Bonjour Contiki: A Case Study of a DNS-based Discovery Service for the Internet of Things

Ronny Klauck[1] and Michael Kirsche[2]

[1] Innovations for High Performance Microelectronics (IHP)
Leibniz-Institute for Innovative Microelectronics, Germany
klauck@ihp-microelectronics.com
[2] Computer Networks and Communication Systems Group
Brandenburg University of Technology Cottbus, Germany
michael.kirsche@tu-cottbus.de

Abstract. With the integration of everyday objects and sensors into the Internet, users gain new possibilities to directly interact with their environment. This integration is facilitated by the development of tiny IP stacks that enable a direct Internet connection for resource constrained devices. To provide users with the same level of usability that is predominant in the current Internet infrastructure, a self-configured discovery service for sensors and objects is needed. We thus present a use case of a discovery service based on Multicast DNS and DNS Service Discovery, which we adopt for resource constrained devices and operating systems. Applications using this service can realize direct connections between resource constrained devices following the end-to-end principle of the IP-based Internet, allowing for a seamless integration of potentially millions of objects and sensors into the current Internet and facilitating the pervasive infrastructure that is envisioned by the Internet of Things.

Keywords: Internet of Things, Discovery, mDNS/DNS-SD, Contiki

1 Introduction

The Internet is rapidly reaching over into the physical world with the integration of sensor networks and the embedding of communication technology in everyday objects. This technological progress was named *"Internet of Things"* by Kevin Ashton [1]. The realization of the IoT vision goes hand in hand with the development of Internet Protocol (IP) solutions for embedded devices. With the development of small IP stacks like uIP(v6) [2], embedded and resource constrained devices can be connected directly to the Internet, while at the same time omitting approaches that break with the Internet's end-to-end principle.

With an integration of everyday objects and sensors into the Internet, a new kind of pervasive and ubiquitous infrastructure will be available, leading to new types of applications and services for the interaction of users with their environment. To facilitate a seamless integration, appropriate solutions must be compliant with the current Internet infrastructure. There are two prerequisites

to achieve this: a standardized discovery scheme complying with the IP standard of the conventional Internet domain and the self-configuration ability to handle the possibly large number of objects emphasized by the IoT vision [3]. Discovery, in our work, covers the topics of finding and addressing objects as well as issues of interoperability between different device classes. Self-configuration, as an underlying paradigm of device management, covers the problems of scalability and bootstrapping. Both aspects are vital to provide the same level of service quality that users and developers are accustomed to from the conventional Internet. We therefore want to facilitate solutions that provide autonomous bootstrapping and service discovery instead of complex manual setups.

A main problem for a seamless integration lies in the restrained nature of embedded devices. Typical limitations are: communication over short ranges and failure prone wireless links, limited power supply, and limited memory sizes. Another problem is the multitude of device types, whereas each type has different characteristics, limitations, and physical communication abilities, leading to the need for bridging solutions to enable a device-overlapping interconnection. Next to different communication technologies, embedded devices often use specialized and proprietary protocols on the higher layers of their communication stack. Examples are the Constrained Application Protocol (CoAP) [4] and the Message Queue Telemetry Transport (MQTT) [5] protocol, both rely on gateways and protocol translators to connect embedded devices to Internet-based systems. This leads to a loss of flexibility and end-to-end functionality because messages need to be translated [3], which is typically a time-consuming, failure-prone, and complex task. So instead of trying to introduce new protocols for the discovery of applications and services offered by everyday objects, we propose to adapt established protocols that work in compliance with the current architecture for the integration of *Things* into the *Internet of Things*.

In this work, we introduce a lightweight discovery service implemented for the Contiki [6] operating system for embedded devices. Our approach is based on the combination of *Multicast DNS* (mDNS) [7] and *DNS Service Discovery* (DNS-SD) [8] in a lightweight implementation with adjustments for embedded devices (e.g., small code footprint, minimized overhead), while enabling interoperability and service discovery at the application layer through DNS messages. mDNS and DNS-SD are combined with *Zeroconf* [9] under the term *Bonjour* [10], an established and widely used standard in current IP-based networks. While Bonjour enables computers to communicate ad hoc without a complex setup or manual bootstrapping, our approach will be a first step in this direction by integrating everyday objects and sensor devices in the current IP-based Internet architecture in order to enable the discovery of devices and services of resource constrained devices in compliance with the current infrastructure as a basic discovery service for the Internet of Things vision.

