Service Discovery in Resource Constrained Networks using Multicast DNS

Aleksandar Siljanovski, Anuj Sehgal, Jürgen Schönwälder
Computer Science, Jacobs University Bremen, Campus Ring 1, 28759 Bremen, Germany
Email: {a.siljanovski, s.anuj, j.schoenwaelder}@jacobs-university.de

***Abstract*—The Internet of Things is expected to network a large number of constrained devices. The variety of services these devices would deliver and the access they may need to preexisting services introduces a need for an efficient and simple device and service discovery solution. Using standardized and widely adopted solutions would allow for faster integration into the Internet because tools and systems that could utilize such systems already exist. Multicast DNS (mDNS) combined with DNS Service Discovery (DNS-SD) provides a solution that meets these criteria. In this paper we present the implementation of mDNS and DNS-SD for resource constrained devices. An evaluation on the suitability of using these in resource constrained networks is also presented.**

I. Introduction

Internet of things (IoT) is expected to be constituted of self-configuring wireless networks made up of embedded sensors and actuators that interconnect everyday objects. Integration of these embedded sensor and actuator devices is likely to lead to a new services not only due to the capabilities of these devices, but also due to the combination of existing Internet services with the new forms of data [1]. A lot of work has recently gone into designing suitable physical layers and appropriate application protocols and security mechanisms are being investigated.

The IEEE 802.15.4 radio was designed to address the need for a low-cost and low-power wireless communication solution suitable for the IoT [2]. However, since this physical layer only provides small frames and the potential of such networks can increase with IP interconnectivity, the IETF has standardized technologies used in the IoT, so as to ensure interoperability of these emerging networks with the existing Internet. The 6LoWPAN adaptation layer was designed to bring [3] IP networking to IEEE 802.15.4 networks and work has also been carried out in designing suitable application layers protocols and security mechanisms.

Seamless integration of resource constrained networks in the existing Internet is usually problematic due to short communication ranges, wireless links prone to failure, limited power supply and constrained computation resources. One approach to solving this is via gateways that translate [4], [5] messages between constrained network protocols and those suitable for the Internet at large. However, this approach is usually time-consuming, failure-prone and complex, thereby leading to a loss of flexibility and end-to-end functionality of the Internet [1]. This indicates that when possible it would be better to adopt preexisting Internet protocols for use in constrained networks as well.

The emergence of new services enabled by constrained networks and their dependence on utilizing instances of existing services necessitates the development of a service discovery mechanism. Self-organization of large scale networks, as is expected in the IoT, makes autonomous discovery of services and devices in a network important to the usability and manageability of the network itself. Such a service discovery protocol must not only be cross-platform, i.e. it works in constrained and traditional networks, but also suitable for the computational limitations of the devices used in resource constrained networks such as the IoT.

Multicast DNS (mDNS) [6] is a protocol, originally designed for service discovery in home and office environments, that could aid in service discovery within resource constrained networks as well. DNS based service discover [7] is already used across the Internet and coupled with mDNS it is widely used for the discovery of networked devices in ZeroConf environments as well, especially for setting up printers, scanners, music-devices, etc. [8]. Usage of mDNS in resource constrained environments would enable cross-platform service discovery.

In this paper, an implementation of mDNS, suitable for constrained networks, for the Contiki operating system is presented. Other research related to the mDNS implementation is presented in Section II. This is followed an overview on mDNS and service discovery using DNS in Sections III and IV respectively. The development environment and implementation are described in detail in Sections V and VI, followed by insights into testing and evaluation in Section VII. Conclusions are drawn in Section VIII.

