A Wireless Sensor Network Protocol for the OMG Data Distribution Service

Kai Beckmann and Marcus Thoss
Distributed Systems Lab, RheinMain University of Applied Sciences
Unter den Eichen 5, D-65195 Wiesbaden, Germany
Email: {kai.beckmann|marcus.thoss}@hs-rm.de

***Abstract*—Wireless Sensor Network technologies are maturing into industrial applicability, but the availability of broadly supported architectures and integration of existing standards is still lagging behind. As a proposal to ameliorate this situation, we consider the OMG Data Distribution Service (DDS), a middleware standard for interoperable data-centric publish/subscribe architectures with real-time capabilities, a candidate for standards-based realisations of equally data-centric WSN scenarios.**

DDS is rooted in larger-scale architectures. Consequently, the RTPS network protocol defined as a sibling standard to DDS assumes Ethernet-sized network frames and substantial communication resources, which are potential obstacles to applying DDS to WSNs. We therefore propose SNPS as an alternative transport protocol for DDS communication. SNPS has been designed as part of sDDS, a model-driven DDS realization architecture supporting highly resource-constrained embedded sensor node platforms. Still, SNPS is generally independent of sDDS; it was demonstrated to be usable even with minimalist solutions lacking a proper DDS run-time system.

For SNPS, a modular layered architecture focussing on minimum footprint and scalability was defined. SNPS packets are assembled from an extensible set of submessages that are chosen and filled according to a well-defined, unambiguous protocol state engine. Keeping a stack of context information for a stream of SNPS submessages minimizes explicit state representation in the resulting packet structure. Other aspects of the protocol design are the support for bundling of data to minimize the total number of link layer frames exchanged and the leveraging of multi- and broadcast properties of wireless sensor networks. Most of the features are beneficial for wired sensor networks as well.

SNPS has been implemented for several wireless and wired network protocols (ZigBee, 6LoWPAN, and Ethernet/UDP/IP) on diverse embedded sensor node and PC platforms. It has been embedded in the sDDS architecture and integrated into minimalist standalone implementations. It will be the main DDS transport protocol used at sensor integration levels of current projects of the authors' laboratory research group targeting home and industrial automation scenarios.

I. Introduction

Wireless Sensor Network technologies are maturing into industrial applicability. Most innovations, however, still focus on research questions and on finding innovative, but sometimes isolated solutions to them. Despite some prominent exceptions like the widespread support of the network standard IEEE 802.15.4, the use of well-established standards for WSN technology is still lagging behind. This paper describes an effort that aims at closing part of this gap.

Data-centric modeling techniques are a good match for wireless sensor networks. With the Data Distribution Service (DDS) [1], the OMG has created a middleware standard for interoperable, platform- and vendor-neutral data-centric publish/subscribe architectures with real-time capabilities. A data model is defined in terms of OMG IDL and related to *topics*, which can be subscribed to or published by applications. Additionally, the API of DDS provides of a rich set of data-centric QoS policies.

DDS is not intended for WSNs, but the functionality matches many requirements proposed for WSN middleware. The recommended wire protocol is RTPS [2]. Being rooted in larger-scale architectures, it assumes Ethernet-sized network frames and substantial communication resources, and thus its protocol overhead seldom fits WSN requirements. Therefore, one of the challenges to overcome for leveraging OMG DDS technology for WSNs is the replacement of RTPS, which is an optional sibling part of the standard. Instead, an alternative protocol is presented that is suitable for extremely resource-critical WSN networking environments.

Here, the Sensor-Network Publish-Subscribe protocol (SNPS) is proposed for making DDS concepts better available to the WSN community. Based on the modular architecture of DDS, SNPS has been devised as an alternative wire protocol for DDS nodes. Implementations of it have been created in the authors' lab within, but not dependent on, a prototypical DDS development and implementation framework (sDDS) targeting both small embedded and PC-level node hardware [3]. SNPS aims at supporting a wide set of underlying communication layers to integrate heterogenous architectures and to provide a scalable realisation of DDS. So far, ZigBee and 6LoWPAN are supported with regard to typical small-scale wireless architectures as well as wired Ethernet/IP/UDP and CAN.

