Wireless and Mobile Technologies, 2023, Vol. 5, No. 1, 1-6
Available online at http://pubs.sciepub.com/wmt/5/1/1
Published by Science and Education Publishing
DOI:10.12691/wmt-5-1-1

A Testbed for Protocol Analysis for the Internet of Things

Edward Sereko Younge*

Aalto University – School of Electrical Engineering, Finland
*Corresponding author: eddieyounge1@gmail.com

Received January 05, 2023; Revised February 11, 2023; Accepted February 20, 2023

Abstract In future smart environments, wireless sensor networks will play a key role in sensing, collecting, and disseminating information about environmental phenomena. These are made possible by the availability of sensors that are smaller, cheaper and intelligent. In addition, the sensors have wireless interfaces with which they communicate with one another in a network. However, these sensors have limited processing resources, memory and power. Due to such limitations, the design of a wireless sensor network depends heavily on the application and associated factors such as the environment, the objectives of the application design, cost, hardware, and system constraints. This paper investigated the difference(s) between the PUSH and PULL protocols by the use of a test-bed in different topologies of a wireless sensor network, in the "Internet of Things" concept. At the end, the better protocol was selected.

Keywords: *PUSH, PULL, internet of things, wireless sensor network*

Cite This Article: Edward Sereko Younge, "A Testbed for Protocol Analysis for the Internet of Things." *Wireless and Mobile Technologies*, vol. 5, no. 1 (2023): 1-6. doi: 10.12691/wmt-5-1-1.

1. Introduction

In future smart environments, wireless sensor networks will play a key role in sensing, collecting, and disseminating information about environmental phenomena. [1] These are made possible by the availability of sensors that are smaller, cheaper and intelligent.

In addition, the sensors have wireless interfaces with which they communicate with one another in a network. However, these sensors have limited processing resources, memory and power. Due to such limitations, the design of a wireless sensor network depends heavily on the application and associated factors such as the environment, the objectives of the application design, cost, hardware, and system constraints. [2]

Sensor networks have been for a while subject to research [17,18,19,20,21] and that today's technology can bring this closer to reality. Also, sensor networks are no longer used in isolation, but seen as part of the Internet, which gets us to the "Internet of Things" concept.

1.1. Objectives

This paper studied the difference(s) between the PUSH and PULL protocols by the use of a test-bed in different topologies of a wireless sensor network (WSN), in the "Internet of Things" concept.

- This paper focused on the differences between the PUSH protocol (regular UDP) and the PULL protocol (CoAP) with respect to network behavior due to changes in traffic rates.
- This was to understand the behavior of the protocols and/or the technologies used in the "Internet of Things", one element of the future of the Internet.

2. Literature Review

2.1. Introduction

This section gives the theoretical background of the main concepts used in this work. These are the REST architecture, the CoAP protocol and the RPL protocol. The web environment using HTTP (Hyper Text Transfer Protocol), is used for distributed, collaborative and hypermedia information systems on the application level. Practical information systems need more functionality than simple retrieval, in addition to search, front-end update and annotation. HTTP supports an open-ended set of methods and headers that show the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI), as a location (Uniform Resource Locator) or as a name (Uniform Resource Name), for showing the resource to which a method is to be applied. The messages are passed in a way identical to that used by the Internet mail as specified by the Multipurpose Internet Mail Extensions (MIME). [5] MIME allows for textual message bodies in character sets other than US-ASCII, an extensible set of different formats for non-textual message bodies, multi-part message bodies and textual header information in character sets other than US-ASCII. [6]

Furthermore, HTTP is used as a generic protocol for interaction between user agents and proxies/gateways to other Internet systems, in addition to those supported by SMTP (Simple Mail Transfer Protocol), NNTP (Network News Transfer Protocol), FTP (File Transfer Protocol), Gopher and WAIS (Wide Area Information Server) protocols. In this way, Hyper Text Transfer Protocol allows basic hypermedia access to resources available from different applications. HTTP also uses the request/response model of communication in which a client sends a request to a server using a request method, URI (Uniform Resource Identifier), and protocol version preceding a MIME-like message that contains, among other things, client information. The server then replies with a status line, in addition to the message's protocol version and a success or error code, preceding a MIME-like message containing, among other things, server information. [5]

2.2. Rest

Epresentational State Transfer (REST) is an architectural style for networked systems and not protocol-dependent, though most practical implementations are built on HTTP. Moreover, REST is not a standard but prescribes the use of standards such as XML (for resource representation in Extensible Markup Language which is both human-readable and machine readable), TEXT/XML (for content type in text form encoded in Extensible Markup Language), HTTP and URL.

