Connecting Things to the Web using Programmable Low-power WiFi Modules

Benedikt Ostermaier, Matthias Kovatsch, Silvia Santini
Institute for Pervasive Computing, ETH Zurich
Zurich, Switzerland
{ostermaier, kovatsch, santinis}@inf.ethz.ch

ABSTRACT

We present first experiences of using programmable low-power WiFi modules for connecting things directly to the Web. Instead of relying on dedicated low-power radio technology and specialized protocols, we leverage the ubiquity of IEEE 802.11 access points and the interoperability of the HTTP protocol. Using a loosely coupled approach, we enable seamless association of sensors, actuators, and everyday objects with each other and with the Web. Our experimental results show that low-power WiFi modules can achieve long battery lifetime despite the fact that we are using HTTP over TCP/IP for communication.

1. INTRODUCTION

Making everyday objects "smart" by endowing them with computing and communication capabilities as well as with sensors and actuators, is the long-standing vision of Pervasive Computing. Early attempts to connect things to the Internet stem from the desire to remotely monitor and control them. A popular example is the vending machine serving cold beverages at CMU, which was first connected to the Internet in the 1970s.[1] Using a simple text-based interface, users could not only check if beverages were available, but also if they were cold. The idea of connecting everyday objects to the Internet, thus building an *Internet of Things* [8], has gained significant momentum in the last couple of years. More recently, concepts and standards that had been developed for the World Wide Web have been leveraged for representing, discovering, managing, and accessing resources made available by and for Internet-enabled physical objects, creating a *Web of Things* [19].

In this paper, we show how programmable, low-power WiFi modules can be used to connect things directly to the Web and to each other. Instead of using dedicated radio technology and communication protocols that are optimized for low power consumption, such as IEEE 802.15.4, 6LoWPAN [9], or CoAP [15], we rely on the ubiquity of WiFi and the interoperability of HTTP. No gateways nor protocol translations are necessary in order to make physical objects fully Web-enabled. This represents a significant advantage with respect to efforts that rely on more energy-efficient, but less interoperable protocols for low-power networks.

In the next section, we briefly discuss related work. In Sect. 3, we provide background information and present our approach for connecting things to the Web. Sect. 4 describes the hardware platform we leveraged to support our investigations. In Sect. 5, we discuss the prototypical implementation of our approach. Preliminary experimental results that demonstrate the feasibility of our approach are presented in Sect. 6. Finally, Sect. 7 concludes the paper and provides an outlook on further research.

2. RELATED WORK

The Mediacup project [1] represents an early attempt to augment everyday objects with tiny computational devices. For instance, ordinary coffee cups were unobtrusively augmented with sensor nodes measuring movements of the cup or the temperature of the beverage contained by the cup. Nodes were connected to the Internet through an IrDA gateway and could send messages using UDP. A more generic approach was later taken by the Smart-Its project[2], in which sticker-like computational devices that include small sensor nodes were developed, which could be attached to everyday objects in order to make them "smart". In the Cooltown project [16], people, places, and things were given a Web presence, which could be used for interaction. For example, a lamp connected to a gateway server could be controlled over HTTP using form-encoded POST requests.

With the advent of modern Web approaches and also the large-scale deployments of sensor networks, these ideas recently gained momentum. There are services which support the connection of sensors or actuators to the Web, like Pachube.com or Microsoft SenseWeb [7], and research approaches such as sMap [2], sensor.network [6], or WebPlug [11]. Contiki[3], an operating system for resource-constrained embedded devices such as sensor nodes, features an HTTP server and client as central components. There are also approaches to adopt Internet protocols for energy-constrained devices, like 6LoWPAN [9] for using IPv6 over IEEE 802.15.4 or EBHTTP [17] and CoAP [15], as a more efficient HTTP substitute.

[1] `http://www.cs.cmu.edu/~coke/`

[2] `http://www.smart-its.org`

[3] `http://www.sics.se/contiki/`

WoT 2011, June 2011; San Francisco, CA, USA

3. APPROACH

Connecting everyday objects to the Web does not only simplify access to sensors and actuators but also enables a wide range of novel applications and services: One could make things "smart" by leveraging context available from other sensors on the Web, create novel search engines based on real-time data available from sensors [10], and "program the real world" by considering the Web as an application layer for physical objects. We are interested in lowering the entry barrier for connecting things to the Web (and to each other), in order to foster rapid deployment of Web-enabled everyday objects. To do so, we leverage existing WiFi- and HTTP-based infrastructure.