II. Background

A. OMG's Data Distribution Service

With the Data Distribution Service (DDS), the OMG has published an open standard for a data-centric publish-subscribe middleware platform with a rich set of real-time capabilities [1]. The DDS API and the fundamental system entities are expressed in a platform independent and object-oriented manner in OMG IDL.

Compared to other OMG standards like CORBA, UML and MDA, DDS is not very popular yet. Still, with the OMG's focus on portable, vendor-neutral standards, and their successful record of standardizing industry-oriented distributed systems architectures, the DDS standard is likely to assume a relevant position in the data-centric middleware landscape.

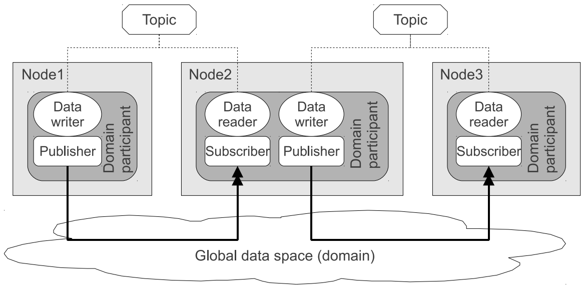

Fig. 1. Essential DDS architecture

Since DDS is data-centric, the modelling of the data proper is of primary importance for the definition of a DDS system. DDS is based on the idea of a global data-space, where typed data can be published and subscribed by participating entities. The basic structure is displayed in fig. 1. Each distinguished kind of data is addressed by a *topic*, which adds a name and an optional set of mandatory QoS policies to its data type. The names of topics must be unique within a *domain*, which provides application-level grouping of data types and communication realms. The data types can be defined with a subset of the OMG IDL syntax, offering complex types like structures and arrays as well as simple atoms like integers. Topic types are application-specific since DDS does not specify a distinct run-time typing authority. Therefore, matching application interfaces must be generated from the IDL definition to support type-safe handling. Those interfaces are derived from the *data-writer* and *data-reader* classes, which are the point of access for interchanging data samples with the middleware.

A subscription is established if at least one publisher (data writer) and at least one subscriber (data reader) for the same topic exists. QoS policies can be assigned to DDS objects to constrain the subscription relation with regard to reliability of communication, bandwidth consumption, and latencies.

The DDS standard only specifies a middleware API. To provide compatibility of DDS implementations from different vendors, a matching wire protocol is needed. The "Real-Time Publish-Subscribe" (RTPS) [2] protocol standard was created to be used within DDS, and it is the officially recommended protocol to provide DDS interoperability. RTPS requires a datagram service as a subordinate layer and defines a mapping for UDP/IP. To that effect, RTPS implies a minimum frame size and routing behaviour. Furthermore, RTPS maps the loosely coupled data-centric topic model to a set of relations among concrete nodes providing and requesting resources. QoS and real-time aspects, as formalized by DDS, are realised through the functionality of the protocol.

B. The sensornet Data Distribution Service

The data centric publish-subscribe approach and the rich set of accompanying QoS policies of the DDS standard match many special requirements of applications for WSNs, but DDS was not initially designed for this purpose. The limited and often heterogeneously distributed availability of resources with regard to computing and storage capacity or energy within a WSN mandates a downscaling of middleware features to match the applications and target systems at hand.

With the "sensornet Data Distribution Service" (sDDS), the authors have designed and realised an approach to generate individual DDS implementations at node level, by applying a model driven software development (MDSD) process [4]. In this process, system, application and target requirements, capabilities and the desired feature sets are defined separately with domain-specific languages (DSLs), and joined based on a meta-model spanning all aspects. Utilising that information, middleware functionality is selected within the MDSD process, mapped to the target environment, and the middleware implementations are generated for the individual nodes of the system. The application itself is implemented using the subset of the DDS-API and functionality selected. Tailoring the DDS type space and subsetting the API are considered reasonable, and often necessary, measures for achieving application footprints that match the typical, limited WSN node resource offers. By virtue of the MDSD process, changes affecting the target hard- or software or the functionality needed merely require the adaption of the model and a re-generation run.