2.2.1. Design Principles of a REST-based System

The communication between a client and a server must be stateless, in the sense that information known to the server must not be used but rather a request from a client must include all the information needed by the server to respond appropriately. Also, there should be a uniform interface that supports state transfer and comprising of a constrained set of properly defined operations (such as HTTP methods GET, PUT, POST and DELETE) and a constrained set of content types (such as TEXT/XML and IMAGE/JPEG. Furthermore, there should be Client-server pull interaction with client pull representations. Also, resources should be uniquely named by the use of URI (Uniform Resource Identifier). In addition, there should be layered, interconnected resource representations by the use of URLs (Uniform Resource Locators) in order to make progression through states by clients possible. Last but not least, there should be cacheable responses in order to make the network operate efficiently.

2.2.2. REST Architecture

Below is a REST-based architecture for client-server interaction.

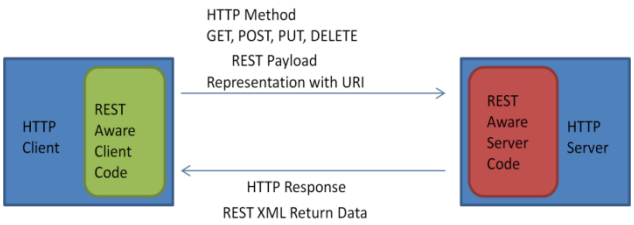

Figure 1. REST-based architecture for Client-Server interaction

As shown above, HTTP is used as the uniform interface and the resource representations are shared over this interface. On the client side, the REST aware client code is usually part of a web page, loaded from a web server. This client is commonly written using JavaScript and embedded into HTML for manipulating resources. The client side code also parses and acts upon data returned from the server side based on the logic of the application.

Furthermore, the architecture above shows how different underlying standards such as HTTP, URI, HTML and XML are used in supporting a REST based architecture. [8]

2.3. CoAP

Constrained Application Protocol (CoAP) [1] is a specialized transfer protocol for the web and used by constrained networks and nodes in machine-to-machine applications like control of energy consumption through smart energy metering and automated systems.

Limited processing power and storage space are some of the characteristic or features of constrained nodes. Constrained networks are usually lossy networks with high rates of packet error and a limited throughput of up to tens of kilobits per second. 6LoWPAN (IPv6 over Low power Wireless Personal Area Network) is a typical example of such a network.

CoAP has an interaction model identical to the client-server model of interaction of HTTP (Hyper Text Transfer Protocol) but the two low power interacting machines or devices require a CoAP implementation in both ends of the communication or interaction. The client first sends a resource request which is represented by a URI (Uniform Resource Identifier) on the server through method codes similar to that of HTTP. A response from the server also has a response code identical to that of HTTP and may come along with a resource representation. The response codes of CoAP are usually represented as "2.xx", "4.xx" or "5.xx" whereas those of HTTP are "2xx", "4xx" or "5xx". As a result, one could do CoAP-to-HTTP mapping or HTTP-to-CoAP mapping.

CoAP depends on the REpresentational State Transfer (REST) architecture of the web. However, CoAP interaction is asynchronous through UDP and logically done via a layer ofmessages supporting optional reliability with the use of exponential back-off. Moreover, CoAP has four message types and they could be requests or responses depending on the method codes and response codes they may carry. The message types are "Confirmable", "Non-Confirmable", "Acknowledgement" and "Reset". Also, CoAP is a single protocol having messaging and requests or response as features of its header. Figure 2 below shows the abstract layering of CoAP.

