Encoding and Compression for the Devices Profile for Web Services

Guido Moritz[1], Dirk Timmermann[1], Regina Stoll[2]
University of Rostock
[1] Institute of Applied Microelectronics and CE
[2] Institute of Preventive Medicine
18055 Rostock, Germany
guido.moritz@uni-rostock.de

Frank Golatowski
Center for Life Science and Automation
CELISCA
18119 Rostock, Germany
frank.golatowski@celisca.de

***Abstract*—Most solutions for Wireless Sensor Networks (WSN) come equipped with their own architectural concepts which raise the problem of possible incompatibility of computer networks and the WSN. Often gateway concepts are used to overcome this problem. But this is not the best solution on the long term. Other research fields and industrial domains are heading for universal cross domain architecture concepts based on internet technologies that are more mature and better understood. The IETF 6LoWPAN working group provides the groundings for standardized communication using existing network protocols like IPv6 also in low power radio networks. A big challenge when deploying further application layer network protocols on top of 6LoWPAN is the message size of existing - mostly XML based - protocols which does not meet the resource requirements of deeply embedded devices without further research efforts. This paper presents different data compression techniques for the Devices Profile of Web Services (DPWS) to be applied in 6LoWPAN networks. Therefore, we analyze a realistic scenario. We determined 18 message types in the scenario and compressed and encoded all messages by using existing schemes and tools. For the first time, we also investigate on the Efficient XML Interchange (EXI) format for DPWS.**

SOAP; Compression; Encoding; Web Sevices for Devices; DPWS; Devices Profile for Web Services

I. INTRODUCTION

The deployment of matured and well-known internet protocols like IP, TCP, and UDP is a proper approach to overcome interoperability problems of networking devices. Especially deeply embedded devices, with very limited resources like computing power, power supply, and only tens of kB RAM and ROM, were excluded from this standardized communication due to missing tailored concepts, standards, and implementations. Thus, the Internet Engineering Task Force (IETF) has established the 6LoWPAN working group [1] in accordance to the IPv6 specification. The focus of 6LoWPAN is to compress IPv6 headers to be sent on top of IEEE 802.15-based technologies, especially IEEE 802.15.4 [2]. Thereby 6LoWPAN establishes the basis for TCP and UDP data transmissions in battery powered wireless sensor networks, driven by deeply embedded devices with tens of kB RAM and ROM. These deeply embedded devices are the target platforms of this paper.

But the applicability of application layer protocols that meet the resource constraints is still a big challenge. High research efforts are made to develop cross domain communication middleware basing on architectural concepts like REST (Representational state transfer) and Service-oriented Device Architectures (SODA) [3] and on technologies like UPnP (Universal Plug and Play), JINI, and DPWS (Devices Profile for Web Services). In August 2008 a technical committee (TC) at OASIS was formed for the "Web Services Discovery and Web Services Devices Profile" (WS-DD) [5]. WS-DD standardizes the lightweight subset of the Web services protocol suite that makes it easy to find, share, and control devices on a network. The work of this TC is based on the former DPWS, WS-Discovery and SOAP-over-UDP specifications. Focus of this paper is to apply DPWS also in 6LoWPAN networks.

However, to deploy DPWS on deeply embedded devices, it is necessary to find solutions for message size reduction by a tailor made data compression or encoding. This paper presents a wide survey and comparison of existing techniques. In contrast to existing investigations, this paper uses messages taken from a realistic deployment scenario of DPWS messaging and includes the evaluation of the Efficient XML Interchange (EXI) encoding and Fast Infoset (FI) in schema-informed mode.

II. MESSAGE FORMAT AND HTTP COMPRESSION

The Devices Profile for Web Services (DPWS) bases on well-known protocols and adds several extensions to enable Web services based communication for embedded devices. In accordance to client/server architecture as widely used by web applications, DPWS also specifies two different roles: *devices* and *clients*. A DPWS device hosts zero or more *Hosted Services* and one *Host Service*. The *Host Service* represents the *device* itself and announces the *Hosted Services* which are the endpoint services. Every *Hosted Service* may have miscellaneous operations. A core feature of DPWS is the automatic discovery of devices and their *Hosted Services,* even in a dynamic changing environment. The complete discovery process is described more detailed in [6].