To further reduce the memory footprint and optimise the processing of data, the MDSD process and code generation for sDDS produce a deliberately compact middleware implementation with less abstraction layers within the code base than those found within in the model spaces. The fine-granular object-oriented modularisation and abstraction layers are thus kept on the meta- and specific model level. To facilitate broad platform support and ease the adaption to new systems, most of the middleware code is plain ANSI-C. Platform-specific elements that are encapsulated in corresponding code templates are abstracted through sDDS-internal interfaces.

The custom selection of DDS functionality and the heterogeneity of implementations for the nodes of a system, also with regard to functionality, entails the need for a communication protocol that can cope with these aspects in addition to basic WSN communication tasks. SNPS is our proposal to meet those requirements.

This paper focuses on the syntax and semantics of SNPS, which we consider a major contribution to the applicability of sDDS for DDS-based WSN scenarios. Beyond that, we will not delve into the details of sDDS and the corresponding MDSD process, but in [3], we were able to show that, on a technical level, DDS can be used in WSN scenarios, and in [4], the MDSD process is presented thoroughly.

III. Related Work

Being one of the pioneers for many developments in the WSN community, the TinyOS project [5] was also used in an effort to integrate DDS technology with the embedded sensor network landscape, coined TinyDDS [6]. As TinyOS development and APIs are based on a proprietary programming language called nesC, DDS APIs also had to be adapted to that environment for TinyDDS. Thus, portability on the DDS standard level, which lacks official language bindings for nesC, is lost. Regarding the protocol realisation, TidyDDS uses an approach following the OSI network layer model with replaceable layer implementations. TinyGIOP, forming a subset of the OMG General Inter-ORB Protocol (GIOP), is used at the session layer for the exchange of data between DDS objects. The data itself are encoded in TinyCDR, a subset of CDR, adapted to the requirements of WSNs. The routing functionality is encapsulated in the "Overlay Event Routing Protocol" layer (OERP) and can be replaced with several appropriate implementations. The "TinyDDS L4 Adaption Layer" (L4AL) connects the OERP with network and data link layer protocols like AODV and ZigBee [7]. TinyDDS must be considered to aim at different goals as sDDS/SNPS, sacrificing portability and a standardised language binding for a rather complete, easy-to-use solution that is well integrated with TinyOS and its protocol stack. Its strictly layered protocol stack provides abstractions and flexiblity at the expense of code size and protocol overhead (cf. [8]). It should be noted, though, that TinyDDS can also be used with a Java binding, which provides compatibility with the DDS standard in that language context.

Another approach of applying the DDS standard for embedded systems, with focus on real-time aspects is the "micro Data Distribution Service" (μDDS) [9]. μDDS implements a subset of the DDS standard as well, but it is implemented in the C programming language and uses the POSIX 5.1 interface of the PaRTiKle real time OS. Additionally, Java with jRate is considered for a real time Java-based solution. IEEE 802.15.4 and ZigBee are employed as communication technologies, and the routing functionality is described to be replaceable. Code generation is only used for the type-dependant interfaces of DDS. μDDS is intended for larger embedded systems, focusing on real time rather then minimal resource consumption in WSNs.

A good example for a platform supporting both the standard RTPS protocol and efficient proprietary solutions for DDS communication is OpenSplice DDS [10]. Its modular design and its availability as an open-source version make it an ideal candidate for DDS deployments and adaptations over large scales of system architectures. The extremely limited resources of an embedded node, a few hundred KBytes of code at most, that are considered in the work presented here cannot be covered by downscaling OpenSplice DDS, though.