The remainder of this work is structured as follows: Section 2 introduces our case study together with a discussion of related problems. Section 3 presents a service-oriented solution to discover and address sensor devices. A verification through a prototypical implementation is presented in Section 4. Related work is discussed in Section 5 and concluding remarks are presented in Section 6.

2 Case Study for Locating Sensors over IP

An autonomous discovery of devices and services inside a network domain is a central usability criterion. Applications and users need mechanisms to identify new devices and gather relevant information to access offered services. For Internet of Things scenarios, this means that devices need to be discovered in wired and wireless networks, with support through pre-configured infrastructure as well as possibly without (ad hoc) any infrastructure support. Next to these requirements, a discovery solution should also work in compliance with current systems. We thus favor using IP-based solutions to facilitate an easy integration into current networks and systems.

We choose our own working environment as a testbed for a typical IoT scenario. In our “smarter workplace” scenario, different devices (sensors as well as actuators) are distributed to support workers in their daily routines, as Figure 1 illustrates. This leads to a case comparable to common “smart home” scenarios. Smart homes are widely covered in research, although typically used technologies are proprietary and not IP-enabled [11], thus requiring gateways and protocol adapters. As we omit protocol gateways in favor of an end-to-end connection with common Internet protocols, we focus on setting up a testbed of different devices connected over IP links. We connect various stationary computers and workstations, mobile devices (e.g., smartphones, netbooks), as well as embedded devices, sensors, and actuators over their respective communication technologies. Figure 1 depicts the architecture of the system. It is important to note that we cannot omit bridges to interconnect different technologies physically. We assume that one of the following bridging methods is provided to interconnect diverse communication technologies: interconnection via USB/serial port, support for multiple radio technologies within a single device, or embedding the sensor/device in power outlets for a connection over power line communication.

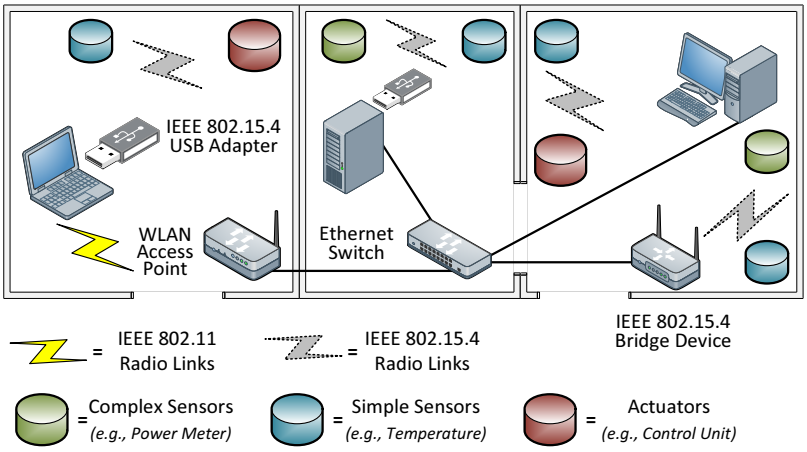

Fig. 1. System Architecture and Use Case

The discovery of non-constrained devices (e.g., workstations, smartphones, netbooks) and their respective services over IP links is provided through today's standards like *Bonjour*, which we adopt for the depicted resource constrained sensors and actuators. For the bridging of different communication technologies, we currently work with USB adapters to link IEEE 802.15.4 and Ethernet over intermediate devices like notebooks. A next step will be the practical deployment of a IEEE 802.15.4 bridging device in form of a small power outlet connected access point for IEEE 802.15.4, comparable to the *Sheevaplug* [12].

Autonomous discovery of new devices and services increases usability and user autonomy. We eventually want to reach a point where sensor devices are just plugged in (in power outlets) or started (if battery operated) and users can “search” the network for new services and then “subscribe” to them, gaining access to information and new ways of interacting with their environment. This operational scheme is directly connected to the concept of Service-oriented Architectures (SOA) [13]. SOA enables a seamless integration and interaction of different device types while specific devices are abstracted as services according to their offered functions. This abstraction of functionality into services is what we ultimately aim for with the discovery service for constrained devices.

Several problems arise due to the resource constrained nature of typical embedded sensor and actuator hardware as well as due to the non-permanent attachment of such devices to the network infrastructure. The following list is a selection of problems that we explicitly address in this work:

- Resource constrained devices operate on tiny 8-bit microcontrollers with limited memory (typically 8-16 KB of RAM and 64 - 128 KB of ROM);
- Network connection is often ad hoc and non-permanent in contrast to non-constrained devices due to a sensor's limited battery-driven power supply;
- Embedded communication standards (e.g., IEEE 802.15.4) support only small Maximum Transmission Units (MTUs) when compared to normal IP-based infrastructure (cp. 802.15.4 with 127 Bytes to IPv6 with 1280 Bytes),

We cover these problems in our solution design with several adaptations and optimizations. The following section presents the standards that we use and our adaptations while Section 4 presents a practical verification that specifically addresses the problems of embedded microcontrollers and limited memory.