In general, DPWS requires two different transport patterns. During the discovery phase, messaging requires group addressing (multicast). This allows DPWS *devices* to announce their presence in or their leaving of a network and DPWS *clients* to discover unknown *devices* without any specific knowledge about network and/or *devices* infrastructure. Because of the absence of a direct end-to-end communication, multicast messages are carried by the UDP transport protocol. In contrast to the messaging during discovery phase, DPWS requires direct end-to-end communication pattern between *clients* and *devices*, e.g., for service invocation and eventing.

For data transport and representation, the SOAP 1.2 protocol [7] is applied. SOAP in turn uses XML based data representation and XML Schema [8] for encoding and document structuring. The SOAP envelopes can be carried by various underlying protocols (binding). The SOAP specification defines the SOAP-over-HTTP binding [9]. The HTTP protocol in turn is bound to the TCP protocol. Because the former described non-discovery specific messages in DPWS need direct end-to-end communication, for these message types the SOAP-over-HTTP binding is used due to compatibility with existing Web services mostly using SOAP-over-HTTP. For the discovery in DPWS, where UDP multicast is required, SOAP-over-HTTP is not applicable. Hence, a new SOAP-over-UDP binding is specified by the OASIS WS-DD technical committee [10].

In summary of the above described message types, transport schemes and representation formats, a differentiation has to be made between SOAP-over-UDP multicast messages and SOAP-over-HTTP messages carried over TCP, when facing data encoding and compression for DPWS messages. SOAP-over-UDP leaves the SOAP envelope unchanged and the only difference to SOAP-over-HTTP messages is the missing HTTP header (see WS-Discovery [10]). As specified in the SOAP-over-HTTP binding and the according HTTP specification [11], the following HTTP header fields are required in a HTTP request:

- **HTTP Method, URI and Protocol**
  e.g.: POST /{URI} HTTP/1.1
  The first line specifies the HTTP method. All DPWS SOAP-over-HTTP messages are using the POST method. The URI format is defined in [12].
- **Target host transport specific address and port**
  e.g.: Host: {IP address }:{port}
- **Content-Type:** application/soap+xml; charset=utf-8
  All XML SOAP messages require this Content-Type definition.
- **Content-Length:** {Length of SOAP Envelope in bytes}
- **Connection:** {connection-token, e.g. close|keep-alive}
- **SOAPAction:** "{Action URI}"
  This header field is mandatory in the SOAP-over-HTTP binding, but may be empty.

For the corresponding HTTP response, the following HTTP headers are required:

- **HTTP/1.1 {Status Code} {Status Code Meaning}**
  e.g.: HTTP/1.1 200 OK
  The HTTP status codes can be used to identify problems during message parsing and processing, e.g. Additionally, the SOAP envelope contains SOAP specific error messages.
- **Server:** {Server Type/Name}
- **Content-Type:** application/soap+xml; charset=utf-8
  All XML SOAP messages require this Content-Type definition.
- **Content-Length:** {Length of SOAP Envelope in bytes}
- **Connection:** {connection-token, e.g. close|keep-alive}

Additionally to the presented header fields, own header fields for HTTP request and response can be defined for domain or vendor specific extensions.

Most of the required HTTP header fields do not provide additional useful information due to the usage of WS-Addressing in DPWS. Hence, nearly all required information are carried in the SOAP envelope and are at the best redundant, at worst not required in the HTTP header. All DPWS messages use the POST method of HTTP 1.1 and the resource address is carried in the SOAP envelope. The target host address and port can be derived from the transport layer in most implementations. The content type is defined in the SOAP-over-HTTP binding and is stable. Connections can be tokenized as keep-alive, but applications should not rely on this behavior. The SOAPAction field is mandatory, but may be empty. The server HTTP header field is also mandatory, but not analyzed in most implementations. Even in error or fault cases, SOAP envelopes contain information about details and the HTTP status code might be not required. To sum up, out of the vast number of HTTP header fields, the content length in bytes, the identification of the payload format (SOAP envelope and encoding), and in some scenarios the HTTP status codes are required only. This information can be represented in a much more efficient binary encoding as in compliant HTTP headers. This would have a big impact on the HTTP related overhead as described in section IV.

