An architecture for fully reconfigurable plug-and-play wireless sensor network testbed

Adnan Bekan*,†, Mihael Mohorcic*,†, Justin Cinkelj§, Carolina Fortuna*,‡
*Jozef Stefan Institute, Ljubljana, Slovenia
Email: {adnan.bekan, miha.mohorcic, carolina.fortuna}@ijs.si
† Jozef Stefan International Postgraduate School, Ljubljana, Slovenia
‡ iMinds - Ghent University, Gent, Belgium.
§XLAB LLC, Ljubljana, Slovenia
Email: justin.cinkelj@xlab.si

***Abstract*—In this paper we propose an architecture for fully-reconfigurable, plug-and-play wireless sensor network testbeds. The proposed architecture is able to reconfigure and support easy experimentation and testing of standard protocol stacks (i.e. uIPv4 and uIPv6) as well as non-standardized clean-slate protocol stacks (e.g. configured using RIME). The parameters of the protocol stacks can be remotely reconfigured through an easy to use RESTful API. Additionally, we are able to fully reconfigure clean-slate protocol stacks at run-time. The architecture enables easy set-up of the network - *plug* - by using a protocol that automatically sets up a multi-hop network (i.e. RPL protocol) and it enables reconfiguration and experimentation - *play* - by using a simple, RESTful interaction with each node individually. The reference implementation of the architecture uses a dual-stack Contiki OS with the ProtoStack tool for dynamic composition of services.**

***Keywords*—plug-and-play; testbed; architecture; wireless; sensor network; IoT; low end M2M, capillary networks**

I. Introduction

Different aspects of wireless sensor networks (WSN), such as their single and multi-hop connectivity, their energy consumption and system level aspects, have been well studied for more than a decade now. More recently, there are signals of industrial adoption through the Internet of Things (IoT) concept and machine type communications (MTC) paradigm. We are seeing that WSNs are starting to play an increasingly important role in monitoring and automatizing our cities and houses [1]. One of the reasons why their adoption still seems to be slow is the fact that most of the studies were theoretical, with relatively few results verified on actual testbeds [2]. Even though projects such as Arduino and RaspberryPi significantly lowered the experimentation barrier, setting up and tuning an ad-hoc network of sensors, for instance in an industrial warehouse or production plant, is still a challenging and time consuming task.

Typical WSN deployments can be found in research institutions and consist of few tens to few hundreds of low cost, low-power devices with limited communication capabilities. These are typically centrally controlled and have a two or three-tier architecture [3]. The two-tier architecture comprises WSN devices (tier1) and the wired backbone to the server (tier2). WSN devices are organized in a flat architecture in this case. The three-tier architecture includes an additional gateway tier that enables a hierarchical WSN architecture. The three-tier architectures tend to be less energy efficient as the middle tier contains a significant number of high-power devices. Examples of two-tier sensor network deployment is MIRAGE from Intel Berkeley and of three-tier are WISBED, TWIST and SmartSantander.

These deployments are controlled and managed through a wired management network and are mostly used for experimental research on wireless protocols. A number of more recent sensor deployments and testbeds such as SmartSantander [1], CitySense [4] and LOG-a-TEC [5] are located outdoor. To reduce the deployment costs, outdoor testbeds typically use a wireless management network [3]. More recently, a trend towards mobile wireless sensor network testbeds such as MOTEL, CONET-IT and RoombaNet can be noticed [6]. Obviously, due to mobility, these testbeds are also controlled through a wireless management network.

Independent of their architecture, type of management network or degree of mobility, the deployments dedicated to sensor data collection such as IoT-Lab [7], WISEBED [8] and SmartSantander provide RESTful [9] interfaces through which the data is collected. The deployments dedicated to experimental research and development of new wireless technologies, e.g. NITOS, w-iLab.t and TWIST, provide non-RESTful interfaces such as direct ssh access to each node.

In this paper we take a natural next step and propose an architecture for a single-tier plug-and-play testbed for experimental research and development that can be (i) deployed on an ad-hoc basis in any target environment and (ii) configured and controlled using simple RESTful APIs. The proposed architecture is able to reconfigure and support easy experimentation and testing of standard protocol stacks (i.e. uIPv4 and uIPv6) as well as non-standardized clean-slate protocol stacks (e.g. configured using Rime). The parameters of the protocol stacks can be remotely reconfigured through the RESTful API. Additionally, we are able to fully reconfigure clean-slate protocol stacks at run-time. The architecture enables easy set-up of the network - *plug* - by using a protocol that automatically sets up a multi-hop network (i.e. RPL protocol), and it enables reconfiguration and experimentation - *play* - by using RESTful interactions with each particular node. We also provide a reference implementation of the architecture, the

validation of its functionality and an evaluation in terms of the size and deployment time or experimental functionalities.