II. Related Work

There have been numerous studies on integrating protocols and services from traditional IP networks into resource constrained environments, however, there does not appear to be much effort in the direction of service discovery. The authors of [9] present a web services approach to integrating sensor networks with existing infrastructures, however, there is no discussion on how applications that do not fit within the scope of web services would be dealt with. Other approaches that utilize specialized middleware to keep track of services in a constrained network [10] or use residential gateways to keep track of and discover services offered in heterogeneous networks [11] have also been proposed. These approaches, however, depend upon tertiary devices keeping track of nodes within the network and maintaining a directory of services offered by them. Not only is this cumbersome and error-prone, but it can also lead to stale data in a resource

978-1-4799-5280-9/14/$31.00©2014 IEEE

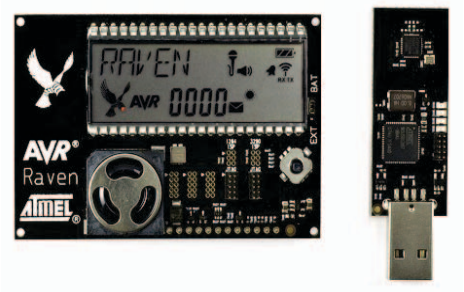

Fig. 1. The AVR Raven development board and the RZUSB stick; both equipped with IEEE 802.15.4 radios.

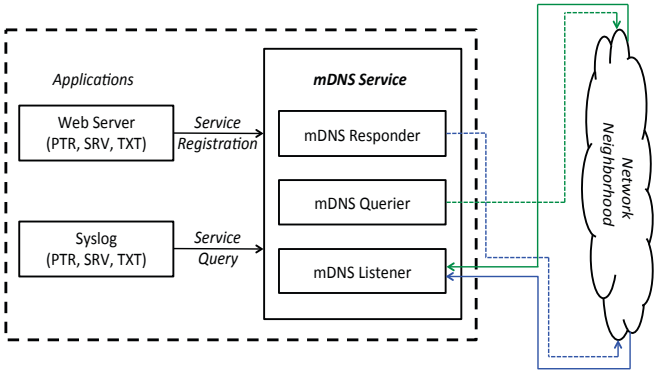

Fig. 2. The implementation architecture of the mDNS service. Blue lines represent communication originating in the network, whereas green represents those from the node. Dashed arrows are mDNS packets going to the network and solid arrows are arriving from the network.

constrained networks, which are especially prone to packet losses, disruptions and nodes possibly disappearing due to lack of energy.

Rather than concentrating information at one or few points of likely failure in constrained networks, it is better to have it distributed, i.e., maintained by nodes responsible for services themselves. The resource discovery approach adopted by CoAP [4] is as such much more promising, since each node has a well known web interface that can be queried for a list of resources and their associated services [12]. However, a limitation of this approach is that it is closely tied to CoAP resources only and services that might operate using other protocols cannot be discovered this way. Furthermore, since CoAP does not offer any node discovery mechanism, this approach assumes that all nodes in a network are already known.

We present, an implementation of mDNS for Contiki is presented in this paper, as a means for service discovery in constrained networks using an existing and widely deployed protocol. Since mDNS can be used to advertise services based on any protocol, advertise new devices that appear in a network and also offers the benefit of being widely used in regular network environments, it forms a good basis for service discovery in constrained networks that need to integrate into existing infrastructures.

III. Multicast DNS (mDNS)

Multicast DNS (mDNS) allows clients to perform DNS queries and seek responses over IP multicast in a network where a DNS server may not be installed. Since mDNS resource record names are auto-generated, unlike DNS, they require little to no configuration or administration. mDNS is designed to function in networks where little infrastructure is available [6], thereby making it a suitable candidate for creating uniquely named entities in resource constrained networks, which tend to form self-organized ad-hoc topologies with no specific infrastructure.

mDNS host names typically take the form `<dns-name>.local` and are limited to 255 bytes in length. As such, any DNS queries related to the `.local` domain are sent to the mDNS multicast group, which in case of IPv6 networks is the address `FF02::FB`, at port 5353 [6]. Since multiple queries could be sent in a single mDNS packet, a new node in the network wishing to discover information about its neighborhood could flood the network due to the number of responses that might arrive. To avoid this situation, a querier can set the top bit in the class field of mDNS packets, thereby indicating to receivers that it is willing to accept unicast responses to its multicast query as well [6]. Such an approach can be quite useful in constrained networks where channel and device resources are scarce.

A response to mDNS queries is only sent when the responder contains authoritative records for the queries received. Cache information is not sent out in response messages to avoid any stale information propagating within the network.

IV. Service Discovery using mDNS

Given a type of service and a domain name in which the client is looking for a service, mDNS service discovery allows the client to discover a list of named instances of that desired service using standard DNS queries.