III. SOAP Compression and Encoding Schemes

Currently, no compression schemes for the HTTP protocol and the HTTP header fields are widely used. Within the IETF 6LoWPAN working group, an active document is published describing a specific 6LoWPAN HTTP compression called Chopan - Compressed HTTP Over PANs [13]. Nevertheless, this document is not complete and fully matured in the time of writing this paper. To sum up, only the SOAP envelope part within DPWS messages can be compressed using specific encoding and compression schemes, due to the absence of a standardized HTTP header compression format.

For SOAP compression and encoding, several XML specific and XML non-specific compressors and schemes exist. This paper only concentrates on schemes which are open solutions and which are not protected by a patent or any other legal regulation. Hence, e.g., Adaptive SOAP [14] is out of scope of this paper. Furthermore, the schemes have to be implemented and thus not only a theoretical assumption on the expected compression rate can be made, but measurements of message size in a realistic scenario are possible. Hence, the compression schemes XEBU [16], Exalt [17], and XGrind [18] have to be left out. Some implementations exist, but are not maintained any longer and could not be used for evaluations. In the remainder of this section, potential schemes are briefly described.

A. The WAP Binary XML

The WAP Binary XML (WBXML) [19] format is a binary representation for XML based messages, to allow compression in mobile networks. WBXML is specified by the Open Mobile Alliance and is currently a W3C note. The WBXML format requires previously defined structures and naming for tags and attributes to be included in the WBXML specification. This

makes WBXML not applicable to be used for SOAP compression due to not known message formats and arbitrary naming.

B. *Differnce encoding*

Werner describes in his dissertation [15] a difference encoding format. Out of the service describing WSDL files, skeleton messages are generated. These skeleton messages are embedded at compile time into the implementation. During runtime, only the difference between these skeletons and the full messages are transmitted. For deeply embedded devices with only tens of kB RAM and ROM, difference encoding cannot be applied. During run-time, the comparison of skeleton message and difference document requires too much memory (RAM), to store the messages during processing.

C. *XMLPPM*

The XML-Conscious PPM Compression (XMLPPM) scheme is described by Cheney in [20], [21] and [22]. XMLPPM combines a general algorithm for text compression with a SAX parser and the Prediction by Partial Match (PPM) algorithm. The latter utilizes existing context information while stepping through XML structures (e.g. specific SOAP envelope format including SOAP header and SOAP body).

D. *gzip and bzip2*

In contrast to the other presented compression and encoding schemes, gzip [24] and bzip2 [25] are not specific XML schemes, but are already widely used in computer and network applications, even by other existing HTTP based protocols. Both compressors can be configured to allow the best ratio between compression rate and required computing power. But both algorithms are too heavy weight to be applied on deeply embedded devices due to the leakage of memory and processing power. Nevertheless, in section IV.C measurements of both algorithms will be presented.

E. *XMill*

The XMill XML compressor was developed by Liefke et. al in 1999 [26]. It was the first XML specific compressor that could be used without specific knowledge about the XML document structure. Therefore, the SAX based solution stores events (structural information) and payload (data) in different containers. The structural container items points to specific items in the data container. This omits redundant data storage. Liefke et al. promise a compression rate similar to that one of raw bzip2, while requiring less computing time and reaching similar compression times like the raw gzip algorithm. The XMill scheme itself is not a pure binary format, but can be converted into and thus the document size further reduced by applying XML unspecific algorithms like gzip or bzip2 on the resulting XMill containers.

F. *Efficient XML Interchange (EXI)*

The mission of the Efficient XML Interchange Working Group [27] of the W3C is the development of “a format that allows efficient interchange of the XML Information Set, based on the conclusions of the XML Binary Characterization Working Group”. As result of this working group the EXI format is developed. EXI is the original Efficient XML format of AgileDelta Inc, with a number of added features partly from contributor candidates like Fast Infoset, Xebu, etc.