3.1 Connecting to Things

We consider two fundamentally different approaches for connecting things to the Web: The first approach is to attach a pre-configured node to the physical object and use the node's built-in sensors and actuators to monitor and control the object, following the idea of Smart-Its. For example, a node featuring a motion sensor could be attached to an office chair to detect whether it is currently occupied. The second approach is to manipulate the object itself by electrically connecting the node, thereby offering immediate connectivity to sensors and actuators of the physical object. For instance, embedding the node into a light switch, which is essentially a binary sensor, allows to connect the switch to the Web.

Orthogonal to this characterization is the question of power supply: Nodes running on batteries have limited energy and will therefore not be able to stay continuously connected to the wireless network. Instead, they will need to persist in a *sleep mode* most of the time and wake up only on certain events. However, if a grid-based power supply is available, nodes will be able to stay permanently connected to the network and thus also to control actuators in (near) real-time. For example, a node could be attached to a meeting room sign and signal if the room is currently occupied, or it could be embedded in a switchable power outlet to be able to switch the connected devices via the Web. A summary of the considered design space is depicted in Fig. 1.

3.2 Connecting Things to the Web

We rely on the REST paradigm [4] to expose things, including their sensors and actuators, as resources on the Web. Compared to SOAP, REST utilizes HTTP as an application protocol rather than as a transport protocol. For this reason, it introduces less overhead and can also directly benefit from HTTP's features such as cache-control and content-negotiation, for example. Additionally, the resource-oriented view of REST maps nicely to physical resources.

In our approach, every node is running a built-in Web server exposing sensor, actuator, and configuration data as Web resources. These are identified by self-descriptive URLs and can be accessed with HTTP using standard operations such as GET and POST[4]. For example, in order to read a current sensor value, an HTTP GET request is sent to the resource of the sensor. The response includes a textual representation of the current sensor value. This concept works similarly for actuators and configuration variables exposed

[4]To maximize interoperability, we currently resort to POST instead of using the more appropriate PUT.

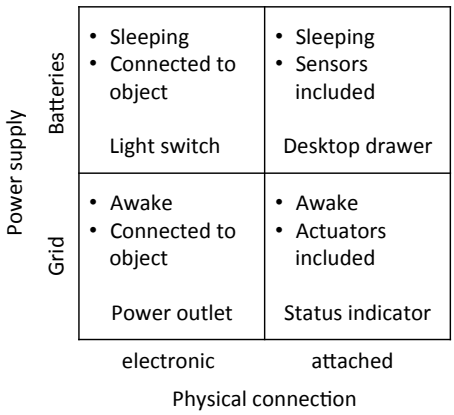

Figure 1: Considered design space

as resources, except that we can also update them by sending an HTTP POST request with the new desired value to their URLs.

To avoid continuous polling on resources and also to enable near real-time applications, we support HTTP callbacks (also known as Webhooks[5]) for resources representing sensor readings. In particular, as soon as the value of a sensor changes, an HTTP POST request is sent by the node to a pre-specified URL, containing the updated value, i.e., the current sensor reading, in the message body.

3.3 Putting Things Together

We illustrate our approach through a simple example consisting of two Web-enabled things: A switchable power outlet, which is a simple actuator and a light switch, which is essentially a binary sensor. Both objects feature a built-in Web server implementing our interaction model.

The power outlet features two resources: The main resource representing the outlet, `http://poweroutlet/` located at the root, and a sub-resource representing its actuator, a relay, located at `http://poweroutlet/power`. Accessing the root resource returns an HTML representation of the outlet, including a graphical representation of its current switch state. Accessing the resource of the built-in relay returns a textual representation of its current state: *false* if it is currently switched off, *true* if it is currently switched on. In order to change the state of the relay, one simply needs to POST the desired state as a textual representation to `http://poweroutlet/power`, using the according content-type `text/plain`. This way, the power outlet can be easily controlled using a script on an HTML page, from the command line using a tool like `curl`, or from a program using an HTTP client library.