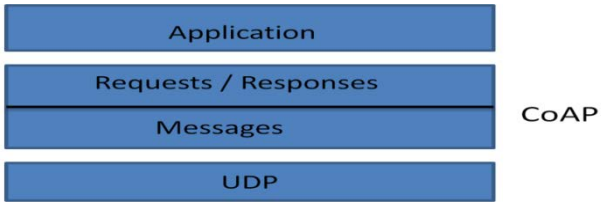

Figure 2. Abstract layering of CoAP [1]

2.4. RPL

IPv6 Routing Protocol for Low power and Lossy Networks (RPL) [1] is a routing protocol which satisfies the requirements of Low power and Lossy Networks (LLNs). LLNs are made up of nodes constrained in terms of limited memory/storage, processing power and energy, if they are being powered by dry cells (batteries). Moreover, the connections between routers in these networks are lossy with low data and packet delivery rates.

Furthermore, the traffic patterns could be point-to-point (least dominant flow), point-to-multipoint Or multipoint-to-point (most dominant flow) with over thousands of nodes in the networks.

2.5. Summary

In the web, such as HTTP, there is plethora of applications that could be implemented.

However, in a restricted environment, such as LLN, the limited resources must be efficiently utilized to ensure the proper and continuous functioning of the network. Hence, the use of light-weight protocols such as CoAP (constrained applications) and RPL (routing) in such networks, based on the REST architecture.

As these protocols are designed to support an "Internet of Things", we investigated their suitability in practice in the remainder of this paper and compared different modes of operations.

3. Methodology

3.1. Introduction

The operating system used for this study is the Contiki OS. The Contiki OS is the open source operating system for the "Internet of Things". It runs on networked embedded systems and WSNs. This OS (Operating System) provides IP communication, both for IPv4 and IPv6, and its uIPv6 stack (fully tested) is IPv6 Ready Phase 1 certified. Furthermore, its power-efficient radio mechanisms such as ContikiMAC, make it possible for battery-operated devices to participate in IPv6 networking. Also, Contiki supports 6lowpan header compression, IETF.

RPL IPv6 routing, IETF CoAP application layer protocol, and many other protocols. It is written in the C programming language and has an event-driven kernel, on top of which application programs can be dynamically loaded and unloaded at run time. Its processes are lightweight and so provide a linear, threadlike programming style on top of the event-driven kernel. In addition, it supports per-process optional multithreading and inter-process communication using message parsing. Contiki also provides a regular "malloc()" operation, memory block allocation and a managed memory allocator for memory management. [3]

The device used for building the wireless sensor network was the Redbee-Econotag and below is a picture of it.

It is a Freescale MC13224v ARM7 microcontroller with, among others, 802.15.4 radio, integrated bootloader, PCB antenna (open-air line-of-sight range approximately 500 ft at 0.

dBm; full transmit power is 4.5 dBm), 24 MHz crystal, 2 Light-Emitting-Diodes for general purpose or for receiving and transmitting data, and flash erase jumpers. [4]

The experiments were carried out to investigate the differences between the PULL and PUSH protocols in a constrained or restricted environment, and to select the better protocol, if applicable. The protocols studied were RPL (IPv6 Routing protocol for low power and Lossy Networks – for routing), CoAP (Constrained Application Protocol - for the PULL mechanism) and the usual uIPv6 UDP (for the PUSH mechanism). In the experiments, ten RedBee-Econotags (Freescale MC13224v ARM7 microcontroller with 802.15.4 radio) were organized or setup in six different topologies, loaded with a testbed and nine different traffic rates applied to each topology. The six topologies were selected or chosen because they were the most reasonable and appropriate setups for the limited number of nodes available.

Furthermore, the chosen topologies provided initial cues for the relative behavior of the two protocols and for the initial bits scale (especially in cases where things go wrong). These experiments were the same for both protocols and the results or data collected saved to a file with timestamps. Detailed descriptions of all these can be found in the next sections.

3.2. Evaluation Methodology

The results gathered were then evaluated by comparing the latency of the data collected during the least traffic rate to the remaining eight traffic rates. This was to show the effect of increasing traffic rate on latency. Also, the stability of the network was observed as the traffic rate was incremented. These were done for both protocols.