Because the EXI format is a dedicated XML compressor, it also makes use of structural information and separates between XML structure and payload data. For encoding into the EXI format, string tables are generated. These tables have variable length and indexes to tokenize and identify each occurred string. The bit width of these indexes is also kept variable to allow most compact data format. Other non-string payload data like integer or date/time values are represented by defined built-in binary formats. Recurring structure information fields, e.g., closing tags, are also pre-defined. Because of the strict separation of structure and data elements, after converting the XML document in the EXI format, structure and payload information can be reordered and arranged in a pattern to allow similar structure and data items to be close to each other. This in turn provides the best preconditions to apply compression schemes after re-encoding which exploit repeated occurrence of equal bit strings. For this compression, EXI is bound to the Deflate algorithm [30].

Additionally to the separation of structure and data information and the usage of structure information already provided by XML in general, EXI is capable of assigning further information for dedicated documents and document types. These information are available through XML Schema documents that are available along with XML based protocols like SOAP and further WS-* specifications. The schema documents provide an abstract description of layout of an XML document representing the according protocols. EXI uses standard XML schemas without further mark-ups or annotations to specify additional information used for encoding and decoding. The resulting format is called EXI schema-informed.

A more detailed and comprehensible description of the EXI format can be found in the EXI Primer [28].

During examining EXI in schema-informed mode, we found two major improvements to be mentioned. DPWS points to versions of WS-Addressing and WS-Eventing that are inconsistent by their referring namespaces. For the measurements presented in section IV of this paper, this inconsistency was resolved. Furthermore, DPWS specifies well defined values to be used as XML tag values (e.g., discovery messages in wsa:Action field) or as attribute values. These values can be included in the XML schema files as enumerations, which has a considerable influence on the resulting message sizes.

G. *Fast Infoset*

Fast Infoset (FI) [23] is a standardized binary representation of XML documents. Strings are tokenized in FI at their first occurrence. Redundancies within the strings are omitted because the strings can be referenced among each others. Both concepts are typical for XML specific compressors and similar to EXI.

Additionally, FI also features schema-informed mode. As difference to EXI, in FI it is possible to use schema-informed mode without need for the decompressor to have access to the schema. In this mode, only the data types are encoded in the most compact binary format. FI also is capable of data compression after re-encoding, but is not bound to a specific compressor like EXI.