DNS service discovery (DNS-SD) helps in registering, browsing and resolving service names to DNS host names. Service instances are described using SRV and TXT records [7]. The SRV record takes the form `<Instance>.<Service>.<Domain>` and contains the target host and port where the service can be reached. A TXT record of the same name contains any additional configuration information necessary to access this service, organized in multiple records of key-value pairs.

Service instances can be discovered by sending queries for PTR records that take the form `<Service>.<Domain>`. Such a query results in responses containing a list of records for associated service instances [7]. For example, if a node was interested in discovering all instances of HTTP servers in its neighborhood, it could send a PTR query for `_http._tcp.local` to the mDNS group, i.e. [FF02::FB]:5353. It would then receive responses to the PTR query that contains a list of pointers to instances of this service type. The node can then query the SRV, TXT and AAAA records for the specific instances it might be interested in.

V. Development Environment

The Contiki 2.5 operating system [13] was chosen since it provides an implementation of 6LoWPAN [3], which brings IP networking to the low-power and lossy IEEE 802.15.4

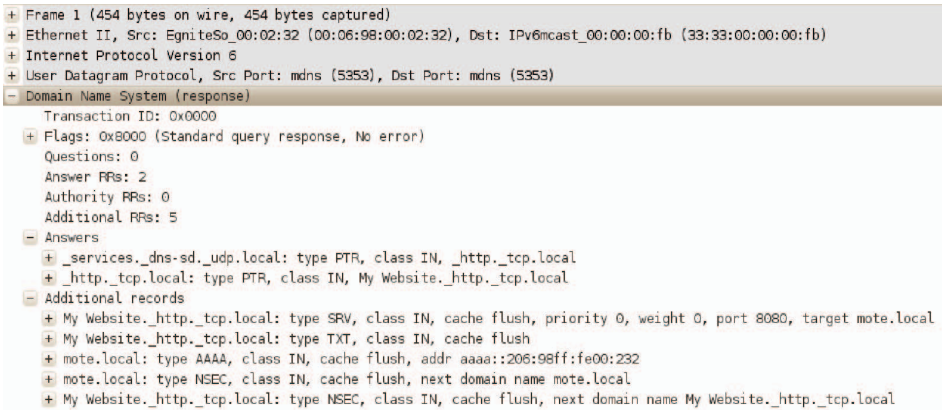

Fig. 3. Screenshot of an mDNS response packet as seen in Wireshark from a successful service advertisement sent by a node in response to a query for all known services in the network.

channel [14]. However, Contiki does not support multicasting packets since the ability to join and leave groups is not present. It is also designed to only accept unicast packets, or those sent to very specific multicast groups (all nodes or all routers). As such, the Contiki operating system was modified by adding into its network libraries a multicast library that allows subscription to any multicast groups an application might be interested in; associated changes were also performed within the Contiki TCP/IP code to make sure that packets destined to subscribed groups are accepted and then only processes that are subscribed to these groups get notified.

The Atmel AVR Raven development kit (shown in Figure 1), which consists of an RZUSB stick and the AVR Raven itself, was used as the development platform since it is compatible with Contiki and also allows for bridging traditional IP networks with 6LoWPAN based IEEE 802.15.4 networks. The AVR Raven is equipped with two microcontrollers; ATMega3290P is used to control the onboard LCD while the ATMega1284P is used for general computing tasks and interfacing with the AT86RF230 transceiver. As such, only the capabilities of the ATMega1284P were used for developing the mDNS implementation. This microcontroller runs at 8 MHz and provides 128 KB flash memory and 16KB RAM. These capabilities, while being representative of devices expected to operate in constrained networks [15], are also sufficient to equip the mote with not only the Contiki operating system and mDNS, but a few additional applications for testing as well.

VI. Implementation

Each process in Contiki that wishes to advertise its service for discovery needs to register itself with the mDNS implementation. Other nodes on the network that wish to discover services offered by a node will send queries, and as such, a node must also be able to listen to these queries and respond to them with any services regarding which it has authoritative information. To allow incoming and outgoing queries and responses to be processed simultaneously, the mDNS implementation was broken into the following parts. An overview of the entire architecture's organization can also be seen in Figure 2.