The light switch features three resources: The root representing the switch located at `http://lightswitch/`, a sub-resource representing the state (or sensor) of the switch at `http://lightswitch/position`, and a second sub-resource located at `http://lightswitch/callback`. Similar to the power outlet, accessing the root resource of the light switch returns an HTML representation including a depiction of the current state of the switch (Fig. 4). The state of the light switch is also returned as a textual representation reading *true* or *false*, respectively. In order to enable push-based

[5]`http://www.webhooks.org`

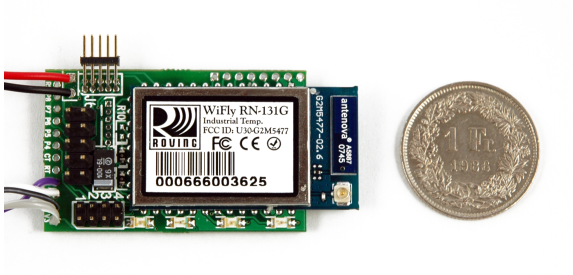

Figure 2: The Roving RN-134 evaluation board, which includes an RN-131.

operations, we can read and set the callback target URL using the same approach as for actuators. We can also dynamically adjust the callback target of a sensor based on some external state like other sensor readings, time, or location without having to place any additional logic in the light switch.

An important aspect of our approach is that we can link sensors and actuators directly: In order to control the power outlet with the light switch, one simply needs to update the resource `http://lightswitch/callback`, the callback target of the light switch, to the value "`http://poweroutlet/power`", which is the URL of the relay of the outlet. Note that this concept of association is comparable to HTML hyperlinks, which are also unidirectional and implemented at the source of the relation. Now each time the light switch is pressed, its state changes and thus an HTTP POST request is sent to the actuator of the power outlet, causing it to reflect the current state of the light switch (Fig. 3). To realize more complex scenarios, like a light switch that controls multiple actuators simultaneously, based on some external condition, one would utilize a service like WebPlug [11], which would receive the HTTP callback from the light switch and distribute it to one or more targets based on some pre-defined conditions. Connecting sensors and actuators via a gateway or Web service may of course be problematic in certain scenarios: To stay with our home automation scenario, users will probably neither tolerate noticeable delays when pressing a light switch nor the outage of their light controls when a Web service or their Internet connection breaks down.

3.4 Addressing Energy Constraints

As mentioned in Sect. 3.1, battery-powered nodes cannot remain continuously associated with an access point and connected to the Internet, as they would otherwise exhaust their batteries very quickly. Indeed, modern low-power IEEE 802.11 transceivers may drain just a few μA of current while in sleep mode but require currents that are more than 10000 times higher when actively sending and receiving data [13, 18].

To provide for an energy-efficient operation mode, we resort to a typical duty-cycled model that interleaves short activity intervals to longer periods during which the node persists in sleep state. While sleeping, the node is not able to send nor receive data, but it can quickly wake up upon certain events. For instance, if the value monitored by an attached sensor changes, the node can wake up, perform the HTTP callback, and go back to sleep. As controlling actuators often requires them to be accessible in (at least near) real-time, we assume that nodes operating in duty-cycled mode feature sensors only.

However, if the node operates in duty-cycled mode and is thus sleeping most of the time, configuring it interactively using the built-in Web server is no longer possible. Instead, the node needs to wake up periodically and poll for configuration data stored at a well-defined location on the Web. Alternatively, nodes could be kept awake for a limited period of time once a specific event (e.g., a "double-click" of the light switch) occurs.

4. PLATFORM

We utilize the RN-131 [13], a low-power programmable WiFi module manufactured by Roving Networks[6]. The module was initially developed by G2 Microsystems[7] under the name G2M5477 [5], before the technology was acquired by Roving Networks. It is a tiny (20x38x4 mm), WiFi-certified, IEEE 802.11 module that features a fully-fledged TCP/IP stack. The module is equipped with a 44 MHz 32-bit RISC processor, 128 kB of RAM, 2 kB of battery-backed memory, 2 MB of ROM, and 8 Mbit of flash storage. The RN-131 supports a number of standard communication interfaces, including UART, SPI, SDIO, RFID, and IEEE 802.11b/g. It also provides 10 general-purpose I/O pins and 8 analog sensor interfaces. The module features a small on-board ceramic chip antenna and also a connector for an external antenna. The built-in IEEE 802.11b/g transceiver supports connection rates of up to 54 Mbit/s and also supports standard authentication methods such as WEP, WPA-PSK, and WPA2-PSK.