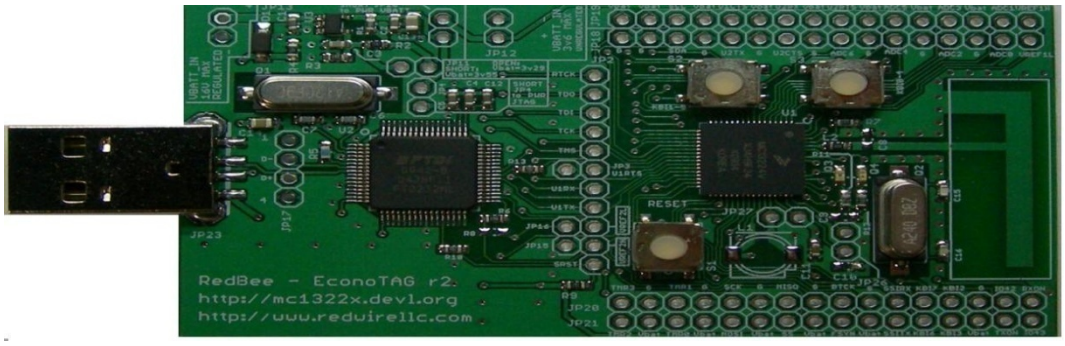

Figure 3. Image of the Redbee-Econotag

3.3. Building and Using the Testbed

The testbed comprises of the server side and the client side. For the server side, a UDP server and a REST server were combined into one system and configured over-the-air. The over-the-air configuration was used only by the PUSH protocol, since the PULL protocol did not need any configuration due to its request-reply model.

In addition, both protocols had the same size of data to send and it was 12 bytes in all cases.

On the other hand, the clients were separated due to the differences in their mode of communication. Also, the UDP client was what was used to send the configuration parameters and control messages for stopping and restarting the tests. Detailed descriptions of all these can be found in chapters 4 and 5.

3.4. The Testbed

This chapter describes the components of the testbed used for the tests. The experiments were carried out to investigate the differences between the PULL and PUSH protocols in a constrained or restricted environment, and to select the better protocol, if applicable. In the experiments, ten RedBee-Econotags (Freescale MC13224v ARM7 microcontroller with 802.15.4 radio) were organized or setup in six different topologies, loaded with a testbed and nine different traffic rates applied to each topology. The testbed comprises of a combined normal UDP server and REST server with automated clients. In this system, the UDP client is used for sending the configuration parameters (string) to the combined UDP and REST servers but only used by the UDP server (PUSH protocol), since CoAP does not need any configuration parameters because it is a PULL protocol.

For the UDP client-server communication, the client first sends the configuration parameter to the server and then a user-button on the client device is pressed to start the simulation. In addition, the user-button can also be used to stop and restart the tests. The server then sends the results or test data to the client for storage in a file, with timestamps.

For the CoAP client-to-REST server communication, the client sends resource requests to the server at specified times, equal to that of the UDP, and stores the received replies in a file with timestamps. In addition, both protocols had the same size of data to send and it was 12 bytes in all cases.

The results gathered were then evaluated by comparing the latency of the data collected during the least traffic rate to the remaining eight traffic rates. This was to show the effect of increasing traffic rate on latency. Also, the stability of the network was observed as the traffic rate was incremented.

3.5. Test Cases (Setups)

This chapter shows the different test cases used in this thesis. The comparative test cases were six in number (because these were the most reasonable setups that could be constructed from them due to limited number of nodes available), in which different setups were investigated to see how the two different protocols, normal udp client-server (sender-sink) and CoAP client-server (CoAP client– REST server), behave with respect to the rate (frequency) of traffic within different network topologies. The rate (frequency) of traffic was from 10 seconds to 5 milliseconds and each run lasted for an hour.

Also, each traffic rate for a test case was repeated three times. The results will be presented in the next chapter. Below is an overview table of the components involved in each test case.

4. Tables of Results and Analysis

This section presents the results of the test cases outlined in the previous section, starting from the first use case.

From Table 2, it could be seen that the PUSH protocol (UDP) has a delay of 1 ms whereas the PULL protocol (CoAP) has an RTT delay of 6 ms when the traffic rate is greater than 6 ms and 8 ms when traffic rate is a millisecond less than its usual RTT. This means that the PUSH protocol, in such a setup, can accommodate a traffic rate of 1 ms without any problem and as a result the sink can receive a total of 1000 data points per second. On the other hand, the PULL protocol, in such a setup, can also handle a maximum traffic rate of 6 ms without any effect on its usual RTT and a total of 166 data points per second could be gathered.