A comparison with the many non-DDS-related WSN protocols that have been proposed and implemented is well beyond the scope of this presentation, which focuses on providing a platform- and implementation-neutral protocol for DDS implementations on WSNs. As to this specific requirement, this is considered a novelty, with the exception of TinyDDS, as mentioned. An example for a more general overview of the goals of WSN protocols and the middleware concepts thus supported is [11].

IV. SNPS

As a protocol for a DDS implementation for WSNs, SNPS must meet several specific requirements. To conserve energy, radio networks for WSNs support only small frame sizes and provide limited bandwidth. Besides the action of sending and receiving the data proper, significant amounts of energy are needed to initiate transmissions at all. Therefore, reasonable aggregation of data and the low overhead for multi- or broadcasting schemes in the case of radio networks should be utilised by WSN-optimized protocols. As a primary requirement, relevant parts of DDS functionality must drive the design of the protocol, and, following the strengths of sDDS, nodes of a system with heterogeneous implementations should still be able to collaborate.

A. Fundamental SNPS Properties

To meet these requirements, we present the Sensor-Network Publish-Subscribe (SNPS) protocol, whose design focuses minimum protocol overhead, fine-granular modularity and the usability within and without sDDS middleware. SNPS is meant to carry data, and control and management information within a data centric publish-subscribe network. To facilitate efficient message buffer handling and processing, the protocol implementation shall be tightly coupled to the middleware or application using it.

Different from RTPS, SNPS supports very small frame sizes and utilises the natural ability of radio networks for low-overhead multi- or broadcasting to support the aggregation of multiple data samples for different receivers within one packet. To achieve this, the addressing scheme is data- or content-specific, which is another difference to RTPS. In terms of energy consumption, sending a larger packet to several receivers can be favourable to the transmission of two unicast packets. The reception of possibly unneeded information in the packet can be used for establishing redundancy and for future data routing or aggregation mechanisms.

SNPS assumes datagram service functionality in a subordinated protocol layer. Currently, SNPS realisations cover WSN-oriented networks like ZigBee and 6LoWPAN as well as generic Ethernet/UDP/IP-based stacks. While the current implementation of SNPS needs supporting routing functionality, the protocol design anticipates the option of intra-SNPS routing. A data centric middleware being aware of and controlling routing decisions has the potential for more effective distribution of the data and a better usage of system resources. Especially the exploitation of multi- or broadcast properties is a promising aspect [12].

SNPS has been designed to be used within sDDS. As described earlier, this DDS implementation targets WSNs, and it can be individually generated for each node using an MDSD process [4]. Hence, the middleware functionality supported may differ on the nodes within a network, but a peer-to-peer-semantic shall still be possible. To cope with this fact, and to reduce the protocol overhead, SNPS uses a dual approach of implied structure and self-description within the protocol. When only little common knowledge about the size of protocol elements is required, a middleware implementation can skip unknown parts of a message that state their own length, which corresponds to ignoring locally unsupported functionality. Also, SNPS does not depend on sDDS: Even stand-alone protocol implementations lacking a DDS run-time system can be used to have very small targets interact with sDDS nodes, which was demonstrated in a prototypical implementation.

The structure of SNPS is based on the idea of small atomic information units, called "Submessages", which can be used to encode information proper, or to convey control and management functionality. Sub-structuring of messages is part of the RTPS design as well. In contrast to RTPS though, the sequential order of Submessages establishes contexts for the interpretation of the encoded information. Thus, protocol overhead and the encoded state information are reduced during message processing. The grouping of Submessages can be related to specific functionality of the DDS standard.

The encoding of the payload is the responsibility of the middleware implementation using SNPS. For sDDS, for instance, the set of data types supported was selected with respect to fixed type lengths and reasonable processing effort. The encoding itself is based on CDR except for the alignment on byte boundaries, which helps to reduce the payload size. The endianness is determined during the code generation for the particular topic type according to the target system. When necessary, SNPS uses the same encoding for primitive data types of the protocol messages.