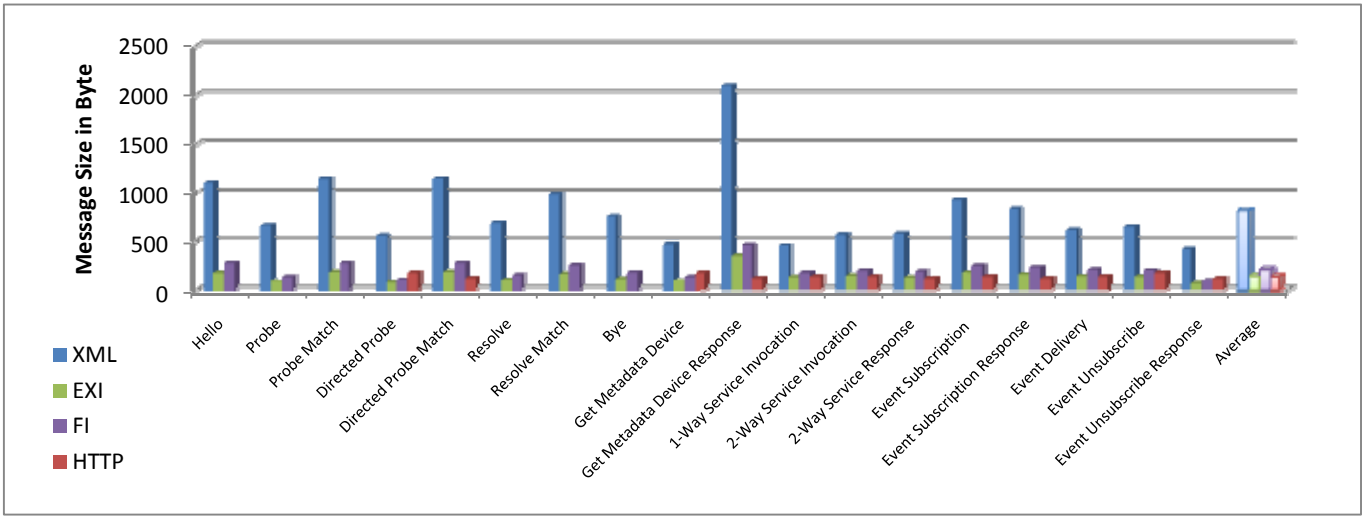

Figure 1. DPWS message sizes in XML, EXI, and FI format

IV. EVALUATION

A. Test Scenario

To evaluate the different existing compression and encoding schemes for SOAP and thus for DPWS messaging, we implemented a test scenario by using our WS4D-gSOAP DPWS toolkit available at [29]. Because measuring other performance parameters than message sizes is out of scope of this paper, we recorded all message types of a scenario and made an offline evaluation. We implemented the air conditioner tutorial example already existing in the WS4D-gSOAP toolkit, because this example provides all possible message types and formats and reflects a realistic scenario to be applied on deeply embedded devices. Deeply embedded devices are not expected to provide complex services and operations. More important than complex services is the integration of this class of deeply embedded devices in an existing networking infrastructure of resource richer devices. Thus, the air conditioner example realizes basic functionalities like invoking operations to read the current temperature or to set a new target temperature to be kept by the air conditioner. This example and the measured message sizes are only one possible realization and are only used to evaluate existing compression schemes. Other implementations and measurements may vary. Scope of these measurements is a qualitative comparison of the former presented schemes and modes.

B. Message type classification

While examining and capturing possible message types and their size, we identified 18 different message types that can be separated in different groups:

- **Discovery**
  Hello, Probe, Probe Match, Directed Probe, Directed Probe Match, Resolve, Resolve Match, Bye
- **Metadata Exchange**
  Get Metadata Device, Get Metadata Device Response
- **Service Invocation**
  1-Way Service Invocation, 2-Way Service Invocation, 2-Way Service Response
- **Eventing**
  Event Subscription, Event Subscription Response, Event Delivery, Event Unsubscribe, Event Unsubscribe Response

The discovery specific messages are characterized by using partly multicast addressing. For metadata exchange a service (not a device) may provide a WSDL file for service description. Providing a WSDL is optional according to the DPWS specification and omitted in our scenario. It is not realistic that a deeply embedded device stores its own WSDL, which requires much memory and is only significant at development time for most Web services toolkits. The service invocations are divided in 1-Way and 2-Way messages whereas only 2-Way message require a response on a request. Within the example it is also possible to use a push eventing delivery mechanism in opposite to pull data by polling the service. Therefore, clients can use a publish/subscribe pattern for eventing in DPWS. For clarity, in the analyses we left out event subscription management specific messages like subscription status requests and lease time management.

C. Measurements

In section II, the general format of DPWS messages consisting of SOAP envelopes which are partly embedded in the HTTP protocol is described. Figure 1 gives an overview about the recorded messages and their size on top of the transport layer, separately for HTTP header and SOAP envelope. Figure 1 also depicts best case compression for EXI and FI for each message. The required header size for TCP, UCP, and IP are not included in these measurements. The 6LoWPAN working group defines specific header compression for IPv6 and UDP headers. A TCP header description is currently not available but imaginable. Because the scope of