3 mDNS / DNS-SD-based IoT Discovery Service

The presented IoT discovery service is based on mDNS and DNS-SD. Both protocols were implemented for Contiki, an operating system for resource constrained hardware with a small code footprint and integrated IPv6 support. Our combination of Contiki, mDNS, and DNS-SD is named *uBonjour*. uBonjour enables a high interoperability between non-constrained and resource constrained sensor devices at the application layer through the use of standardized DNS messages. Our implementation is based on the *Ethernet Bonjour* [14] project. We extended and reimplemented it in an optimized way for Contiki with IPv6 support and tested it with *Avahi* as described in Section 4.

3.1 Background Information

This subsection presents an overview of Contiki, mDNS, DNS-SD, and the SOA concept to substantiate the subsequent descriptions of the discovery service for IP-based embedded sensor devices.

Contiki [6] is an open source operating system, running on embedded hardware with constrained memory and computing resources. IP connectivity is provided by the integrated *uIP* stack, which features ARP, IPv4/v6, SLIP (Serial Line IP), ICMP echo, UDP, and TCP protocol support. Contiki's core system is based on an event-driven kernel with on-demand preemptive multithreading to effectively share marginal memory resources between all processes. To provide concurrency, processes are implemented as event handlers that run till completion and return back to the kernel when finished. A process is implemented either as an application program or as a service. Services provide functions that can be used by application programs. Inter-process communication is provided through the kernel by posting events.

Multicast DNS (mDNS) [7] is one part of a group of standards that is used to enable computers to view or find other devices and to share their services with each other in network environments. mDNS's functionality is to resolve domain names without the help of any server by delivering messages to reserved multicast addresses `224.0.0.251` (IPv4) and `ff02::fb` (IPv6) and the UDP port `5353`. Devices inquire network addresses with requests to a multicast group, while the corresponding device responds with its list of DNS resource records (refer to *DNS-SD*). mDNS is often implemented together with DNS-SD. Both are available for various platforms, for example for Mac OS, iOS and Windows with Bonjour [10], and for Linux, BSD, OpenWRT and Android with Avahi [9].

DNS Service Discovery (DNS-SD) [8] is another part of the standards used to discover devices and share their services. It is combined with mDNS and also provided by Bonjour [10] and Avahi [9]. DNS Service Discovery enables the location and announcement of services of entities in a network domain. DNS resource records are again used to provide information about services. A device usually offers its service by propagating the following DNS records: a service name of the service offering device via the `SRV` record, a hostname/domain name mapped to an IPv4 address via the `A` record and optionally for IPv6 with the `AAAA` record, a user-defined text with the `TXT` record, and an assignment of service instances to a service with the `PTR` record.

Network configuration and management is simplified with mDNS and DNS-SD because entities can detect each other inside a network without previously distributed configuration or prior mutual acknowledgment [15]. Extending a network with additional devices is simplified due to the fact that all devices are able to explore their network vicinity for available services and running applications. An extension of this discovery functionality beyond local network boundaries is enabled by Wide-Area DNS-SD [16] using the same DNS-SD APIs. This can ultimately boost the integration of resource constrained devices into the current Internet infrastructure [17], thus supporting the IoT vision.

Service-oriented Architecture (SOA) is an approach to provide transparency through service abstraction for specific device functions as well as seamless interaction with different device types through self-organization and autonomous connection establishment and management. Transparency through SOA enables devices to browse their network domain for neighbor devices and newly published services [18]. SOA also enables an easier and failure-resistant network bootstrapping [19] because hard-coded start-up addresses and broken pre-configurations can be avoided. Devices perform look-ups for specific services after booting and request necessary information to perform their context-based actions [20]. SOA eventually enables a service-oriented network where devices can act and collaborate spontaneously and autonomously.

3.2 Standard Discovery Service

uBonjour is a lightweight service to discover and address devices and available services in network environments. Application protocols can register their availability as services in the network as well as discover other devices that use the same application protocol. uBonjour helps to fulfill two main goals of the IoT vision: standardized discovery and self-configuration. Figure 2 depicts the architecture of a Contiki-based sensor device including uBonjour as a general service for announcing available application protocols.