The main advantage of the proposed architecture is that it enables (i) an ad-hoc deployment of a network of sensors and (ii) custom configuration of the wireless solution in the actual target production environment rather than in a simulator or a lab. This implies that, with the proposed architecture it becomes much easier than before to develop an M2M/MTC/IoT solution, thus fueling innovation potential. Recent market developments show that custom, non-standardized solutions such as the ultra-narrow band, extremely simple lightweight wireless solutions (e.g. SigFox) can scale better and faster than well investigated and standardized technologies.

The paper is structured as follows. Section II discusses challenges for building a custom WSN network with optimal configuration in industrial or building automation environment. Section III introduces a set of requirements for the plug-and-play experimental WSN testbed, while Section IV describes the proposed architecture of such an experimental testbed. Section V describes the reference implementation of the architecture while Section VI details the implemented enablers and procedures for monitoring and control. The validation and evaluation of the reference implementation are provided in Section VII. Finally, Section VIII concludes the paper.

II. Challenges

The adoption of IoT/M2M in application areas such as industry and building automation could be significantly faster if a cost-effective way of deploying reliable wireless communication networks was available. Current off-the shelf solutions tend to be unreliable while custom built solutions by professionals require significant investment costs in infrastructure. The design of a new wireless sensor network with optimal configuration for industrial or building automation environments has to consider a set of challenges outlined in the following.

1) Resource-aware experimentation: The first challenge regards the WSN devices which are typically small in size and cheap. This leads to a series of limitations such as limited memory, lower processing power and short battery lifetime. Given these constraints, the first main challenge is designing an architecture that enables experimentation with as low overhead as possible. This translates into being able to control, debug and reconfigure the network under test using as few bits sent over the air as possible since it is known that the wireless interface consumes most of the energy of such devices [10]. Additionally, the time required for configuring an experiment should be reasonable. Therefore, besides minimizing the bits transferred over the air, the architecture should also reduce complex operations such as large and slow memory relocations on the embedded device itself whenever possible.

2) Context-aware deployment and configuration: The second challenge regards the target deployment of the experimental ad-hoc network. In power plants, warehouses or buildings, there are two main factors that affect the wireless connectivity, (i) the pre-existing wireless infrastructure and (ii) varying set-ups consisting of moving obstacles (e.g. people, furniture, merchandise) that lead to significant variations in multi-path propagation. Assuming these constraints are given before setting up the testbed, it must be possible to determine, for instance via spectrum sensing, the occupied bands and channels as well as the feasible location for deploying the sensors and the distance between them. Then, the deployed network has to be configured accordingly not to interfere with the existing wireless infrastructure and ensure that the chosen frequency band allows working connections (i.e large distance at low frequency vs. small distance at possibly higher frequency).

3) Remote control and optimization: After deploying the testbed and appropriately configuring it on the spot, it should be possible to assess the functioning and reliability of the resulting system over a pre-defined period of time (e.g. few days). A cost efficient way of realizing this is to support remote control and optimization through an easy to use API. As the system is still under test and in most cases it does not need to be immediately integrated with the company's legacy systems, the simplest way of realizing this is to allow remote control over the web and exposing an easy to use RESTful API. This lowers the skill level required to write scripts for visualization, monitoring and configuring the testbed remotely compared to the traditional command line approaches.

III. Requirements

Following the challenges identified in Section II, we can identify a set of common requirements and design goals for the plug-and-play testbed architecture. By addressing these requirements, the resulting testbed architecture should lower the development and experimental evaluation of custom-built wireless solutions in challenging environments such as industrial and building automation and thus increase the innovation potential in wireless sensor networks enabling M2M/IoT.