1. *mDNS API:* This part of the application serves the purpose of allowing applications to register any services they provide, so that they may be advertised, and also letting applications query the network for services that they are interested in. As such, the basic API, which resides within the Contiki network libraries, is also responsible for the data structures that hold the DNS header, queries and resource records themselves.

When registering a service, the application must provide relevant information for the SRV and TXT records using the functions `mDNS_registerSRV` and `mdns_registerTXT`. The instance, service, priority, weight and port information must be provided to `mdns_registerSRV`. Upon receiving this information, the mDNS implementation adds this service as an instance into a list of services offered offered by the node. If it is the first service being advertised, the `_services._dns-sd._udp.local` PTR record will also be created. For example, if the service to be advertised was a website with the name "MyWebsite" on port 8080 and, both, priority and weight being 0, it could be registered with mDNS using the following function call:

```
mDNS_registerSRV("MyWebsite","_http._tcp",0,0,8080);
```

Since there is no specific TXT record associated with this registration, a call to the `mDNS_registerTXT` is not made. A screenshot from Wireshark, showing the mDNS response packet for a query on all services to a network that contains a node advertising the aforementioned example, can be seen in Figure 3. It must be noted that while a method for resolving name conflicts is provided by mDNS, this is not currently implemented.

An application interested in receiving information regarding specific services must use the function `mDNS_query`, which takes the service name, a reference to an empty structure of `mDNSService` type that can be filled with information on the service, a priority and a weight. For example, if a service like Syslog running on the node was interested in discovering servers where it could send messages for logging, it could use the following function call:

```
mDNS_query("_syslog._udp",emptyStruct,0,0);
```

Upon successful completion of the query an event of type `mdnsResponse` is raised so that the process knows when to look inside `emptyStruct` for the data. This structure

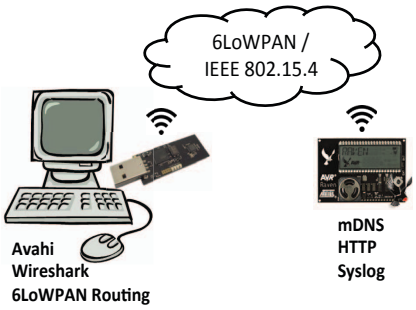

Fig. 4. Output from Avahi on a Linux machine that acts as a router to the 6LoWPAN network. The "My Website" service of type _http._tcp is successfully advertised on tap0, which is the RZUSB stick masquerading as the Ethernet interface capable of interacting with an IEEE 802.15.4 network.

will contain the IP address, port and any associated TXT records as a list. In case multiple replies matching the specified priority and weight were received, only the first one is picked. The query sending and processing of received messages is handled by the `mDNSQuerier` and `mDNSListener` Contiki applications.

2. *mDNSQuerier:* The `mDNSQuerier` application is responsible for receiving queries from the API, formatting them so that they fit into the IP data buffer used by Contiki and then dispatching them to the mDNS multicast group using the Contiki multicast library. Multiple arriving queries are aggregated together so that only up to one aggregate query is sent into the network within any one second period. A list of all queries sent is maintained by this application and made available to the `mDNSListner` so that arriving responses can be appropriately processed. This application must be set to autostart in the Contiki program before the mDNS library can be utilized.

3. *mDNSListener:* This application, part of the mDNS implementation, is responsible for receiving all mDNS queries and responses. It uses the multicast library to bind itself with the mDNS multicast group, i.e. [FF02::FB]:5353, using UDP. All packets arriving at this endpoint are then passed to this application for further processing.

A packet received by this application is first processed to check if it contains any resource records which have been previously queried by the `mDNSQuerier`. If so, an event is raised and the appropriate process informed, with the data being filled into the appropriate structure. The packet is then checked for any queries it might contain to compare them against a list of registrations and a response is sent via the mDNSResponder process in case a match is found. This application must also be started in order for the mDNS library to function.