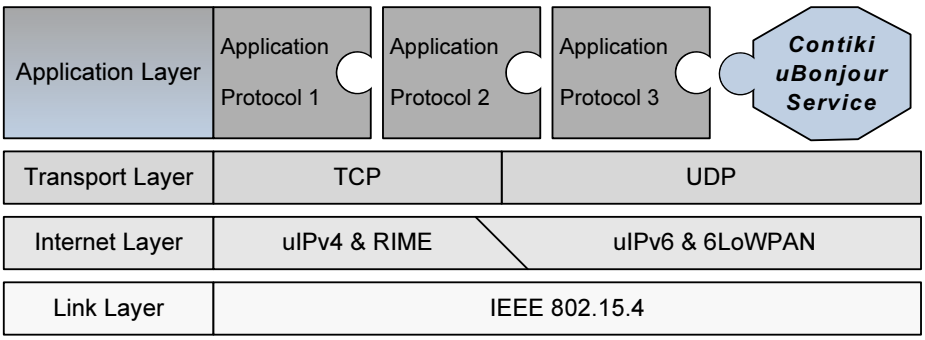

Fig. 2. uBonjour running as a Discovery Service on Embedded Devices

Standardized Discovery defines an application- and device-independent announcement for service offering entities in the network environment. It implements the standardized behavior of mDNS according to [7] and DNS-SD [8] to ensure a compliant message exchange with different kinds of computer systems using either Bonjour or Avahi. This enables computer systems to discover and address sensor devices in an easy-to-use and transparent way without application protocol gateways.

Self-Configuration is an essential aspect of the Internet of Things, as a large number of devices are going to be connected to the Internet. An automatic setup during the bootstrapping phase is necessary to support the diversity of sensor technology as well as configuration possibilities and to facilitate an easy start-up for the subsequent interaction between devices [21]. uBonjour therefore supports self-configuration instead of hard-coded addresses so that devices can scan their network environment and share results without the need to know the exact network topology. Adding a new sensor is performed as easy as starting and requesting information from surrounding devices, while a coordinated exit of a device is enabled with “service unavailable” messages.

3.3 Resolving Hostnames

To discover the address of another device in the network, a device needs to send a DNS query for the domain name to the multicast group. The device with the corresponding domain name replies with an A record including its network address. If a hostname could be resolved, uBonjour will broadcast an event to all listening processes with the IP address as data, or else a timeout for the query will be triggered. Only one name can be resolved at the same time. Sending a new query at the same time will stop the ongoing search for a hostname and will start a new search with the currently submitted hostname.

3.4 Discovering Services

A device initiates the service discovery by sending a `PTR` record to the multicast group containing the name of the searched service. If a service query is resolved, uBonjour posts an event to all processes with `PROCESS_BROADCAST`, containing the resolved IP address and port as data. If the query cannot be resolved, a timeout for the searched service is triggered. Only one service query is supported simultaneously. Sending a new query at the same time stops any ongoing service search and starts a new search with the currently submitted service name.

3.5 Registering, Removing, and Updating Services

To publish an available service, a sensor device has to send four DNS records as described in Section 3.1. Each application running on a device has to register a service with its service name, IP address (provided by Contiki), and port, if it wants to be found in the network by other devices. If a `PTR` query arrives, the corresponding device replies with one `SRV`, `TXT`, `A` or `AAAA`, and `PTR` record. To remove a service from a network, the device needs to send a `PTR` record with the Time-To-Live (TTL) set to zero. Our uBonjour API also supports updating an already published service by resending the four DNS records with changed data. uBonjour can handle up to eight service registrations per device by default. This value can be adjusted to the memory size of the specific device.

3.6 Memory Management Optimization

As constrained devices only support limited memory resources, reducing the code size and the number of used variables and buffers becomes very important. A large quantity (about 60%) of uBonjour's source code size is consumed by the handling of received DNS records and by the generation of DNS responses. We thus optimized the memory management for this code part to minimize the memory consumption. The buffer size of the parser is reduced as the handling is now done directly inside the uIP buffer. The generation of DNS responses requires only a small buffer of the size of a DNS header while the rest of the message generator directly uses the uIP buffer. This *in-place processing strategy* facilitates a memory-efficient discovery service for Contiki.

3.7 Message Size and Flow Optimizations

Contiki's uIP stack uses lower layers (e.g., Rime for IPv4, 6LoWPAN for IPv6) and their provided features (e.g., fragmentation) to efficiently route IP packets in a network. An IP packet relies on the lower layer fragmentation skills, which again depend on the sensor device and its built-in radio transceiver. This limits the supported IP packet size of Contiki, as the following Table 1 shows.