According to the data sheet, the RN-131 can be powered between 2.0 and 3.7 V and drains $15 - 212$ mA of current when in active mode, depending on the specific operation [13]. When in sleep mode, however, the module has a nominal current consumption of only 4 μA, which is comparable to the consumption in sleep mode of mote-like devices [12]. Clearly, the shorter the module is kept in active mode, the longer will its battery last and thus the module itself be able to operate. For this reason, the RN-131 is optimized for short boot-up and access point association times. It also provides hardware support for wake-up triggers based on sensor readings or timers.

The RN-131 can be controlled in two different ways: It can connect to a host CPU and operate as a client, handling all the network-related operations. In this scenario, the user's application is running on the host CPU and communicates with the RN-131 via a serial connection. The other possibility is to run the application directly on the RN-131. In order to use this possibility one has to develop the applications using a development kit provided by Roving Networks.

The RN-131 runs eCos[8], an embedded real-time operating system that supports multi-threading. It utilizes the lwIP TCP/IP stack [3] for communication.

Rapid prototyping using the RN-131 is enabled by the availability of the RN-134 evaluation board [14]. This board, shown in Fig. 2, integrates the RN-131 module and some additional components, such as LEDs and a TTL to RS232 level converter.

[6] `http://www.rovingnetworks.com`
[7] `http://www.g2microsystems.com`
[8] `http://ecos.sourceware.org`

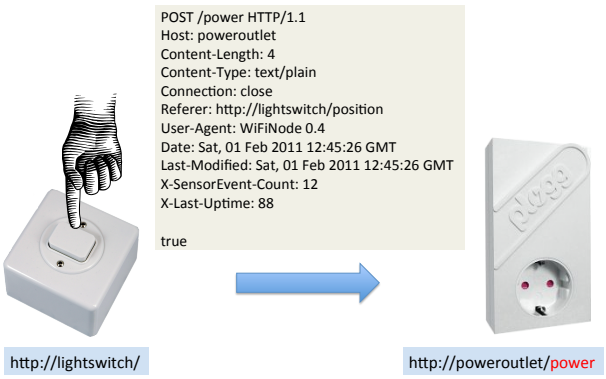

Figure 3: Connecting things using HTTP callbacks

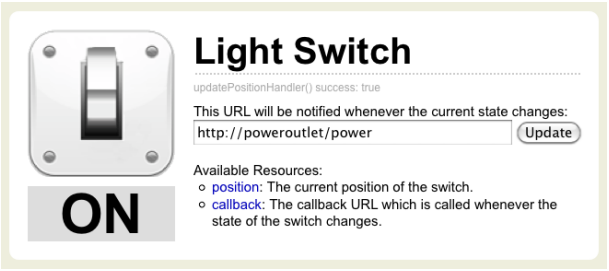

Figure 4: Front-end of the Web-enabled light switch

5. IMPLEMENTATION

We now outline how we connect sensors and actuators with each other and to the Web using the low-power WiFi module introduced in Sect. 4.

5.1 Software

We implemented both a single-threaded HTTP server and a HTTP client, which run on the RN-131 and currently include only the features required for our concept.

Considering the implementation of sensor modules, we have to minimize the time the node is awake in order to preserve battery. As capturing sensor events is supported in hardware by the RN-131, we configured the device to maintain a low-power sleep mode and only wake up on an event of the connected sensor. The typical wake-up cycle is as follows: (1) CPU and RAM are powered up and the device boots our application, (2) it connects to a pre-configured access point, (3) it acquires an IP configuration via DHCP, (4) it looks up the IP address of the callback target, (5) it creates a TCP/IP connection to the host of the callback target, (6) it performs the HTTP callback, and (7) it shuts down again. We can also optimize this process by caching certain settings in the non-volatile memory, such as the IP configuration or the IP address of the callback target so as to skip steps (3) and (4), for example.