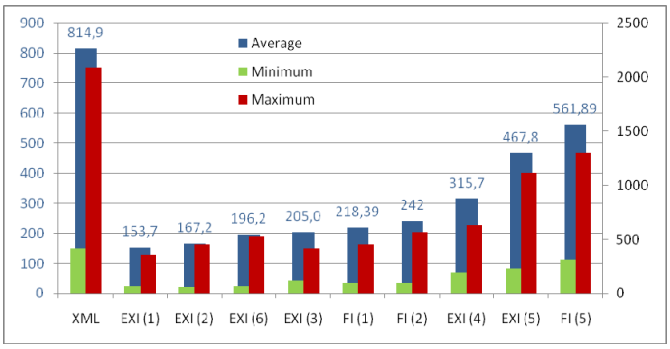

Figure 2. Summary EXI and FI
(numbers in brackets derived from table 1)

this paper is the compression and encoding of DPWS messages on top of 6LoWPAN protocols and the 6LoWPAN header compression schemes vary, all layers below and including the transport layer are not taken into account.

The table 1 presents the overall size of the compressed SOAP envelope excluding the HTTP header. The averages are computed out of all 18 message types. The average HTTP header size of all messages is 147 byte (maximum 184 byte, minimum 128 byte), keeping in mind that HTTP header is not used for discovery messages using the SOAP-over-UDP binding.

The measurements differentiate for EXI and FI between schema-informed and schema-less mode. For schema-informed mode, we used default schema files as published along with the related specifications and additionally optimized versions of these schemas, tailored for DPWS. Because main scope is to investigate on a general compression for DPWS in 6LoWPAN independent of application scenario, for the measurements no additional schema information for the implemented exemplary scenario was used. Furthermore, influence of compression with Deflate after re-encoding is presented. FI is capable of other compressors, but for comparability this paper presents FI only with Deflate. A better compression rate could be achieved using a PPM based compression, resulting in 23% average compression rate. The implementation for FI used for these measurements did not support for not preserving namespace prefixes and for compact binary optimized representation for uuids used, e.g., as message ids in DPWS. Each message id has a size of 45 byte. Binary encoding of uuids would have a considerable influence on the resulting message size. A comparison of EXI and FI in the different modes is depicted in figure 2.

Of particular importance is the influence of the Deflate compression for EXI and FI. Deflate uses a window to reference already occurred strings/data in the message. The window size is by default defined to 32kB in [30]. Because the resulting message size is far below 32kB, the complete message most be cached while parsing and requires additional memory. Using schema-informed mode with optimized schema files, usage of Deflate has a minor influence on compression rate for both EXI and FI.

The results show that there is no significant difference between all compressors exclusive Fast Infoset and EXI. Unexpected is the equal compression rate for bzip2 as the compression factor has no influence on the resulting size. A

TABLE I. MESSAGE SIZE (SOAP ONLY)

Compressor	Average in byte	Average in %	Minimum in byte	Maximum in byte
EXI [1]	153,72	19,60	66,00	354,00
EXI [2]	167,22	20,60	55,00	452,00
EXI [6]	196,17	24,13	66,00	533,00
EXI [3]	205,00	26,25	119,00	414,00
Fast Infoset [1]	218,39	27,98	103,00	455,00
Fast Infoset [2]	242,00	30,09	97,00	563,00
EXI [4]	315,67	40,31	192,00	630,00
XMLPPM	425,22	55,16	274,00	749,00
gzip (C=9)	425,56	55,66	297,00	755,00
gzip (C=1)	437,44	56,99	300,00	799,00
Xmill (C=9)	459,39	59,78	303,00	824,00
Xmill (C=1)	463,72	60,18	304,00	852,00
EXI [5]	467,77	59,64	234,00	1118,00
bzip2 (C=1)	474,78	61,82	315,00	852,00
bzip2 (C=9)	474,78	61,82	315,00	852,00
Fast Infoset [5]	561,89	69,70	315,00	1301,00
XML	814,89	100,00	418,00	2089,00

[1]schema-informed (optimized) / compression with Deflate
[2]schema-informed (optimized) / without compression
[3]schema-informed (default) / compression with Deflate
[4]schema-less / compression with Deflate
[5]schema-less / without compression
[6]schema-informed (optimized) / without compression / byte aligned

reason for this behavior could not be found. The similarity of the resulting size of most of the compressors and formats can be ascribed to the specific test scenario. Deployment of DPWS on deeply embedded devices, which is the main scope of this paper, results in simple and lightweight services and thus in simple message structures. In the messages only few repeated strings occur, which makes the compressors result in nearly the same values. Non-XML specific compressors like gzip and bzip2 also result in nearly the same compression rate, but require much more resources.