Table 1. Supported IP Payload Sizes of Contiki 2.5 in Byte.

Sensor Device & Radio Module	IPv4	IPv6
AVR Raven / Redbee Econotag	1300	1300
Tmote Sky / AVR ZigBit	108 / 240	240
MEMSIC IRIS / MICAz	128	240
STM32	140	140
MSB430	116	116
ESB	110	110
Zolertia Z1	108	140

Table 1 summarizes the maximum available payload sizes of an IP packet for each supported sensor device and radio module. If these values are exceeded, DNS records will not fit into a single IP packet and IP packet reassembly must be enabled, which will cost an additional amount of RAM and 700 Bytes of code size. Experiments for performing lower layer fragmentation have shown that this mechanism is energy-efficient for request/response cycles as there is no need to optimize the number of fragments [22, Sec. VI]. Devices like the AVR Raven or the Redbee Econotag can handle the lower layer fragmentation very well and thus support a higher IP payload size when compared to other devices (e.g., Tmote Sky, Zolertia Z1). We therefore decided that each DNS record of uBonjour must fit into a single IP packet to avoid the use of IP packet reassembly. The TTL flag

in the DNS header needs to be set for each DNS record with a time in seconds to specify how long a published service will be available. A normal value in this case is 120 seconds, which should be increased for further optimization. Larger TTL values minimize the number of sent messages between devices and they do not interfere with the joining of new devices, because those can explicitly ask for available services in the network. Evaluations of the impact of increased TTL values on the data traffic in larger networks are future work.

Further optimizations for uBonjour are possible by implementing compression methods to keep the data traffic to a minimum, especially in multi-hop networks. Two different methods are currently available: the *Known-Answer Suppression* [7, Sec. 7.1] and the *Duplicate Question Suppression* [7, Sec. 7.3] method. The known-answer suppression method reduces the total number of answers while a device sends a response for a group of devices (including himself), thus reducing the number of necessary responses to gather information about the whole network. For this each device needs to cache published service offerings in the network and wait a randomly chosen time before it answers a request. If a device detects a cached answer to a request, then this answer will be added to its own. If other devices recognize this they will refrain from sending their own answers. The duplicate question suppression method, in contrast, reduces the total number of requests. A device will assume a request as its own when it sees a request that matches its own. This prevents the sending of redundant DNS responses because less `PTR` query messages are sent. Again, each device has to wait a random period before it can send its request.

At the moment we decided to refrain from choosing either one of these two methods for uBonjour because both optimizations need to store a bundle of message related data for proper functionality. This results in an increase of needed buffers and code size, therefore increasing the use of RAM and ROM for the discovery service. To avoid this, we developed our own optimization approach for uBonjour, which is introduced in the following section.

3.8 One-Way Traffic Optimization Approach

uBonjour should assist devices in finding available services and being discovered by other classes of computational devices inside a network. The implementation therefore must be as slim as possible to allow other applications to reside in the device's limited memory as well. Since the *Known-Answer Suppression* and the *Duplicate Question Suppression* method would consume too much memory (as described in the previous section), we developed an economical optimization of mDNS and DNS-SD for resource constrained devices called *One-Way Traffic* (OWT). The OWT optimization is built-in and can be activated during the compilation of uBonjour. This optimization puts a sensor device into a passive mode in which the device only publishes its services periodically (via TTL) and responds only to incoming name and service requests. Passive mode disables the active resolving of hostnames and the ability to parse service query responses from other devices in the network. This also avoids ping-pong effects of DNS responses, while service query responses are targeted only on non-constrained

devices. The activation of OWT and the subsequent disabling of hostname resolving and service query response parsing reduces the used code size significantly and also saves energy because message parsing and network traffic are minimized overall. The OWT optimization leads to a reduction of lines of code too, since the parser handling for incoming service query responses can be skipped, which frees around 400 lines of code. We do not lose much of the core functionality of uBonjour because sensor devices are still able to actively register services and to react to requested services from desktop and mobile systems inside the network environment. Overall, this behavior facilitates the lightweight aspect of the discovery service by coupling non-constrained devices with resource constrained hardware. Desktop and mobile systems can scan their environment for sensor devices with a pre-installed mDNS and DNS-SD service, while nearby sensor devices can directly answer to them with DNS records, without the need of installing additional protocols or using application protocol gateways either. This establishes an easy-to-use discovery mechanism for consumers and offers a simple integration strategy for system administrators.

4 Evaluation

This section presents a verification and prototypical evaluation of the performance of our uBonjour solution in relation to our use case for both IPv4 and IPv6 in terms of memory footprint, message size and response time.

