science and Engineering, 2009. CSE '09. International Conference on*, vol. 2, Aug 2009, pp. 71–78.

[11] S. Brown and C. J. Sreenan, "Software update recovery for wireless sensor networks," in *Sensor Applications, Experimentation, and Logistics*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, 2010, vol. 29, pp. 107–125.

[12] J. Jeong and D. Culler, "Incremental network programming for wireless sensors," in *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, Oct 2004, pp. 25–33.

[13] P. Levis and D. Culler, "Mate: a tiny virtual machine for sensor networks," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, vol. 37, Oct. 2002, pp. 85–95.

[14] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt, "Run-time dynamic linking for reprogramming wireless sensor networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '06. New York, NY, USA: ACM, 2006.

[15] W. Dong, C. Chen, X. Liu, J. Bu, and Y. Liu, "Dynamic linking and loading in networked embedded systems," in *Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on*, Oct 2009, pp. 554–562.

[16] G. Manik and B. Eliane, "Towards the design of a component-based context-aware framework for wireless sensor networks," in *The Sixth International Conference on Sensor Technologies and Applications, SENSORCOMM 2012*, Aug 2012, pp. 101–105.

[17] A. Taherkordi, F. Loiret, R. Rouvoy, and F. Eliassen, "Optimizing sensor network reprogramming via in situ reconfigurable components," *ACM Trans. Sen. Netw.*, vol. 9, no. 2, pp. 14:1–14:33, Apr. 2013.

[18] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM, 2003, pp. 14–27.

[19] N. Baccour, A. Koubaa, L. Mottola, M. A. Zuniga, H. Youssef, C. A. Boano, and M. Alves, "Radio link quality estimation in wireless sensor networks: a survey," *ACM Transactions on Sensor Networks (TOSN)*, vol. 8, no. 4, p. 34, 2012.

[20] T. Šolc and Z. Padrah, "Network design for the log-a-tec outdoor testbed," in *The 2nd International Workshop on Measurement-based Experimental Research, Methodology and Tools*, May 2013.