For completeness, we also analyzed the influence of a specific byte alignment option of EXI. The EXI format uses a variable string table bit width. Most systems are optimized to handle data in multiples of 8 bit. Thus, and for debugging purposes, EXI allows the usage of a byte alignment option. The string tables and all other structural information have a fixed width of multiples of 8 byte in this mode and thus the payload data can be processed easier. We analyzed this byte alignment option with respect to our test scenario in schema-informed mode with optimized schema files. Because of the above discussed minor influence of Deflate and because focus of applying byte alignment is to reduce parsing complexity, the following values do not include Deflate compression. The overall byte aligned messages in schema-informed (optimized) mode without compression are on an average 3.5% bigger than non byte aligned in the same mode. Hence, using byte aligned mode to reduce parsing efforts is a reasonable solution.

Certainly the usage of all XML schema definitions on deeply embedded devices is a big challenge. The size of all used schemas is 27,2kB in XML format. This is far too much do be embedded on deeply embedded devices directly. The schemas have to be converted into a binary format at compile time to reduce memory consumption. Additionally, the performance requirements of EXI and FI, including and excluding usage of schemata, have to be analyzed. These

analyses where not possible due to missing implementations which meet the memory requirements of the target platforms.

V. Conclusion and Future Work

This paper presents different XML specific and XML non-specific compressors and their influence on message size of the Devices Profile for Web Services (DPWS). Therefore a test scenario was analyzed with 18 different messages occurring in the test scenario. Main scope of the paper is the SOAP compression to make DPWS applicable for deeply embedded devices in 6LoWPAN network, which are characterized by minimal resources like computing power, limited power supply, and few tens of RAM and ROM. The results show that most existing compressors suffer from the simplicity of XML structures which are the results of non-complex services deployed on the deeply embedded device. Only the Efficient XML Interchange (EXI) and Fast Infoset (FI) format provides a much better compression rate, because of the usage of XML schema definitions to include further structure information. Usage of compression after re-encoding has a minor influence.

Further efforts most include performance evaluations and resource requirements analyses of the presented schemas, whereby main focus should be on EXI and FI.

Acknowledgment

This work has been achieved in the ITEA2 project uSERVICE and OSAmI and has been funded by the German Federal Ministry of Education and Research under contract numbers 01|S0902F and 01|S08003I.

We would like to thank Noemax Technologies Ltd. (http://www.noemax.com) for their considerable contribution regarding Fast Infoset.

References

[1] IETF, IPv6 over Low power WPAN (6lowpan), Technical report, http://tools.ietf.org/wg/6lowpan/, 2008.

[2] IETF Network Working Group, Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC 4944, http://tools.ietf.org/html/rfc4944, 2008.

[3] Deugd, S. d., Carroll, R., Kelly, K., Millett, B., and Ricker, J., "SODA: Service-Oriented Device Architecture," IEEE Pervasive Computing, vol. 5, no. 3, 2006, pages 94-C3.

[4] (SOCNE 2008), Ginowan, Okinawa, Japan, 2008, pages 1381-1386.

[5] OASIS Web Services Discovery and Web Services Devices Profile (WS-DD) TC, www.oasis-open.org/committees/ws-dd/, (2009)

[6] Guido Moritz, Elmar Zeeb, Steffen Prüter, Frank Golatowski, Dirk Timmermann, Regina Stoll, "Devices Profile for Web Services in Wireless Sensor Networks: Adaptations and Enhancements," IEEE 14th International Conference on Emerging Technologies and Factory Automation (ETFA2009), In Proceedings in, La Palma, Spain, 2009.

[7] World Wide Web Consurtium (W3C) Recommondation, SOAP Version 1.2, Online, http://www.w3.org/TR/soap/, 2007.