However, in the case of the PULL protocol, a traffic rate of 5 ms would result in a total of 125 data points per second (41 data points lesser than the usual case).

Table 1. Test Cases

Use Cases	Number of Senders or Clients	Number of Sinks or Servers	Number of RPL Routers	Traffic Rates
Case 1	1	1	2	10, 5, 3, 2 and 1 (seconds), 50, 20, 10 and 5 (milliseconds)
Case 2	1	1	8	Same as previous
Case 3	3	1	6	10, 5, 3, 2 and 1 (seconds), 5 (milliseconds)
Case 4	3	3	4	Same as previous
Case 5	1	9	-	10, 5, 3, 2 and 1 (seconds), 50, 20, 10 and 5 (milliseconds)
Case 6	9	1	-	1 second

Table 2. Use Case 1: One Client to One Server with Two RPL Routers

Rate	10s	5s	3s	2s	1s	50ms	20ms	10ms	5ms
CoAP (RTT delay / ms)	6	6	6	6	6	6	6	6	8
UDP (delay / ms)	1	1	1	1	1	1	1	1	1

Table 3. Use Case 2: One Client to One Server with Eight RPL Routers

Rate	10s	5s	3s	2s	1s	50ms	20ms	10ms	5ms
CoAP (RTT delay / ms)	6	6	6	6	6	6	6	6	8
UDP (delay / ms)	1	1	1	1	1	1	1	1	1

Also, the same effect as seen and explained in Table 2 further above applies to Table 3 above because the two scenarios are similar with respect to the source and destination (one-to-one mapping).

Table 4. Use Case 3: Three Senders to One Sink with Six RPL Routers and Three Clients to One Server with Six RPL Routers

Rate	10s	5s	3s	2s	1s	5ms
CoAP (RTT delay / ms)	6	6	6	6	6	(8 – 9)*
UDP (delay / ms)	1	1	1	1	1	1*

Table 4 above has the same results as in Table 2 and Table 3 for traffic rates from 10 seconds to 1 second. However, for a rate of 5ms, only the UDP sender which started sending first worked and also the CoAP client that started requesting for resources first worked. Furthermore, when that UDP sender was stopped another sender then took over the sending and the same applied to the CoAP client. The reason for this behavior may be that the 5 ms traffic rate is really fast and so the other nodes did not get the chance to really initiate any operation once a node is in operation (sending or requesting) or their data just got lost. Due to this effect, the PULL protocol had RTT delay of 8 to 9 milliseconds whereas the PUSH protocol had its usual 1 millisecond delay.

Table 5. Use Case 4: Three Senders/Clients to Three Senders/Servers with Four RPL

Rate	10s	5s	3s	2s	1s	5ms
CoAP (RTT delay / ms)	6	6	6	6	6	8*
UDP (delay / ms)	1	1	1	1	1	--

Also, Table 5 above has the same results as in Tables 2, 3 and 4 above for traffic rates from 10 seconds to 1 second. However, for a rate of 1 second, only one of the UDP Senders worked at first when the three sending nodes were started but when they were all stopped and restarted, they all started working as expected. This test was done several times, as done for all, but the same behavior recurred. Also, for a rate of 5ms, none of the UDP Senders worked but only the

CoAP client that started requesting for resources first worked at all times with an RTT delay of 8 milliseconds. The results were surprising since all the sources (senders and clients) had dedicated destinations (sinks and servers) and so should not have had any conflict. Maybe, this behavior was due to network instability as a result of the high traffic rate (5ms). In addition, in the case of the PULL protocol, it seems the client-server pair that starts interacting before the rest takes control of the network and thereby prevents the others from communicating.

Table 6. Use Case 5: One Client to Nine Servers (CoAP) and Use Case 2: One Client to One Server with Eight RPL Routers (UDP)

Rate	10s	5s	3s	2s	1s	50ms	20ms	10ms	5ms
CoAP (RTT delay / ms)	6	6	6	6	6	6	6	6	8
UDP (delay / ms)	1	1	1	1	1	1	1	1	1

Table 6 above also had the same results as recorded in Table 3 further above. This is expected because both scenarios are similar in their operation.