4. *mDNS Responder:* This process is an intermediate process that is started to dispatch any mDNS responses that need to be sent to answer received queries. It formats all relevant resources into DNS packets and dispatches them to the mDNS multicast group. Since the process is only needed during dispatch of response messages, it gets created on-demand so as to avoid using memory when it is idle.

VII. Testing and Evaluation

A tool-set comprising of Wireshark, avahi-daemon and avahi-browse was used in order to test the functioning of the mDNS implementation. Avahi is an mDNS implementation for Linux, which facilitates service discovery. The avahi-daemon component registers local IP addresses and static services, and then advertises or discovers them using mDNS and DNS-SD.

An overview of the experimental setup can be seen in Figure 4. The mDNS application was loaded, along with an HTTP server to present a simple webpage and a Syslog client to log messages at a server location, on an AVR Raven mote. Each of these Contiki applications was modified to make use of the mDNS application. The HTTP server was configured to register itself with mDNS so that it would be advertised on the network and the Syslog application was configured to seek instances of Syslog servers where it could dispatch syslog messages.

The RZUSB stick was programmed with the Contiki firmware that allows it to appear as an Ethernet interface on Linux machines, and connected to a computer running the Ubuntu 10.04 distribution. This allows for easy bridging of an existing traditional IP network with a 6LoWPAN network of constrained devices. The Ethernet interface appears as `tap0` on the computer. The `radvd` routing daemon on Ubuntu was configured to advertise the IPv6 prefix aaaa::/64 over the `tap0` interface. The avahi-daemon service was configured to advertise a Syslog server that was installed on the Ubuntu machine, over the tap0 interface as well.

The avahi-browse tool was used to discover all services advertised via mDNS on a network, the output of this test can be seen in Figure 5. As can be seen from the output, the website advertised by the AVR Raven (IP address aaaa::206:98ff:fe00:232), appears in the list of services discovered on the `tap0` interface. This confirms that the service advertisement implementation works correctly, and can function with preexisting cross-platform mDNS tools as well. While not depicted in the figure, the mote was also able to send its log messages to the Syslog server on the Ubuntu machine, thereby confirming that service discovery works as well.

Evaluating the interoperability and memory usage of the implementation are important because these provide insight into how suitable it is for use in constrained networks. The implementation is already known to work with the Avahi tools. It was also evaluated for interoperability against the Bonjour SDK and iTunes. The implementation was able to successfully detect Bonjour SDK advertised services and all AVR Raven advertised services were also visible on the Windows XP machine. When a DAAP service was advertised by the AVR Raven, it was listed in iTunes as a possible library to browse.

To obtain the memory usage of an application in Contiki it is necessary to first obtain the base memory usage of the operating system. The `avr-size` tool lists the amount of Program, Data and EEPROM memory used by a binary. The Program memory contains the `.text`, `.data` and `.bootloader` sections, which is basically an indication of the total Flash memory occupied. An analysis of the `contiki-avr-raven.map` file, generated at compile time, reveals that the `.bootloader` section takes up 2 KB of Flash memory. By using a simple Contiki program containing

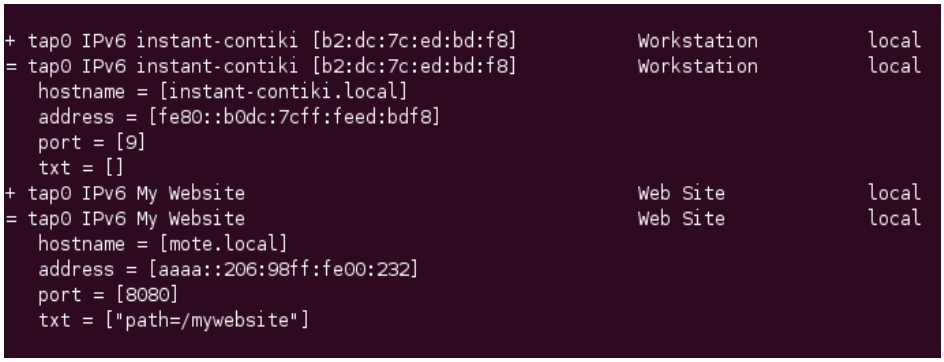

Fig. 5. Output from Avahi on a Linux machine that acts as a router to the 6LoWPAN network. The "My Website" service of type _http._tcp is successfully advertised on tap0, which is the RZUSB stick masquerading as the Ethernet interface capable of interacting with an IEEE 802.15.4 network.

the HTTP server and Syslog client, the `avr-size` tool determined that the basic operating system occupies about 25.3 KB in `.text` section, thereby using about 27.3 KB of Flash. The statically allocated RAM used, in the `.data` section, is about 10.7 KB.