For the sensor callback, we leverage both existing headers of the HTTP protocol and include custom headers as a side channel. For example, we set the `Referer` to the monitored resource, `Last-Modified` to the timestamp of the sensor event and `Date` to the timestamp of sending the message. The custom headers `X-SensorEvent-Count` and `X-Last-Uptime` include the number of sensor events and the length of the last uptime. An example of a complete HTTP request for a sensor event is depicted in Fig. 3.

However, as a sensor is usually offline, there is no way to verify if it is still working (if there is no sensor activity) and also no way of (re-)configuring it. We address this problem as follows: In its initial configuration, the sensor stays online and is therefore accessible via its built-in HTTP server. As soon as the user has configured the sensor, he instructs it to shut down by sending an HTTP POST request to the `/sleep` resource of the sensor, containing the text string `true`. The sensor will then periodically wake up and send a heartbeat message to a pre-configured URL, containing status information encoded in JSON. If appropriate, the server can include configuration data in its response, e.g., to update the callback URL of the sensor.

5.2 Hardware

We experimented with three different types of sensors, which were chosen to enable the measuring of activity of an object or a room the node is attached to. First, a reed switch, which in combination with a permanent magnet creates a contactless switch, for example to monitor whether a door is currently open or closed. Second, a micro-mechanical "ball-in-tube" motion sensor, which detects if the attached object is currently moving. Third, a passive infrared (PIR) sensor, which can detect the presence of people in a room, for example. The first two sensors are passive elements which do not consume power by itself, the PIR sensor is an active element which consumes about 300 μA. While all sensors are binary, neither the platform nor our concept impedes the use of non-binary sensors.

We also embedded the RN-134 into several objects, including: A Web-enabled light switch, a Web-enabled power plug based on a modified Plogg[9], which can not only switch a connected consumer load but also sense its current power consumption, and a Web-enabled LED candle, which can act as a subtle notification device.

6. EVALUATION

We performed preliminary experiments and measured the time required to process a sensor event, the callback failure rate, and the energy consumption of the node. Our analysis focuses on the sensor configuration, since energy consumption and the wake-up period are not relevant for actuators, as discussed in Sect. 3.4.

6.1 Quantitative Analysis

For our analysis, we use an optimized approach of the wake-up cycle outlined in Sec. 5.1. In particular, we cache the last channel of the access point, the DHCP lease, the ARP table, and the DNS table. These optimizations can significantly reduce the time required to connect to the callback target.

The nodes were configured to wake up every 10 seconds and send a heartbeat callback using an HTTP POST request to a PHP script hosted at the Web server of our institute. Heartbeat messages were encoded in JSON and included cumulated timing information about the operation of the node, timing information considering the previous heartbeat operation, information regarding the access point in use, and other relevant parameters. The size of a heartbeat message

[9] `http://www.plogginternational.com`

Scenario	S1	S2	S3
Location	Office Space	Home 1	Home 2
Authentication	None	WPA	WPA2
Multiple BSSIDs	Yes	No	No
DHCP lease time	15 mins	24 hours	24 hours

Table 1: Scenarios considered for the evaluation

Scenario	S1	S2	S3
Total average uptime	128 ms	930 ms	182 ms
Callbacks initiated	297180	74444	241428
Callbacks received	295314	66300	240970
Callbacks received (%)	99.37 %	89.06 %	99.81 %

Table 2: HTTP callback statistics

(including HTTP headers) is approximately 900 bytes. The preferred connection rate was set to 24 Mbit/s. In order to prevent an excessive draining of the batteries due to network problems, we limited the maximum time the node can stay awake to 5 seconds.

For our evaluation, we utilized the existing WiFi infrastructure at three different locations, listed in Tab. 1. S1 leverages the WiFi infrastructure of ETH Zurich at the offices of the authors' institute, which is an enterprise-level system consisting of multiple access points and a central DHCP server. The access points at S2 and S3 are single, standard consumer-level routers with built-in DHCP servers.

Table 2 provides aggregated data for each of the three scenarios. As each heartbeat message also includes a sequence number, we can detect failed callbacks at the server. An HTTP callback may fail due to several reasons, including failure to associate with the access point, network problems, and problems at the callback target. The node at S2 performs significantly worse than the nodes at the other scenarios. This is due to the access point used at that location; replacing it with the model used at S3 resulted in performance comparable to S3. This is an important issue, as we cannot assume the existence of access points of a given model when leveraging existing WiFi infrastructure. Note that success rates could be increased by queuing failed callbacks for later transmission, which is currently not implemented.