A. Remote monitoring and diagnosis

The first important requirement from the architecture for an ad-hoc plug-and-play testbed is to enable the remote monitoring of the radio and network parameters as well as diagnosis of the overall network. The nodes in remotely installed testbeds have to be able to provide on request information about their status, health and current configuration. For instance, ping times and round-trip times might be requested each few minutes from the remote monitoring application that assesses the overall performance of the tuned network. Based on these statistics, the network performance can be improved manually or automatically. Micro controller and surrounding temperature might be an important parameter to monitor, especially in industrial production environments, as these affect the performance and the lifetime of the nodes. An overall network configuration such as routing or neighborhood tables as well as current protocol configurations should be provided on demand for performing remote diagnosis.

B. Remote parameter tuning

The second and equally important requirement from the architecture for an ad-hoc plug-and-play testbed is to enable the remote tuning of parameters. These parameters can be grouped into (i) transceiver parameters such as transmit power, operating frequency and operating bandwidth, and (ii) protocol parameters such as contention window size for CSMA, time to live value for IP or the ceiling of the exponential back-off

in TCP. These are all important for the remote configuration and tuning of the custom wireless solution. For instance, if the connection between two deployed nodes proves to be systematically unreliable, perhaps the transmit power has to be increased or the transmit channel changed. After applying the changes, the behavior of new configuration can be re-evaluated.

Remote parameter tuning can be performed at run-time or by rebooting the corresponding node so that the changes take effect. For the sake of experimentation speed, run-time reconfiguration should be supported for as many parameters as possible. This is also convenient for advanced algorithm development where dynamic interference mitigation or power allocation settings are chosen automatically because the system can perform timely and agile changes. For instance, with advanced power control algorithms such as the one proposed in [11], a run-time reconfigurable system can adapt the transmit power and thus maintain good links in dynamic environments where transmitters and obstacles appear and disappear.

C. Over the air software updates and upgrades

The third and equally important requirement from the architecture for an ad-hoc plug-and-play testbed is to enable efficient over the air software updates and upgrades. Such updates are useful for debugging, upgrading existing functionality such as a protocol setup or adding new functionality [12]. In [13], the authors identify three types of required updates by experimental testbeds: OS/firmware upgrades, driver updates and application updates. OS/firmware upgrades are expected to occur when new versions of software are released or when a major flaw is discovered and needs immediate fixing. However, for experimental setups that require software modification, the need for over the air programming (OTAP) might be more frequent. In many cases, these upgrades can be achieved using dynamic linking, thus avoiding the need for realizing a full 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. Performing a full OS/firmware upload for each application tends to be uneconomical.

D. Modular stack reconfiguration

The fourth requirement coming from the architecture for an ad-hoc plug-and-play testbed is to enable modular stack reconfiguration. While the first three requirements are mandatory for such systems, this last requirements is only needed for advanced users that, besides tuning and configuring protocol parameters, intend to reconfigure the protocols that form a stack as part of the design and development of their custom wireless solution. This can also be done via remote reprogramming in case the protocol stack has a monolithic implementation, however it is less energy efficient and more time consuming due to the higher number of bits sent over the air.

For modular implementations of protocol stacks such as the standards based IPv4/6 implementations in PicoMESH [1] or the clean-slate CRime [14], the entire protocol stack can be reconfigured on the node provided all the modules have been pre-installed. For instance, one can easily switch between IPv4 and IPv6 by just changing the layer three protocol implementation module while the MAC and transport layer protocols remain the same. This requirement supports the development of efficient communication protocols and algorithms that are generic and are independent of the operating system. Additionally, the experimental evaluation of cross-layer [15] and cognitive networking [14] techniques can be made much easier with such functionality.

[1]PicoMESH repository - https://github.com/tass-belgium/picotcp-tinyOS

E. Discussion

Table I presents mapping between the challenges identified in Section II and the architecture requirements as identified in this section. It can be seen that by supporting remote monitoring and configuration as well as over the air reprogramming, more resource aware experimentation is facilitated, particularly because of the convenience of the final solution. Additionally, with careful design and optimal engineering, the number of bits sent over the air for experimentation can be reduced. Remote control and optimization can be achieved by enabling remote parameter tuning and software updates/upgrades.

TABLE I. REQUIREMENTS.

Challenges	Req. A	Req. B	Req. C	Req. D
Resource-aware experimentation	✓	✓		✓
Context-aware deployment and configuration	✓	✓		✓
Remote control and optimization	✓		✓	

IV. ARCHITECTURE

In this section, we propose an architecture that enables a fully reconfigurable plug-and-play wireless sensor network testbed, complies to the requirements in Section III and thus addresses the challenges identified in Section II.