Table 7. Use Case 6: Nine Senders to One Sink (UDP) and Nine Clients to One Server (CoAP)

Rate	1s
CoAP (RTT delay / ms)	6 - 7
UDP (delay / ms)	1 - 2

Finally, Table 7 above resulted in RTT delay of 6 to 7 milliseconds for the PULL protocol whereas the PUSH protocol had delay of 1 to 2 milliseconds. A reason which could be assigned to this behavior is that the increased number of nodes (senders or clients up to 9) created the extra 1 millisecond delay in both protocols. The 1 second traffic rate was the only rate used in this scenario because the lower rates (10, 5, 3 and 2 seconds) would have resulted in a normal delay as seen in the others and the higher rates (50, 20, 10 and 5 milliseconds) would have resulted in the behavior experienced in test case 3.

5. Discussion

From the results and analysis, it was very clear that the PULL protocol performed normally in most of the cases and also had better performance than the PUSH protocol even in cases where the network was loaded with enormous amount of traffic in a short space of time.

Furthermore, the PULL protocol having an RTT of 6 milliseconds in normal cases as compared to a delay of 1 millisecond for the PUSH protocol is very reasonable because of its request-reply communication model.

6. Conclusion

It was very clear that the PULL protocol performed normally in most of the cases and also had better performance than the PUSH protocol even in cases where the network was loaded with enormous amount of traffic in a short space of time.

In addition, the PULL protocol having an RTT of 6 milliseconds in normal cases as compared to a delay of 1 millisecond for the PUSH protocol is very reasonable because of its request-reply communication model.

However, in a network of many clients/senders to many servers/sinks respectively, both protocols performed poorly when the traffic rate was very high (5 ms), though the performance of the PULL protocol was better than that of the PUSH protocol. This could mean that, in such a network (which is usually the case in real wireless sensor networks), the traffic rate should not be too high but about a maximum of 1 second. Also, the abnormal behavior of the two protocols in this situation could mean that these protocols may not be able to withstand very heavy traffic

rates or may not function well in very busy wireless sensor networks.

So, with the aforementioned observations, it seems the PULL protocol (CoAP) performed better than the PUSH protocol (normal UDP). Moreover, future wireless sensor networks are more likely to use CoAP in their operations.

In the end, the purpose of this work, which was to compare the PULL and PUSH protocols used in wireless sensor networks and select the better protocol, was achieved or accomplished.

The total number of nodes used was just 10 and so it would be difficult to generalize the results of this research. Therefore, to have results which could be applied to real wireless sensor networks, a total of about 100 nodes should be used.

References

[1] Geoffrey Werner- Allen, Patrick Swieskowski, Matt Welsh (2005). Motelab: A Wireless Sensor Testbed. *IPSN '05 Proceedings of the 4*th *International Symposium on Information, Article No. 68* http://dl.acm.org/citation.cfm?id=1147685.1147769 on 20.02.2012.

[2] Jennifer Yick, Biswanath Mukherjee, Dipak Ghosal (2008). Wireless Sensor Network Survey. *Department of Computer Science, University of Carlifornia, Davis, CA 95616, United States* http://www.sciencedirect.com/science/article/pii/S1389128608001254 on 21.02.2012.

[3] Adam Dunkels, Oliver Schmidt, Niclas Finne, Joakim Eriksson, Fredrik Österlind, Nicolas Tsiftes, Mathilde Durvy (2012). The Contiki OS: The Operating System for the Internet of Things. http://www.contiki-os.org/p/about-contiki.html on 1.7.2011.

[4] Redwire Redbee Econotag: The Perfect Development board for the Freescale MC13224v ARM7 microcontroller and 802.15.4 radio. http://www.redwirellc.com/store/node/1 on 1.7.2011.

[5] R. Fielding (UC Irvine), J. Gettys (Compaq/W3C), J. Mogul (Compaq), H. Frystyk (W3C/MIT), L. Masinter (Xerox), P. Leach (Microsoft), T. Berners-Lee (W3C/MIT) (1999). Hypertext Transfer Protocol – HTTP/1.1. *IETF – Network Working Group*. http://www.ietf.org/rfc/rfc2616.txt on 3.2.2012.