B. SNPS Submessage types

There are three sets of Submessages defined for SNPS with increasing ranges of size: Basic-Submessages, Extended- and Supplements-Submessages. For each Submessage, the size is known or given as part of the header.

Although every Submessage type is distinct, Submessages with the same functionality are sometimes made available in more than one set, having different header sizes and information capacity, to allow for greater flexibility in the usage of the most common message elements and thus reduce overhead.

For the Basic-Submessages, header type and at least part of the payload are packed in a single byte. They are used for the most common tasks when a payload is needed. From the sixteen possible types, the ten types specified so far are displayed in Fig. 2, leaving room for future extensions. Data samples are addressed with a domain id and topic id of 4 bits each. The most frequently used data types shall have the lowest id, while less frequently used types will use an Extended-Submessage. Likewise, for the data proper, payloads of up to 15 bytes fit in a Basic-Submessage, otherwise, an Extended-Submessage must be used.

To support a reliable transmission of recurring data samples, sequence numbers with 4 bits are assumed to be sufficient, helping to further reduce the overhead. Depending on the use case, a handshaking (ACK/NACK) scheme can be employed, if supported by the operating middleware.

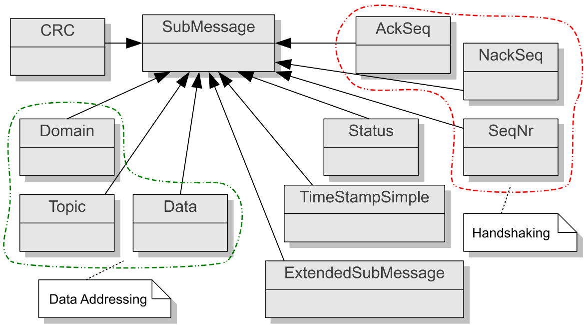

Fig. 2. Basic-Submessages

For the basic timestamp, a maximum payload of 20 bits is allowed, to be filled with an appropriate, system-specific representation. CRC and status types are allotted for the future use of SNPS on minimalist run-time implementations, which would need to integrate data link layer functionality in SNPS.

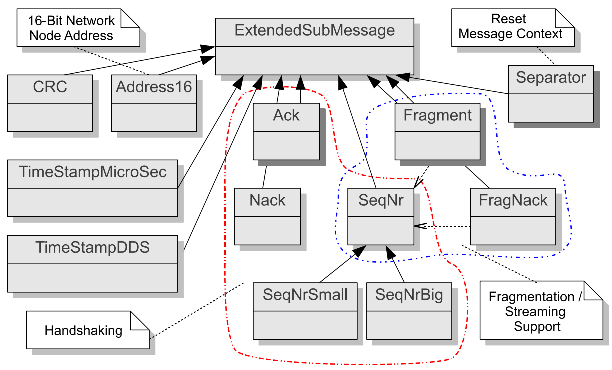

Fig. 3. Extended-Submessages

Extended-Submessages treat the 4-bit field in the first byte not as payload but as a type discriminator. They are used for parameterless indications, like an ACK, or for common tasks whose payload exceeds 4 bits. The most relevant Extended-Submessages are shown in Fig. 3. They also adress the need for higher-resolution timestamps and for longer sequence numbers. Since the focus is on nodes processing small-sized data, support for fragmentation is only found in this slightly bigger set of Submessages. For future use, support of routing and data link functionality and a more accurate CRC type are included. The Separator Submessage is needed to reset a message context, which will be explained in the next section. All Basic- or Extended-Submessages must be well-defined because their size is implicit, and must be recognized, even if the information is not processed by a specific implementation.

The header size of Supplement-Submessages is at least two bytes; it carries explicit size information. Supplement-Submessages are used for rare use cases and allow a more flexible extension of the protocol. The explicit size information ensures compatibility with nodes lacking support for some of the protocol elements.