Figure 1 depicts the system architecture. The architecture has to support (i) one or more configurable wireless transceivers for experimentation and/or management, (ii) a configurable protocol stack and/or a modular and configurable protocol stack and (iii) a monitoring, control and composition block. Additionally, there must be support for software upgrades to address the requirements in Section III-C. The support for software upgrades can be realized in several ways, for instance by using custom firmware or operating system support. This aspect has been researched in several recent works [12] which discuss in detail about the possible designs and their trade-offs.

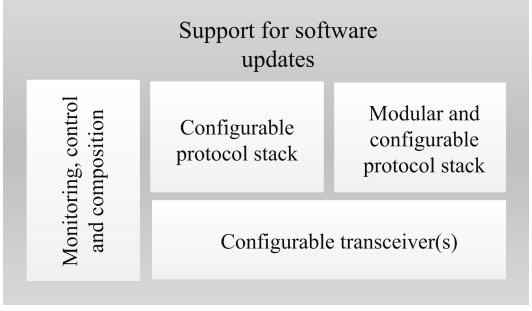

Fig. 1. System architecture.

A. Configurable wireless transceivers

The configurable wireless transceiver has to support parameter reconfiguration (i.e. tx power, etc.) as discussed in Section III-B. This means it has to be carefully selected to be closer to a white-box transceiver that can be reconfigured and can support a large variety of protocol stacks rather than to a black-box one where the transceiver and the protocol stack are tightly integrated and prevent configurations. In some cases, having one single transceiver may be sufficient. This implies that the management network (i.e. the network that performs monitoring, parameter tuning, etc.) is the same as the experimental network (i.e. the network to be optimized and tuned for the custom deployment). In this situation we are dealing with in-band monitoring and configuration.

To avoid in-band management, the architecture also supports two separate transceivers. In this situation, the management and experimental networks can be clearly separated and configured to operate on non-overlapping channels, thus isolating the effects of the management-related communication overhead from the target production wireless network that is subject to customization. This is a preferable solution in most cases, especially when the experimental stack is not fully stable and needs remote debugging. In this case, a reprogramming error can also compromise the management network, and the remote access to the experimental network can be hindered, if single transceiver was used.

Cheap and low-power transceivers tend to have limited support for frequency range. For instance, only SRD 868 MHz or ISM 2.4 GHz, but not both at the same time are supported by such a transceiver. In order to support both frequency bands and also additional ones such as TVWS (UHF, VHF-High, VHF-Low), more transceivers have to be added to the sensor node. This offers more flexibility in developing the solution but poses some additional hardware and software integration challenges. The alternative is to use more powerful software defined radios, however, for the time being, these are more expensive and not designed for operation with constrained devices (even though Embedded USRP is slowly changing this). As a generic architecture, we also have to consider the support of several interfaces to enable flexibility in experimentation.

B. Configurable protocol stack and/or a modular and configurable protocol stack

An open (i.e. white-box) and configurable protocol stack running on top a configurable transceiver is the minimal architectural support for addressing the requirements from Section III-B. For efficiently developing and evaluating variations of the same protocol or enabling advanced experimentation with cross-layer and cognitive networking techniques as discussed in Section III-D, a modular and configurable stack is desirable. These stacks should be able to run with minimum custom firmware as well as integrate with existing operating systems for sensor networks. The firmware and/or operating system should enable software upgrades as discussed in Section III-C, thus adopting existing solutions such as discussed in [12].

In order to enable the *plug* part of the testbed, the stack on which the management operations are performed (see discussion in Section IV-A), should be able to auto-configure the management network, thus requiring no manual work. This makes each experimental device a directly and uniquely addressable and accessible network entity. The experimental network consists of experimental devices and router/sink block. While the experimental devices are only meant for performing experimentation, the router/sink has two functionalities. First one is to seamlessly integrate experimental devices into the existing network, thus providing connection to the end-users. The second functionality is to enable clustering of devices. Regarding the number of routers supported by the architecture, experimental devices can be organized with one router in flat and with many in hierarchical architecture.

C. Monitoring, control and composition block

The monitoring, control and composition block must be able to configure the transceivers and protocol stacks as described in Section III. To enable the *play* part of the testbed, it must provide an easy way of remotely monitoring, configuring and composing network resources. This is typically achieved with a simple API that follows the RESTful architecture. The resulting network is presented in Figure 2 and is comprised of two blocks. On the right side we have the resulting plug-and-play testbed that is connected with the experimenters or end-user block over the wired or wireless backbone link on the left side.