4.1 Experimental Setup

All experiments were performed with Contiki version 2.5 on Zolertia Z1 sensor hardware, which is based on a low-power MSP430F2617 microcontroller with 92 kB of ROM and 8 kB of RAM. The Z1 also provides an IEEE 802.15.4-compliant Chipcon 2420 RF transceiver. The IPv4 test setup consists of devices running uBonjour that are directly connected via the Serial Line Internet Protocol (SLIP) to a computer running Linux. The test setup for IPv6 uses a one-hop network with static routes. One Zolertia Z1 runs the 6LoWPAN border router (shipped with Contiki) connected again to a computer running Linux. The border router converts 802.15.4 / 6LoWPAN frames to Ethernet / IPv6 frames. Two additional Z1 complete the IPv6 setup by running uBonjour. The forwarding of mDNS messages to the Ethernet interface of the Linux PC is done by Avahi (pre-configured parameters were `enable-reflector=yes` and `allow-point-to-point=yes`). We monitor incoming mDNS packets with Wireshark[3] for both cases in order to verify the correctness of generated DNS records from the devices, to measure the DNS record sizes, and to monitor the interaction between the computer and the service offering devices. The response time was measured by sending a `PTR` record to the multicast group and stopping the time between this request and all four received DNS responses sent from the corresponding node.

[3] Wireshark Network Protocol Analyzer [Online] `http://www.wireshark.org/`

4.2 Message Size

Avoiding the use of any kind of additional buffer was mandatory for a slim and memory-efficient implementation of uBonjour. The Zolertia Z1 device has one of the lowest IP payload sizes as depicted in Table 1. It is therefore a good reference platform to test if typical DNS messages (refer to [23, Sec. V-C]) fit into a single IP packet of the uIP stack. The DNS record length combined for all four kinds depends on the sum of the length of the submitted service name, the length of the text information in the TXT record, and the length of the used domain name. The total sum of these freely selectable parameters is *36* for IPv4 and *68* for IPv6 on the Zolertia Z1. The rest is reserved for the DNS header and the DNS message structure. If these measured values are exceeded for the Z1, DNS records will subsequently not fit into a single IP packet and IP packet reassembly must be enabled, which we want to omit as explained in Section 3.7.

4.3 Memory Footprint

uBonjour is realized in only 1450 lines of code. Table 2 shows the detailed memory footprint of uBonjour. The code is compiled with msp430-gcc (GCC) 4.4.5 for the Zolertia Z1. uBonjour with one service that may be registered requires 3.82 kB of ROM / 0.3 kB of RAM for IPv4 and 3.89 kB of ROM / 0.3 kB of RAM for IPv6. As mentioned in Section 3.8, the *OWT* optimization reduces the amount of used memory significantly, in our case it will be cut into half while the lines of code are reduced to around 1050. Each additional service registration for uBonjour will cost around 0.14 kB (IPv4) and 0.23 kB (IPv6) of RAM. These two values are both calculated from the total sum of freely selectable parameters for the DNS records as described in Section 4.2.

Table 2. Memory Footprint of uBonjour with / without uIP stack and OWT.

uBonjour	**ROM in kB**	**RAM in kB**
IPv4 / IPv6	7.12 / 7.69	0.4
IPv4 / IPv6 OWT enabled	3.82 / 3.89	0.3
IPv4 / IPv6 with uIP stack	16.9 / 27.24	1.62 / 3.38
IPv4 / IPv6 OWT with uIP stack	13.6 / 23.44	1.46 / 3.22

The difference in memory consumption of uBounjour between IPv4 and IPv6 is only small because both variations just differ in the used IP address length (16 Byte for IPv6 versus 4 Byte for IPv4 addresses). Minimal larger buffers for sending and storing registered services are therefore needed with IPv6. Unfortunately, this behavior is not adopted by the uIP stack in general: the uIPv6 stack is three times larger in RAM and twice as large in ROM consumption when compared to its IPv4 counterpart. This means that for the Zolertia Z1 nearly half of the memory is allocated by the uIPv6 stack alone. A slim and memory-efficient implementation is therefore even more important for IPv6 then for IPv4.

4.4 Response Time

Discovering services with uBonjour takes 71 *ms* for directly connected sensor devices over SLIP (IPv4). In IPv6 scenarios a 6LoWPAN border router is needed for our use case, hence packets will always be delayed via one-hop. The measured response times for multi-hop are: 1233 *ms* for one-hop, 1954 *ms* for two-hop and 2324 *ms* for three-hop scenarios. No optimizations nor an active forwarding of DNS responses was implemented in uBonjour. Multi-hop routing is handled by Contiki's IP stack and depends on the performance of its used lower layers [24].