C. SNPS message context

As stated in section IV-A, the aggregation of multiple data samples within one package is an important feature of SNPS. The sequential order of Submessages establishes contexts for the interpretation of the encoded information. Therefore, protocol state control can often be implied, reducing the overhead incurred in Submessage headers. Switching protocol states in this fashion also simplifies the parsing and generation of SNPS messages. To demonstrate the context scheme, Fig. 4 displays the encoding of three data samples.

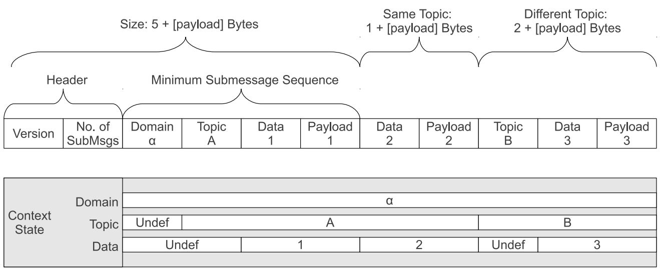

Fig. 4. Anatomy of SNPS Basic Messages

Every SNPS message is initialised with the version of the protocol and the number of Submessages following, giving the protocol implementation a hint for optimising the processing of the message.

The occurrence of each control Submessage, which can be specifying a domain, a topic or a data sample index (shown as “Data Addressing” Basic-Submessages in Fig. 2), spans a context that remains valid until another Submessage of the same type changes the context, or until it is reset completely by a *Separator* Extended-Submessage.

Data samples are related to DDS topics, and these are grouped within DDS domains. In the example of Fig. 4, all data samples are assigned to topics of the Domain *a*, set up as a context by the first Domain Submessage. Similarly, the following conetext of Topic *A* spans two consecutive Data Submessages *1* and *2*. These in turn provide a conext for the data samples proper and optional meta data attached to them, like timestamps (not shown here).

The minimum sequence of Submessages needed for a useful data transmission is displayed in Fig. 4 as well. Besides the header of two bytes, domain, topic and data Submessages, and the data sample payload itself are needed, giving a minimum header size of five bytes.

D. Proof of Concept Realisations

Because of the close relationship of the SNPS protocol and sDDS, the implementation of both is currently integrated. Yet, SNPS implementations have also been split off and used standalone.

The current code base is largely platform independent, and it uses abstract interfaces to embed platform-specific modules to facilitate porting to different target systems at this stage of the development. The SNPS implementation is realised in small platform independent code modules which can be composed during the code generation, considering the functionality needed. So far, although, the SNPS implementation does not take the underlying network layers into account. Currently, SNPS messages are assembled in the platform independent part of sDDS, and then forwarded to a platform-specific network module. This module must provide a generic datagram service functionality, and it encapsulates platform-specific parts, like addresses. This approach has proven beneficial, since we were able to port sDDS and thus SNPS to several different platforms and network architectures.

The need of a generic datagram service can be satisfied with UDP/IP, and the largely generic code base of sDDS permits the usage of popular POSIX PC platforms, avoiding the permanent need for WSN hardware access and providing a versatile primary development environment. Furthermore, the integration of a WSN system into a PC-level network with continuous deployment of sDDS becomes feasible.

Based on the IEEE 802.15.4 protocol, the ZigBee standard is promoted for WSN applications in home and industry environments by different hardware vendors. Attractive radio and microcontroller solutions are thus available, and ZigBee was initially chosen for sDDS and SNPS. A first approach had been to exploit the ZigBee profile functionality and wrap DDS topics in profiles, mapping the topic data structure to profile attributes. The data discovery, distribution and the subscription management could then have been delegated to a ZigBee stack with sDDS as thin layer only, mapping the DDS API to ZigBee. Further analysis showed that the ZigBee profile definition is not flexible enough for this purpose: Profiles must be assigned by the ZigBee Alliance, and not even a sandbox area is provided, whereas the flexible and application-specific topic and data definition is one of the major advantages of DDS. Therefore, only the transport layer of the ZigBee stack is used and more functionality remains with sDDS and SNPS.