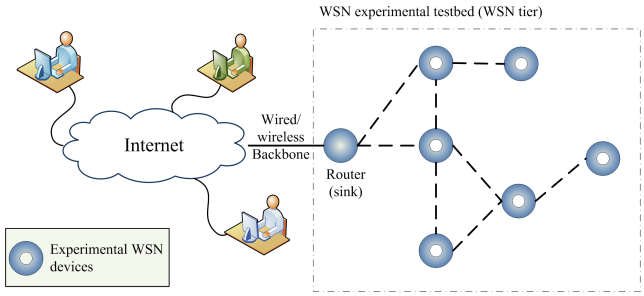

Fig. 2. System topology.

V. Reference implementation

The proposed reference implementation builds on our previous experience with outdoor testbed deployment and experimentation with spectrum sensing and cognitive radio in the LOG-a-TEC testbed [5]. As a result, we use the VESNA sensor platform which is a low power modular platform with several options for transceiver selection and the Contiki OS which is a commonly used sensor operating system that also enables modular updates rather than just monolithic updates.

a) Configurable transceiver selection: For our implementation we chose to use two transceivers. First, we use the TI CC1101 (868 MHz) for the experimental network. This is an open and reconfigurable transceiver with a very minimal implementation on basic MAC functionalities that complies with the architectural specification in Section IV-A. Second, we use the AT86RF231 (2.4 GHz) for the management network. This is an IEEE 802.14.5 compatible transceiver suitable for low power 6LoWPAN [16] networks.

b) Configurable protocol stack and/or a modular and configurable protocol stack: For the experimental network we use the fully modular and reconfigurable CRime stack [14] while for the management network we use Contiki's 6LowPAN/IPv6 stack with RPL. This complies with the architectural specifications in Section IV-B. The RPL protocol is used for automatic management network discovery and configuration. To enable two protocol stacks running in parallel, we used our previous dual-stack Contiki adaptation [13]. The resulting implementation uses Atmel RF231 radio for uIP communication, and TI CC1101 for CRime communication. Both networks can be operated simultaneously without interfering with each other as they operate at different frequencies.

c) Monitoring, control and composition block: The management network enables monitoring, controlling and composing network functionality by using the CoAP protocol on top of UDP/IPv6/6LoWPAN. CoAP includes several HTTP functionalities re-designed for constrained devices such as WSN devices and it is built on top of User Datagram Protocol (UDP)[17]. Therefore it has smaller overhead and enables multicast group communication. To support all the required monitoring, control and composition functionality, we developed a set of CoAP handlers that enable wireless experimenters to remotely *play* with the testbed. These are detailed in Section VI.

To provide remote testbed access from the Internet, we adopted a very lightweight approach. In our implementation the router depicted in Figure 2 shares a small part of the address space of the wide-area network. WSN router only forwards datagrams on the network layer, provides the Internet connectivity and does not include any gateway functionality. For making each experimental device directly addressable and to avoid the problem with lack of IPv4 address space, we used IPv6 with 6LoWPAN optimization. The alternative would be to use an application layer gateway such as used in ZigBee, Z-Wave, Xbee, etc. architectures [18]. However, these gateways are complex to design and manage, and would introduce an additional tier in the system architecture.

VI. Enablers and procedures for monitoring, control and composition

All the enablers for monitoring, control and composition are implemented as CoAP handlers and developed using RESTful principles. This section describes the procedures that use these handlers to address the requirements in Section III.

A. Remote monitoring and diagnosis

Figure 3 presents a sequence diagram that uses CoAP handlers for remote monitoring and diagnosis to perform device hardware and experiment monitoring. First, the $/hardware/status$ is used to receive transceiver configuration setup such as current channel, transmit power, sampling rate, etc. Then the node sends the settings to the user. Second, after the experiment is stared, $/crime/stack\_push$ is used to receive periodic status messages of the running experiment containing the values of LQI and RSSI of the last received packet. Then, after the completion of the experiment, $/crime/get\_prr$ is used to retrieve PRR of the overall experiment. The node then sends this data to the user. As the last event, using the $/crime/get\_results$, the experimenter will retrieve the logged results (RSSI, LQI), stored on the SD card. The node will send the data to the user in chunks if the size of the data exceeds the maximum size of a packet payload.