5 Related Work

Existing research related to our work can be divided into generic work in the area of Internet of Things architectures and IoT integration strategies as well as existing approaches that use either mDNS or DNS-SD for embedded devices. An approach to use web services over IP links to integrate sensor networks into the current IT infrastructure is presented in [25]. We share their idea of using IP links and Contiki but refrain from using web services. Bardin et al. [20] proposed a service-oriented component framework for the integration of devices and subsequent discovery of services inside heterogeneous networks with the help of a residential gateway. Our approach distributes discovery tasks directly to the devices, making residential and centralized gateways obsolete. The authors of [21] provide a general discussion and an overview of the idea of facilitating a service-based Internet of Things. They resort to a service-oriented yet complex middleware to enable the discovery of services.

Examples for practical mDNS and DNS-SD implementations are Bonjour [10] and Avahi [9], both widely used on desktop and mobile systems. Both are open source and written in C/C++, but too big to fit into the memory of constrained devices. A smaller implementation is Liaison [26], which is around 100 kB in size and written in C++. Porting one of these three implementations for resource constrained devices would be an extensive and time consuming task, because a complex refactoring with subsequent design restructuring is necessary to adapt the implementations for the requirements of constrained devices. [27] stated that they implemented and tested mDNS for Contiki with a memory footprint of only 1.0 kB of ROM and 0.5 kB of RAM, but there is no code proof available. A direct integration of mDNS for Contiki can be found online.[4] It offers an advanced version of the uIP hostname resolver functions and supports IPv4 and IPv6, but there is no plan to extend it with DNS-SD. The most promising mDNS and DNS-SD implementation with only 14 kB is Ethernet Bonjour [14] for the Arduino platform. It was written in C++ for the WIZnet chipset on the Ethernet shield by Georg Kaindl and supports only IPv4 for Ethernet frames. We reused the parser / message generator and its functions stub for uBonjour. C++ parts were ported to C and the WIZnet chipset related code was rewritten to use the uIP stack of Contiki. These measures ensure that uBonjour runs properly on Contiki with a minimized memory consumption.

[4] *darkdeep* Contiki Branch [Online] `http://svn.deepdarc.com/code/contiki/trunk`

6 Final Remarks

This work promotes a discovery service for the Internet of Things vision based on mDNS and DNS-SD. We showed that an existing standard can be used economically on resource constrained devices. Furthermore, uBonjour enables self-configured and autonomously acting sensors in a network environment while it avoids the need of hard-coded bootstrap parameters. Sensor devices can react more precisely on topology changes and on joining or leaving devices. uBonjour will help to integrate sensor devices seamlessly into the Internet infrastructure and also enable an easy access from commodity computer systems.

Through the implementation and evaluation of mDNS and DNS-SD for Contiki, we gathered new insights on implementing available and established protocols, which were originally designed for desktop systems, with a low memory and small code footprint for constrained devices. As a result of the implementation process, we showed that uBonjour can be used for self-configuration and standardized discovery of sensor devices. With uBonjour, we enable a transparent service discovery over the Internet or local networks and offer a standardized integration into current infrastructure for the Internet of Things vision.

In future work, we want to perform simulations of uBonjour to identify bottlenecks, the bootstrapping scaling factor, and further implementation optimizations. We also plan to deploy more devices and extend the current state of our small testbed to facilitate larger practical tests.

References

1. Ashton, K.: That ’Internet of Things’ Thing. Online RFID Journal `http://www.rfidjournal.com/article/view/4986/` (2009) Accessed 10-02-2012.
2. Durvy, M., Abeille, J., Wetterwald, P., O’Flynn, C., Leverett, B., Gnoske, E., Vidales, M., Mulligan, G., Tsiftes, N., Finne, N., Dunkels, A.: Making Sensor Networks IPv6 Ready. In: Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys 2008) - Poster session. (2008)
3. Mattern, F., Floerkemeier, C.: From the Internet of Computers to the Internet of Things. In Sachs, K., Petrov, I., Guerrero, P., eds.: From Active Data Management to Event-Based Systems and More. Volume 6462 of Lecture Notes in Computer Science. Springer, Heidelberg (2010) 242–259
4. Shelby, Z., Hartke, K., Bormann, C., Frank, B.: Constrained Application Protocol (CoAP). Internet Draft, IETF (2011)
5. IBM: MQ Telemetry Transport. `http://mqtt.org/` (2012) Accessed 20-02-2012.
6. Dunkels, A., Gronvall, B., Voigt, T.: Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In: Proceedings of the 29th Annual IEEE Conference on Local Computer Networks (LCN 2004), IEEE (2004) 455–462
7. Cheshire, S., Krochmal, M.: Multicast DNS. Internet Draft, IETF (2011)
8. Cheshire, S., Krochmal, M.: DNS-based Service Discovery. Internet Draft, IETF (2011)
9. The Avahi Team: More About Avahi - Details about mDNS, DS-DNS and Zeroconf. `http://avahi.org/wiki/AboutAvahi` (2011) Accessed 23-02-2012.
10. Apple Inc.: mDNSResponder. `http://www.opensource.apple.com/tarballs/mDNSResponder/` (2011) Accessed 20-02-2012.