So far, SNPS has been implemented on two ZigBee stacks from different hardware vendors: Texas Instruments’ zStack [13] on the System-on-Chip platform CC2430 and the BitCloud stack developed by Atmel [14]. Since the broad toolchain and compiler support for Atmel microcontrollers for Linux aids the integration in the sDDS development environment, the ATmega1284 and ATmega128RFA1 platforms are now the primary target platform for sDDS and SNPS.

SNPS is also supported on 6LoWPAN, which offers the IPv6 protocol for WSN environments [15]. With 6LoWPAN, sDDS can be used in mixed WSN and traditional Ethernet-based networks without the need for a complex protocol mapping on a gateway. One comprehensive 6LoWPAN stack is part of the Contiki operating system for very small embedded devices, which is widely used in academic and commercial applications [16]. The current sDDS implementations were ported to Contiki and 6LoWPAN within a student project, taking about three weeks. Large parts and the logic of the UDP/IP realisation could be reused, although adoptions for supporting IPv6 under Linux were necessary as well. A transparent gateway between 6LoWPAN and IPv6 networks is provided by the Contiki project.

As part of another student project, SNPS support for the Controller Area Network (CAN) bus was launched. Other works [17] support our assumption that the utilisation of CAN for DDS, exploiting CAN's message-based addressing scheme for DDS topics, is quite promising, besides its real-time aspects. Admittedly the maximum payload of 8 bytes of a CAN frame is challenging even for SNPS, which is already optimised for small frame sizes. Our project results suggest that the function set of SNPS must be reduced drastically, and the general Submessage concept would need to be abandoned. A detailed discussion cannot fit in this publication, though.

In comparison to RTPS, SNPS gains its advantages by the smaller header. Additionally to the underlying protocol headers, each RTPS message is started with a 20 byte header, and each data segment as well, whereas the payload has an own 8 byte header and has to be aligned. In contrast to SNPS the basic message shown in Fig. 4, would need 48 bytes. If several data samples (eg. n) for one topic are put in one message, the overhead would be $20+n*24+n*payload$ bytes, whereas SNPS would only need $2+2+n*1+n*payload$ bytes, if the payload has less than 16 bytes. Admittedly it have to be noted, that RTPS includes a mandatory sequence number of 8 byte in each data segment, while in SNPS this is optionally and different sizes are selectable, according the individual use case of the application.

V. Summary, Conclusion

The WSN community could benefit from more standards-based approaches, and the OMG DDS Standard provides features that match the requirements of WSN architectures almost naturally. Unfortunately, although DDS is a promising candidate for WSN scenarios, its roots, targeting more powerful hardware, entailed features contradicting WSN applications, like the resource-hungry RTPS communication protocol.

We could show that an alternate protocol approach can be defined that focuses deliberately on low-powered WSN node hardware and sensor node networks with its typical, small frame sizes and transmission schemes. The conception of the SNPS protocol as a replacement for RTPS in such scenarios captures those aspects in its basic packet structures but also in the protocol engine. SNPS supports a reasonable feature set of the DDS standard, and it has been designed expandable to provide additional features with new submessage types. Yet, interoperability with nodes recognizing only a limited set of message types is ensured.

Realisations of SNPS have been created as part of the MDSD-based sDDS architecture. It has been implemented for recent embedded wireless node targets using ZigBee and 6LoWPAN networking. For management and control functionality, and for debugging and measurement applications, SNPS is also available for PC-level platforms using Ethernet/IP/UDP networking.

As SNPS (and sDDS) are still evolving, many concepts and protocol elements will be revisited. There are also many desirable features that have yet to be investigated and realised. The support of the QoS policies of the DDS standard through SNPS beyond this presentation has not been finalised yet. If properties of underlying transport layers cannot be easily integrated, it might be necessary to extend the protocol structure.