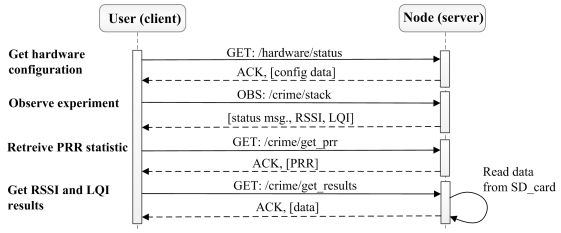

Fig. 3. Remote monitoring.

B. Remote parameter tuning

For remote parameter tuning we implemented several CoAP handlers. Figure 4 presents a sequence diagram that uses two of them to adjust the channel and transmission power of transceivers. First, the $/ti\_cc/set\_power$ is used with the desired power level as a parameter. The experimenter (script) sends it to the target node which acknowledges the request. Then, the $/ti\_cc/set\_chn$ is sent with a numeric value specifying the desired channel. The node acknowledges also this request. Third and last, using the $/ti\_cc/restart$ handler triggers the reboot of the transceiver. After the reboot, the new values, which have been written in the appropriate registries by the routines called by the CoAP handlers, take effect, thus appropriately configuring the transceiver.

In our implementation, several parameter tuning handlers have also been developed for the management network. Instead of referring to the TI transceiver, the requests refer to the Atmel transceiver. This is implemented in a RESTful architecture by changing $/ti/$ into $/atmel/$ in the CoAP request.

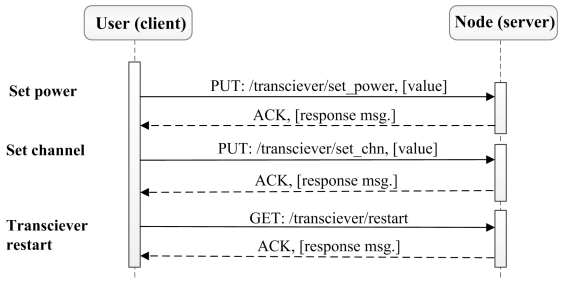

Fig. 4. Remote configuration of channel and transmission power.

C. Over the air software updates and upgrades

Figure 5 presents a sequence diagram that uses CoAP handlers for over the air software updates and upgrades to perform full operating system (OS) update, and Figure 6 presents a sequence diagram that uses the same CoAP handlers for driver and application updates.

First, in Figure 5, the $/firmware/card\_format$ is used if users want to wipe any existing data from the SD card. The node acknowledges this request. Second, the

$/firmware/header$ is used to upload a new OS image meta-data such as size and CRC32 of the image and slot ID where image will be stored on the card. The node also acknowledges this request. Third, the $/firmware/upload$ is used to upload the new OS image. The node acknowledges each transferred chunk of data. Forth, the $/fimware/card\_slot$ is used to select the memory slot of the SD card from which the new OS image has to be loaded. The node will also acknowledge this request. And fifth, the $/firmware/reboot$ is used to reboot the device and start the loading process of the OS image from the SD card into the flash.

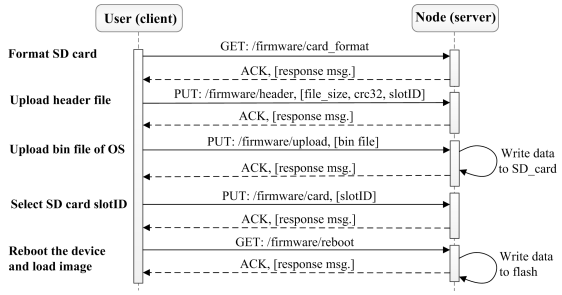

Fig. 5. Full operating system/firmware upgrade.

The first operation in Figure 6 uses the $/elf/header$ handler that uploads the application meta-data such as file size, CRC32 and application name. The node acknowledges this request. Second, the $/elf/upload$ is used to upload the application file. The node acknowledges each transferred chunk of data. After the relocation is done, the node sends the application ID. Third and forth, the $/elf/start$ or $/elf/stop$ is used with application ID as parameter to start or stop the application. The node acknowledges also these requests.

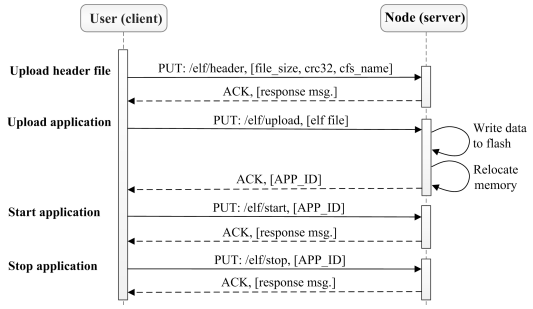

Fig. 6. Application or drivers update.