For advertising a HTTP server and querying for a Syslog server the mDNS application occupies 7.2KB in Flash memory and 0.2KB of static RAM. Processing incoming queries and responses occupies 3.6KB of Flash memory and 0.7KB of static RAM. Optimizations are possible by using Flash memory to store constants and streamlining the implementation, however, this was not done to preserve portability of the mDNS implementation to other platforms.

These results indicate that mDNS based service discovery is a viable solution for constrained networks, especially since it works with preexisting tools and can successfully discover or advertise services related to many different protocols.

VIII. Conclusions

The interconnection of resource constrained networks to existing Internet infrastructures presents the need for service discovery mechanisms that can operate in both types of networks and yet fit within the computational constraints of the embedded devices used in constrained networks. While there are multiple proposed approaches for service discovery in constrained networks, most of them are either tied to specific protocols, or utilize specific nodes within a network to maintain a directory of active nodes and related services. This, while being inefficient, can also lead to failures within lossy networks. As such, this paper presented an implementation of Multicast DNS based service discovery for the Contiki operating system as an approach to be used in resource constrained networks. Since this approach is already widely used in traditional networks, it offers the opportunity to utilize existing tools to achieve interoperability without additional complexity. Testing and evaluation of the implementation shows that it occupies limited computing resources on embedded devices, thereby making it suitable for constrained networks.

Acknowledgments

This work was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Program.

References

[1] F. Mattern and C. Floerkemeier, "From the Internet of Computers to the Internet of Things," in *From active data management to event-based systems and more*. Springer LNCS, 2010, vol. 6462, pp. 242–259.

[2] L. De Nardis and M. Di Benedetto, "Overview of the IEEE 802.15.4/4a Standards for Low Data Rate Wireless Personal Data Networks," in *4th Workshop on Positioning, Navigation and Communication (WPNC)*, Hannover, Germany, March 2007, pp. 285–289.

[3] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," *IETF RFC 4919*, August 2007.

[4] Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol (CoAP)," *IETF Internet-Draft <draft-ietf-core-coap-18>*, June 2013.

[5] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "PMQTT-S – A Publish/subscribe Protocol for Wireless Sensor Networks," in *3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE)*, Bangalore, India, January 2008, pp. 791–798.

[6] S. Cheshire and M. Krochmal, "Multicast DNS," *IETF RFC 6762*, February 2013.

[7] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," *IETF RFC 6763*, February 2013.

[8] P. Palmila, "Zeroconf and UPnP Techniques," Helsinki University of Technology, Tech. Rep., 2007.

[9] D. Yazar and A. Dunkels, "Efficient Application Integration in IP-based Sensor Networks," in *First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)*, Berkeley, CA, USA, November 2009, pp. 43–48.

[10] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas, "Service Oriented Middleware for the Internet of Things: A Perspective," in *Towards a Service-Based Internet*. Springer LNCS, 2011, vol. 6964, pp. 220–229.

[11] J. Bardin, P. Lalanda, and C. Escoffier, "Towards an Automatic Integration of Heterogeneous Services and Devices," in *IEEE Asia-Pacific Services Computing Conference (APSCC)*, Hangzhou, China, December 2010, pp. 171–178.

[12] Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format," *IETF RFC 6690*, August 2012.

[13] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *29th Annual IEEE International Conference on Local Computer Networks (LCN)*, Tampa, FL, USA, November 2004, pp. 455–462.

[14] J. A. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile, "IEEE 802.15.4: A Developing Standard for Low-power Low-cost Wireless Personal Area Networks," *IEEE Network*, vol. 15, no. 5, pp. 12–19, 2001.

[15] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained Node Networks," *IETF Internet-Draft <draft-ietf-lwig-terminology-07>*, February 2014.