We also expect the SNPS implementations to benefit from the interweaving of code snippets realising protocol handling with those implementing sDDS middleware and application layers. So far, the clean separation of the MDSD model layers has been sustained throughout code generation, but especially protocol buffer access and method calls will be streamlined to reduce the size of both code and data buffers.

Besides continuing the development of SNPS and its implementations, one major step currently in preparation will be the deployment of sDDS and SNPS in larger, heterogeneous WSN applications within the authors' research projects.

References

[1] OMG, "Data Distribution Service for Real-time Systems, Version 1.2," Tech. Rep. formal/07-01-01, Jan. 2007.

[2] ——, "The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification (DDS-RTPS)," Tech. Rep. formal/2010-11-01, Nov. 2010.

[3] K. Beckmann, "Konzeption einer leichtgewichtigen, datenzentrierten Middleware für Sensornetze und eine prototypische Realisierung für ZigBee," Master's thesis, RheinMain University of Applied Sciences, Wiesbaden, Germany, Apr. 2010. [Online]. Available: http://wwwvs.cs.hs-rm.de/downloads/extern/pubs/thesis/beckmann10.pdf

[4] K. Beckmann and M. Thoss, "A Model-Driven Software Development Approach Using OMG DDS for Wireless Sensor Networks Software Technologies for Embedded and Ubiquitous Systems," in *Software Technologies for Embedded and Ubiquitous Systems*, ser. Lecture Notes in Computer Science, S. L. Min, R. Pettit, P. Puschner, and T. Ungerer, Eds. Springer Berlin / Heidelberg, 2011, vol. 6399, ch. 11, pp. 95–106.

[5] "TinyOS," Website. [Online]. Available: http://tinyos.net/

[6] P. Boonma and J. Suzuki, "Middleware Support for Pluggable Non-Functional Properties in Wireless Sensor Networks," in *Proc. IEEE Congress on Services - Part I*. IEEE, Jul. 2008, pp. 360–367.

[7] ——, "Toward Interoperable Publish/Subscribe Communication between Wireless Sensor Networks and Access Networks," in *Proc of the 6th IEEE Consumer Communications and Networking Conference (CCNC'07)*. IEEE, Jan. 2009, pp. 1–6.

[8] ——, *TinyDDS: An Interoperable and Configurable Publish / Subscribe Middleware for Wireless Sensor Networks*. IGI Global, Jun. 2010, ch. 9, pp. 206–231.

[9] A. González, W. Mata, L. Villaseñor, R. Aquino, J. Simo, M. Chávez, and A. Crespo, "DDS: A Middleware for Real-time Wireless Embedded Systems," *Journal of Intelligent & Robotic Systems*, vol. 64, no. 3, pp. 489–503, Dec. 2011.

[10] PrismTech, "OpenSplice DDS." [Online]. Available: http://www.opensplice.com/

[11] W. Masri and Z. Mammeri, "Middleware for Wireless Sensor Networks: A Comparative Analysis," in *Proc. of the International Conference on Network and Parallel Computing Workshops. NPC Workshops. (IFIP'07)*. Los Alamitos, CA, USA: IEEE, 2007, pp. 349–356.

[12] H. Karl and A. Willig, *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, Oct. 2007.

[13] TI. zStack. [Online]. Available: http://www.ti.com/tool/z-stack

[14] Atmel. BitCloud. [Online]. Available: http://www.atmel.com/tools/BITCLOUD-ZIGBEEPRO.aspx

[15] N. Kushalnagar, G. Montenegro, D. E. Culler, and J. W. Hui, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," Internet Requests for Comment, RFC Editor, Fremont, CA, USA, Tech. Rep. 4944, Sep. 2007.

[16] The Contiki OS. [Online]. Available: http://www.contiki-os.org/

[17] S. Hasnaoui and R. Rekik, "Application of a CAN BUS transport for DDS middleware," in *Proc. of the Second International Conference on the Applications of Digital Information and Web Technologies (ICADIWT '09)*. IEEE, Aug. 2009, pp. 766–771.