D. Modular stack reconfiguration

Figure 7 presents a sequence diagram that uses CoAP handlers for the modular stack reconfiguration to perform run-time modular stack reconfiguration using JSON configuration messages. First, the $/crime/upload$ is used to upload the JSON stack configuration message. The node acknowledges successful upload. Second, the $/crime/init$ is used to initialize the stack. The node acknowledges the request after the stack is initialized. Third, the $/crime/start$ handler is used to trigger experiment with the desired sample rate and number of packets that should be transmitted. The node also acknowledges this request. Finally, as forth, the $/crime/stop$ is used to stop the running experiment.

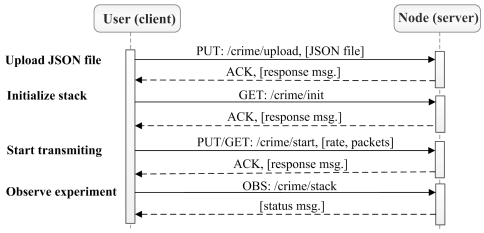

Fig. 7. Run-time stack reconfiguration.

VII. Validation and Evaluation

In order to validate the introduced concept and implementation for the plug-and-play testbed we deployed 20 nodes at the Jozef Stefan Institute campus and obtained a topology as presented in Figure 8. In this testbed, we validated the handlers and sequence diagrams described in Section VI to ensure the provided functionality is as designed.

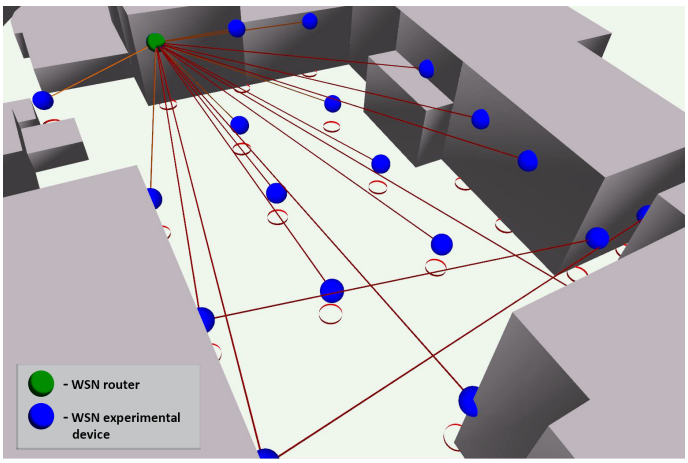

Fig. 8. Example deployment of the plug-and-play testbed.

Table II presents the sizes of selected files or messages that have to be transferred over the air for reprogramming or reconfiguring the network. For full OS reprogramming, the total number of bytes to be sent over the air is 387,800. If we only send a partial code update, the ELF file for the trickle application in this case, the total transfer size is 11,316 bytes. An average message (i.e. CoAP handler) for transceiver reconfiguration requires transferring 32 bytes while a JSON message for reconfiguring the entire protocol stack also amounts to transferring several thousand bytes, in this case, 7937 bytes.

TABLE II. Transfer size.

Use case	transfer size [byte]
monolithic dual stack image	387800
trickle RIME ELF app	11316
transceiver reconfiguration	32
modular stack reconfiguration	7937

The measured average data rate and packet loss in the testbed was 27.56 kbps and 7.054 % of packets, respectively, between the nodes that are 10-30 meters apart as the case in our deployment. Based on these values, the second column in Table III shows the average transfer times required for the messages

listed in Table II. The last column of the table presents node local measurements of the relocation time necessary to find the locations of unresolved function or data addresses, function calls, code execution jumps, etc. As expected, the results confirm that both transfer and relocation times for software upgrades are larger than for reconfiguration messages. Furthermore, while the transfer time for full OS update is larger than for partial code updates, the relocation is smaller for the first (68 sec) versus the second (239 sec). If we consider transfer and relocation time, it can be seen that preparation of an experiment can take from few seconds to few minutes, depending on the complexity of the experiment which, in most cases, is proportional to the number or required configuration messages.

TABLE III. APPLICATION DEPLOY TIME.