For a detailed analysis of the average awake times, which is depicted in Fig. 5, we considered only data of successful HTTP callbacks. For each scenario, the average time is further split into time required for the association with the access point, for getting the IP configuration via DHCP, for performing the HTTP callback, and for other operations. As we cache DNS entries, the time for resolving the IP address of the callback target is negligible (< 0.2 ms in each scenario) and thus not depicted. The node performed best at scenario S1, requiring only 98 ms on average for a successful HTTP callback. Interestingly, the additional overhead caused by the short DHCP lease time is overcompensated by the short transmission time to the HTTP target, which is hosted at the same site. For S2 and S3, DHCP time has no impact as a lease is valid for 24 hours. However, for these scenarios we see that the callback requires considerably more time, which is caused by the higher round trip times to the target host. The time required to connect to the access point at S2 is significantly longer than for the other scenarios, because of the mentioned interoperability issues with that access point.

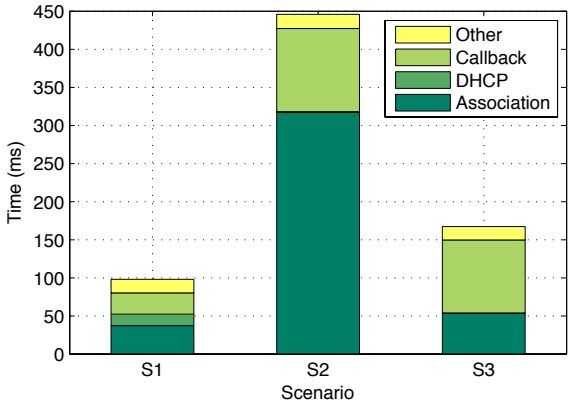

Figure 5: Average time spent awake (considering successful callbacks only)

The shortest awake time at S1, S2 and S3 were 52 ms, 79 ms and 97 ms, respectively.

The node at S1 was optimized for low power consumption by disabling status LEDs and physically removing unused components from the RN-134. It was powered using two daisy-chained NiMH AAA batteries with a capacity of 1100 mAh each. Assuming that the node has a static IP configuration and reports 100 events per day, this would result in an operating time of about 8 years (not considering power consumption in sleep mode and battery self-discharge).

We also tested the response times of our Web server running on the node. The average time for processing an HTTP request, measured at the client between the start of the TCP connection and its shutdown, is 24 ms. This qualifies for (near) real-time control of actuators. We compared this to an Apache HTTP server running on Windows, which required 19 ms on average. For both tests, 1000 requests were issued from a client within the adjacent subnet.

6.2 Discussion

Deploying a traditional sensor network and presenting its sensor readings on the Web usually requires setting up a dedicated infrastructure: A sink node connected to a host computer which is running a dedicated gateway software in order to receive, store, and publish the data. In contrast, deploying WiFi-based nodes is rather simple, as no additional infrastructure is required at the deployment location, provided there is WiFi coverage.

In order to store and publish sensing events, we utilized a simple PHP script consisting of only a few lines of code. However, as a starting point an existing service like `postbin.org` could also be used. For publishing status information regarding the deployed nodes and updating their configuration, we also utilized two PHP scripts. One serves as a heartbeat callback target that stores the heartbeat messages and returns device-specific configuration data to the node. The other script parses the last heartbeat messages and renders an HTML page. Since the heartbeat messages are encoded in JSON, parsing is simple, as PHP comes with a built-in JSON parser.

Leveraging existing IEEE 802.11 networks for communication instead of using dedicated radio technology eases deployment but also comes at the price of reduced and fluctuating network quality, caused by interference in the 2.4 GHz

band or network congestion caused by other hosts. During our experiments at our offices, for which we utilized our university's infrastructure, we sometimes struggled with the fluctuation of the quality of the network, which caused missed callbacks ranging from single events up to several hours. Fluctuations generally seemed smaller at the private networks used at S2 and S3.