[6] N. Freed (Innosoft), N. Borenstein (First Virtual) (1996). Multipurpose Internet Mail Extensions. *IETF – Network Working Group*. http://www.mhonarc.org/~ehood/MIME/2045/rfc2045.html on 3.2.2012.

[7] Michael Richardson, J. P. Vasseur, Adrian Farrel, Rene Struik, Daniel King. Routing Over Low power and Lossy networks (roll). *IETF-Network Working Group*. http://datatracker.ietf.org/wg/roll/charter/ on 4.2.2012.

[8] K. Griffin, J. Rosenberg (Cisco) (2008). Representational State Transfer (REST) for Feature Configuration in Session Initiation Protocol (SIP). *IETF-BLISS*. http://tools.ietf.org/html/draft-griffin-bliss-rest-00#section-1 on 4.2.2012.

[9] Z. Shelby (Sensinode), K. Hartke, C. Bormann (Universitaet Bremen TZI), B. Frank (SkyFoundry) (2012). Constrained Application Protocol (CoAP). *IETF-CoRE Working Group.* http://datatracker.ietf.org/doc/draft-ietf-core-coap/ on 13.3.2012.

[10] T. Winter (Ed.), P. Thubert (Ed. – Cisco Systems), A. Bradt (Sigma Designs), T. Clausen (LIX, Ecole Polytechnique), J. Hui (Arch Rock Corporation), R. Kelsey (Ember Corporation), P. Levis (Stanford University), K. Pister (Dust Networks), R. Struik, JP. Vasseur (CiscoSystems) (2011). RPL: IPv6 Routing Protocol for Low power and Lossy Networks. *IETF-ROLL*. http://tools.ietf.org/html/draft-ietf-roll-rpl-19 on 14.3.2012.

[11] Mariano Alvira. MC1322x Microcontroller Developers Mailing List http://devl.org/pipermail/mc1322x/ on 5.7.2011.

[12] Mariano Alvira. Contiki REST/CoAP Quickstart using Econotags http://mc1322x.devl.org/repos/contiki-mc1322x/cpu/mc1322x/doc/rest-tutorial.md on5.7.2011.

[13] Mariano Alvira. Contiki RPL Quickstart using Econotag http://mc1322x.devl.org/repos/contiki-mc1322x/cpu/mc1322x/doc/rpl-tutorial.md on 6.7.2011.

[14] How to split a string on the comma to get an array with 3 strings using C Programming http://cboard.cprogramming.com/c-programming/128736-how-split-string-comma-get-array3-strings-using-c-programming.html on 20.10.2011.

[15] Zolerita (2011). Lesson 1: A first app, knowing Contiki API, timers and blocking conditions http://zolertia.sourceforge.net/wiki/index.php/Mainpage:Contiki_Lesson_1 on 20.10.2011.

[16] Zolerita (2011). Lesson 0: Contiki basics, filesystem and compile instructions http://zolertia.sourceforge.net/wiki/index.php/Mainpage:Contiki_Lesson_0 on 20.10.2011.

[17] Chee-Yee Chong (IEEE member), Srikanta P. Kumar (Senior IEEE member) (2003). Sensor Networks: Evolution, Opportunities, and Challenges.

[18] Clare, Loren P., Gregory J. Pottie, and Jonathan Agre (1999). "Self-Organizing Distributed Sensor Networks." Proc. SPIE Aerosense99.

[19] Dargie, W. and Poellabauer, C., "Fundamentals of wireless sensor networks: theory and practice", John Wiley and Sons, 2010 ISBN 978-0-470-99765-9, pp. 168-183, 191-192.

[20] Sohraby, K., Minoli, D., Znati, T. "Wireless sensor networks: technology, protocols, and applications, John Wiley and Sons", 2007 ISBN 978-0-471-74300-2, pp. 203-209.

[21] Garcia P., "A Methodology for the Deployment of Sensor Networks", IEEE Transactions On Knowledge And Data Engineering, vol. 11, no. 4, December 2011.