Use case	transfer time [s]	relocation time [s]
monolithic dual stack image	109.930	68.168
trickle RIME ELF app	3.207	239.803
transceiver reconfiguration	<0.1	<0.1
modular stack reconfiguration	2.249	<3.0

VIII. CONCLUSION

In this paper, we argue that an architecture for fully reconfigurable plug-and-play wireless sensor network testbed, suitable for fast on-site deployment, demonstration and testing in real operating environment, can foster the adoption of wireless sensor networks in industrial and building automation environments. These environments pose specific challenges that we translated to concrete requirements and design goals that a plug-and-play testbed has to fulfill with respect to remote operation in user-friendly way. In summary, such testbed should be comprised of configurable transceiver(s), configurable and possibly modular protocol stack(s), and a monitoring, control and composition block; furthermore, it should include support for remote software and firmware upgrades and reconfigurations. To this end, different modes of remote programming/configuration may be applicable to different types of experimentation/operation, also depending on the basic design of the sensor nodes and their capabilities. In the reference implementation we presented the operation of developed CoAP handlers for monitoring, control and composition. We validated and evaluated their operation in terms of achieved average data rate, packet loss ratio and application deployment time under different modes of supported remote reprogramming/reconfiguration. In particular, we showed that in some cases a trade off needs to be carefully considered between the time needed for the transfer of a new firmware image and the relocation time for the software upgrade.

ACKNOWLEDGMENT

The authors would like to thank all our colleagues that contributed to this work, particularly dr. Andrej Hrovat for generating Figure 8. This work was in part supported by the European Commission under FP7 projects ProaSense (grant no 612329) and CREW (grant no 258301), and in part by the European Community from the European Social Fund under the Operational Program Human Resources Development for the period 2007-2013.

REFERENCES

[1] L. Sánchez, V. Gutiérrez, J. A. Galache, P. Sotres, J. R. Santana, J. Casanueva, and L. Muñoz, "Smartsantander: Experimentation and service provision in the smart city," in *Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on*. IEEE, 2013, pp. 1–6.

[2] M. Welsh, "Sensor networks for the sciences," *Communications of the ACM*, vol. 53, no. 11, pp. 36–39, 2010.

[3] 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.

[4] 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.

[5] 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.

[6] A.-S. Tonneau, N. Mitton, and J. Vandaele, "A survey on (mobile) wireless sensor network experimentation testbeds," in *Distributed Computing in Sensor Systems (DCOSS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 263–268.

[7] O. Fambon, E. Fleury, G. Harter, R. Pissard-Gibollet, and F. Saint-Marcel, "Fit iot-lab tutorial: hands-on practice with a very large scale testbed tool for the internet of things," *10èmes journées francophones Mobilité et Ubiquité, UbiMob2014*, 2014.

[8] 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.

[9] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[10] V. Shnayder, M. Hempstead, B.-r. Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 188–200.

[11] C. Anton, A. Toma, L. Cremene, M. Mohorcic, and C. Fortuna, "Power allocation game for interference mitigation in a real-world experimental testbed," in *Communications (ICC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1495–1501.

[12] P. Ruckebusch, C. Fortuna, E. De Poorted, and I. Moerman, "Towards a generic internet-of-things architecture: dynamic updates of network and application modules," *Ad-hoc networks*, 2015.

[13] J. Cinkelj, M. Sterk, A. Bekan, M. Mohorcic, and C. Fortuna, "Design trade-offs for the wireless management networks of constrained device testbeds," in *Wireless Communications Systems (ISWCS), 2014 11th International Symposium on*. IEEE, 2014, pp. 245–250.

[14] C. Fortuna and M. Mohorcic, "A framework for dynamic composition of communication services," *ACM Trans. Sen. Netw.*, vol. 11, no. 2, pp. 32:1–32:43, Dec. 2014. [Online]. Available: http://doi.acm.org/10.1145/2678216

[15] L. D. Mendes and J. J. Rodrigues, "A survey on cross-layer solutions for wireless sensor networks," *Journal of Network and Computer Applications*, vol. 34, no. 2, pp. 523–534, 2011.

[16] Z. Shelby and C. Bormann, *6LoWPAN: the wireless embedded internet*. John Wiley & Sons, 2011, vol. 43.

[17] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," 2014.

[18] J. W. Hui and D. E. Culler, "Ipv6 in low-power wireless networks," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1865–1878, 2010.