[8] World Wide Web Consurtium (W3C) Recommondation, XML Schema, Online, http://www.w3.org/XML/Schema, 2004.

[9] World Wide Web Consurtium (W3C) Recommondation, SOAP Version 1.2 Part 2, Online, http://www.w3.org/TR/2007/REC-soap12-part2-20070427/, 2007.

[10] OASIS Web Services Discovery and Web Services Devices Profile (WS-DD) TC, Online, http://www.oasis-open.org/committees/ws-dd, 2009.

[11] R.Fielding, et al, Hypertext Transfer Protocol .- HTTP/1.1, http://www.ietf.org/rfc/rfc2616.txt, IETF RFC 2616, 1999.

[12] Berners-Lee, T., Fielding, R. and L. Masinter, Uniform Resource Identifiers (URI): Generic Syntax and Semantics, RFC 2396, 1998.

[13] Frank, B., Chopan - Compressed HTTP Over PANs, IETF Networking Group draft, 2009.

[14] Rosu, M.-C., "A-SOAP: Adaptive SOAP Message Processing and Compression", IEEE International Conference on Web Services (ICWS2007), pp.200-207, 2007.

[15] Christian Werner, „Optimierte Protokolle für Web Services mit begrenzten Datenraten," Dissertation, Logos Verlag Berlin, ISBN 978-3-8325-1409-9, 2007

[16] Kangasharju, J., Tarkoma, S., Lindholm, T., "Xebu, A Binary Format with Schema-Based Optimizations for XML Data," International Conference on Web Information Systems Engeneering (WISE2005), In Proceedings on, New York City, New York, USA, 2005, pp. 528–535

[17] Toman, V., "Syntactical Compression of XML Data," International Conference on Advanced Information Systems Engineering (CAiSE2004), In: Proceedings on, Riga, Lettland, 2004.

[18] Tolani, P., Haritsa, J. R., "XGRIND: A Query-friendly XML Compressor," International Conference on Data Engineering (ICDE2002), In: Proceedings on, San Jose, California, USA, 2002, pp. 225–234.

[19] World Wide Web Consortium (W3C) Member Submission, WAP Binary XML Content Format. Online, http://www.w3.org/1999/06/NOTE-wbxml-19990624/, 1999.

[20] Cheney, J., "Compressing XML with multiplexed hierarchical PPM models," Data Compression Conference, 2001. Proceedings. DCC 2001. , vol., no., pp.163-172, 2001.

[21] James Cheney, "Tradeoffs in XML Database Compression," Data Compression Conference, pp. 392-401, Data Compression Conference (DCC'06), 2006.

[22] James Cheney, "An empirical evaluation of simple DTD-conscious compression techniques," Eighth Workshop on the Web and Databases (WebDB 2005), In Proceedings on, pages 43-48, 2005.

[23] International Telecommunication Union (ITU), Recommendation X.891, Generic Applications of ASN.1 – Fast Infoset, 2005.

[24] Deutsch, P., GZIP file format specification version 4.3, RFC1952, http://www.ietf.org/rfc/rfc1952.txt, 1996.

[25] Burrows, Michael; Wheeler, David J., "A Block Sorting Data Compression, Algorithm," Technical Report SRC 124, Digital Equipment Corporation, Palo Alto, Californien, USA, 1994.

[26] H. Liefke and D. Suciu, "Xmill: an efficient compressor for xml data," SIGMOD Rec., vol. 29, no. 2, pp. 153-164, 2000.

[27] World Wide Web Consurtium (W3C), Efficient XML Interchange Working Group, Online, http://www.w3.org/XML/EXI/, 2009.

[28] World Wide Web Consurtium (W3C) Working Draft, Efficient XML Interchange (EXI) Primer, Online, http://www.w3.org/TR/2007/WD-exi-primer-20071219/, 2007.

[29] WS4D: Web Services for Devices, http://www.ws4d.org, 2009.

[30] ETF Network Working Group, DEFLATE Compressed Data Format Specification version 1.3, RFC 1951, http://tools.ietf.org/html/rfc1951, 1996.