An unresolved issue is currently the bootstrapping problem: One has to specify the SSID and, if the network requires authentication, also the key. While this problem could be solved by putting the node initially in ad-hoc mode and then configure it using the built-in Web server, the configuration would need to be sent over an unsecured channel. We are currently investigating whether the RFID interface of the RN-131 could be leveraged in order to communicate this configuration data. Our prototype currently does not support WiFi roaming based on RADIUS, IPSec, or SSL/TLS secured "landing pages". There is also the problem of getting the IP address of the node to know, and to have a configurable domain name for the node. A possible solution would be to collect the IP address at the heartbeat target and show it to the user. Regarding the domain name, a service like `dyndns.com` could be used, for example.

Finally, we would like to emphasize that, as our results are still preliminary, there is still room for optimization. For example, one could use a more receptive external antenna, optimize timing parameters, or reduce the transmission rate of the RN-131 in order to achieve better link quality.

7. CONCLUSIONS

In this paper, we investigated how to connect things to the Web by leveraging existing infrastructure and standards. Instead of relying on dedicated low-power radio technology and specialized network- or application-level protocols, we rely on the ubiquity of IEEE 802.11 and the interoperability of the HTTP protocol. Our results show that, using recently developed low-power IEEE 802.11 transceivers, this approach can be implemented in an energy-efficient manner. Future work includes addressing the bootstrapping problem, to which we sketched a possible solution in the paper. Additionally, the support for WiFi-based roaming based on user authentication would simplify the deployment of mobile nodes.

8. REFERENCES

[1] M. Beigl, H.-W. Gellersen, and A. Schmidt. MediaCups: Experience with Design and Use of Computer- Augmented Everyday Artefacts. *Computer Networks*, 35(4):401 – 409, 2001.

[2] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. sMAP - a Simple Measurement and Actuation Profile for Physical Information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 197–210, New York, NY, USA, 2010. ACM.

[3] A. Dunkels. Full TCP/IP for 8-bit Architectures. In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services*, MobiSys '03, pages 85–98, New York, NY, USA, 2003. ACM.

[4] R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Trans. Internet Technol.*, 2:115–150, May 2002.

[5] G2 Microsystems. Epsilon Module Family Product Brief. http://www.g2microsystems.com/downloads/PB_Epsilon_Modules_Final.pdf.

[6] V. Gupta, P. Udupi, and A. Poursohi. Sensor.Network: An Open Data Exchange for the Web of Things. In *Proc. of the First IEEE International Workshop on the Web of Things (WOT2010)*, Mannheim, Germany, Mar. 2010.

[7] A. Kansal, S. Nath, J. Liu, and F. Zhao. SenseWeb: An Infrastructure for Shared Sensing. *IEEE MultiMedia*, 14(4):8–13, 2007.

[8] F. Mattern and C. Floerkemeier. *From the Internet of Computers to the Internet of Things*, volume 6462 of *LNCS*, pages 242–259. Springer, 2010.

[9] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. Request for Comments: 4944, September 2007.

[10] B. Ostermaier, K. Römer, F. Mattern, M. Fahrmair, and W. Kellerer. A Real-Time Search Engine for the Web of Things. In *Proceedings of Internet of Things 2010 International Conference (IoT 2010)*, Tokyo, Japan, Nov. 2010.

[11] B. Ostermaier, F. Schlup, and K. Römer. WebPlug: A Framework for the Web of Things. In *Proc. of the First IEEE International Workshop on the Web of Things (WOT2010)*, Mannheim, Germany, Mar. 2010.

[12] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *Proceedings of the 4th International Conference on Information Processing in Sensor Networks: Special Track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS 2005)*, pages 364–369, April 2005.

[13] Roving Networks. RN-131 Datasheet. http://www.rovingnetworks.com/Docs/WiFly-RN-131-DS.pdf.

[14] Roving Networks. RN-134 Datasheet. http://www.rovingnetworks.com/Docs/WiFly-RN-134-DS.pdf.

[15] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. Constrained Application Protocol (CoAP). Internet-Draft, January 2011.

[16] T. Kindberg et al. People, Places, Things: Web Presence for the Real World. *Mobile Networks and Applications*, 7(5):365–376, 2002.

[17] G. Tolle. Embedded Binary HTTP (EBHTTP). Internet-Draft, March 2010.

[18] J.-P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP - The Next Internet*. Morgan Kaufmann, 2010.

[19] E. Wilde. Putting Things to REST. Technical Report 2007-015, UC Berkeley School of Information, November 2007.