11. Cheng, J., Kunz, T.: A Survey on Smart Home Networking. Technical Report SCE-09-10, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada (Sept. 2009)
12. Marvell Technology Group Ltd.: PlugComputer Community. `http://www.plugcomputer.org/` (2012) Accessed 28-02-2012.
13. Zender, R., Lucke, U., Tavangarian, D.: SOA Interoperability for Large-Scale Pervasive Environments. In: Proceedings of the 24th IEEE Conference on Advanced Information Networking and Applications (WAINA 2010), IEEE (2010) 545–550
14. Kaindl, G.: Bonjour/Zeroconf with Arduino. `http://gkaindl.com/software/arduino-ethernet/bonjour/` (2012) Accessed 23-02-2012.
15. Edwards, W.K.: Discovery Systems in Ubiquitous Computing. IEEE Pervasive Computing **5**(2) (2006) 70–77
16. Cheshire, S.: Setting up DNS to Allow Clients to Advertise their own Wide-Area Services). `http://www.dns-sd.org/#WA` (2012) Accessed 20-02-2012.
17. Pohlsen, S., Buschmann, C., Werner, C.: Integrating a Decentralized Web Service Discovery System into the Internet Infrastructure. In: Proceedings of the IEEE 6th European Conference on Web Services (ECOWS 2008), IEEE (2008) 13–20
18. Chen, D.K.: Systematic Review of Applying Service Oriented Architecture in Networking. In: Proceedings of the 6th Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2010), IEEE (2010) 167–170
19. Hammoudeh, M., Mount, S., Aldabbas, O., Stanton, M.: Clinic: A Service Oriented Approach for Fault Tolerance in Wireless Sensor Networks. In: Proceedings of the 4th Conference on Sensor Technologies and Applications (SENSORCOMM 2010), IEEE (2010) 625–631
20. Bardin, J., Lalanda, P., Escoffier, C.: Towards an Automatic Integration of Heterogeneous Services and Devices. In: Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC 2010), IEEE (2010) 171–178
21. Teixeira, T., Hachem, S., Issarny, V., Georgantas, N.: Service Oriented Middleware for the Internet of Things: A Perspective. In Abramowicz, W., Llorente, I., Surridge, M., Zisman, A., Vayssière, J., eds.: Towards a Service-Based Internet. Volume 6994 of Lecture Notes in Computer Science. Springer (2011) 220–229
22. Kovatsch, M., Duquennoy, S., Dunkels, A.: A Low-Power CoAP for Contiki. In: Proceedings of the 8th Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2011), IEEE (2011) 855–860
23. Klauck, R., Gaebler, J., Kirsche, M., Schoepke, S.: Mobile XMPP and Cloud Service Collaboration: An Alliance for Flexible Disaster Management. In: Proceedings of the 7th Conference on Collaborative Computing: Networking, Applications & Worksharing (CollaborateCom 2011), IEEE (2011) 201–210
24. Silva, J., Camilo, T., Pinto, P., Ruivo, R., Rodrigues, A., Gaudêncio, F., Boavida, F.: Multicast and IP Multicast support in Wireless Sensor Networks. Journal of Networks (JNW) **3**(3) (March 2008) 19–26
25. Yazar, D., Dunkels, A.: Efficient Application Integration in IP-based Sensor Networks. In: Proceedings of the 1st ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys ’09), ACM (2009) 43–48
26. Sledz, D.: Liaison - because with Liaison, the client finds you. `http://www.acm.uiuc.edu/signet/liaison/` (2003) Accessed 20-02-2012.
27. Schönwälder, J., Tsou, T., Sarikaya, B.: Protocol Profiles for Constrained Devices. In: Proceedings of the IAB Workshop on Interconnecting Smart Objects with the Internet. (Feb. 2011)