Computer Communications xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems ☆

Corinna Schmitt [a,*], Thomas Kothmayr [b], Benjamin Ertl [c], Wen Hu [d], Lothar Braun [b], Georg Carle [b]

[a] Department for Informatics, Communication Systems Group, University of Zurich, Switzerland
[b] Department of Computer Science, Technische Universität München, Germany
[c] Institute for Telecommunication Systems, Chair for Next Generation Networks, Technische Universität Berlin, Germany
[d] CSIRO, Brisbane, Australia

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Wireless sensor networks
TinyIPFIX
Aggregation
Efficient data format
Cyber-physical system

ABSTRACT

Wireless sensor networks (WSNs) as a central part of cyber-physical systems are gaining commercial momentum in many areas, including building monitoring and intelligent home automation. Users wish to successively deploy hardware from different vendors. Interoperability is taken for granted by the customers who want to avoid the need for exhaustive configuration and set-up. Therefore, the need for an interoperable and efficient application layer protocol for machine-to-machine communication in and across the boundaries of WSNs arises. We address these issues with our implementation of TinyIPFIX, an adaption of the IP Flow Information Export (IPFIX) protocol. Throughout the paper we show how to leverage TinyIPFIX in the context of an office scenario and we discuss how the protocol may be applied to other significant WSN deployments presented in literature over the past few years. This article additionally shows how to improve the functionality of TinyIPFIX by adding both syntactic and semantic aggregation functionality to the established system. Finally, we evaluate the performance of TinyIPFIX in a large test bed with over 40 motes running TinyOS and analyze TinyIPFIX's system performance in comparison with previous approaches.

1. Introduction

The Internet of Things (IoT) is seeing rapid adaption across many industries. For example, Cisco Systems is predicting a growth from 369 million machine-to-machine (M2M) modules in 2012 to 1.7 billion M2M modules globally in 2017 [3] – and these figures are only for mobile M2M devices that connect via cellular networks. Arguably, the number of devices connecting via local wireless networks is even higher. Because the IoT has a plethora of different usage scenarios [4], it also covers a wide range of device classes from powerful smartphones on the high end to devices that are highly constrained in memory, energy supply and computing capacity.

The focus of this paper lies in delivering an efficient application protocol for machine-to-machine communication in a cyber-physical system. Our target device classes are the constrained devices (motes) found in wireless sensor networks (WSNs), which often form a key component of a CPS. One common application area can be found in the field of building automation, meaning the autonomic monitoring and control of environmental conditions in residential and commercial buildings for improved comfort, as well as a reduced energy usage and carbon footprint. Wireless sensors are deployed to monitor key values, such as room temperature or brightness, in different locations. They transmit the data to a control and management system, which analyses the measurements and reacts on the results, e.g. turning on/off heating or lights. Not only the devices themselves are constrained in this scenario. The low power wireless network over which they communicate also imposes severe limits on throughput and message size. Any application protocol used in this scenario must be efficient in its use of

☆ Part of this work was published in Proceedings of the 7th European Conference on Wireless Sensor Networks (EWSN) 1 and was mostly done when Corinna Schmitt and Benjamin Ertl were with Technische Universität München 2. The extensions to the EWSN article include: First, an in-depth analysis on how to apply TinyIPFIX for a wide range of sensor network deployments. Secondly we demonstrate in-network aggregation support under TinyIPFIX, which includes data and message aggregation, as well as individual and direct configuration of the aggregation functionality on aggregator nodes. Third, an extensive analysis and system level evaluation of TinyIPFIX's transmission efficiency and resource consumption is presented along with a comprehensive comparison with other approaches. Compared to the original paper, there are significant modifications in Sections 3.4, 4 and 5.

* Corresponding author. Tel.: +41 44 635 7585.
E-mail addresses: schmitt@ifi.uzh.ch (C. Schmitt), kothmayr@in.tum.de (T. Kothmayr), benjaminertl@tu-berlin.de (B. Ertl), wen.hu@csiro.au (W. Hu), braun@net.in.tum.de (L. Braun), carle@net.in.tum.de (G. Carle).

http://dx.doi.org/10.1016/j.comcom.2014.05.012
0140-3664/

network, computational and energy resources. However, it should still be comfortable to use for the system developer as well as complete and generic enough to allow easy deployment by the end user.

Web services are a well-known approach to M2M communication, but directly adapting the techniques from traditional computing are not feasible in constrained networks because they rely on verbose XML formats to exchange messages. For example, it takes up to 442 bytes to get a temperature value encapsulated with SOAP 1.2 [5]. This is often addressed through compression of the XML data [6] or by directly encoding the XML message in a binary format [7]. HTTP itself is also considered too resource intensive for constrained networks and alternatives like the Constrained Application Protocol (CoAP) [8] have been developed. The related work is discussed in Section 2 in more detail.

While suited to the domain of constrained networks, CoAP also introduces additional implementation complexity that might not be needed in all usage scenarios of WSNs. For example, a wireless thermometer that periodically reports to an automation server with a direct user interface is duplicating functionality that is available by accessing the data on the data sink it is reporting to. Our approach focuses on the core functionality of stationary Wireless Sensor Networks: Periodic reporting of sensor data to a data sink with low network, memory and computational overhead while still enabling easy integration of diverse sensor hardware on motes from different vendors with minimal configuration and maintenance overhead. The approach is centered around TinyIPFIX, a lightweight adaption of the IP Flow Information Export (IPFIX) protocol [9] for WSNs. Section 3 presents a brief characterization of the TinyIPFIX protocol and discusses its characteristics focusing on the constrains of wireless sensor nodes. In general, the design space for an application protocol to achieve tight integration of a WSN into a CPS consists of four areas:

1.1. Metrology

Sensor devices measure data, which has a specific format and must be represented accordingly. This representation should be general and universal, meaning that a protocol should be able to uniquely designate each measurement type across all WSN deployments. A measurement type is defined here as a reading from a specific model of a sensor, which carries information about the data type and its conversion to scientific units, rather than an abstract quantity such as "temperature". In the case of TinyIPFIX the sensor measurement data is identified by an individual Type ID and Enterprise Number (EID), which are registered with the Internet Assigned Number Authority (IANA).[1] This ensures adaptability to other platforms or new measurement types. Since an IPFIX Template only carries syntactical meta data for the measurements sent in an IPFIX Data packet the semantics for that data still need to be supplied. If the Enterprise and Type IDs have been allocated globally unique, a public repository for this semantic data, presented as XML markup, becomes feasible. We will give an example for such a markup in Section 5.4.

1.2. Resource efficiency

The resources of sensor nodes are limited in terms of power, memory space and computational capacities. We evaluate TinyIPFIX with regard to its memory requirements and energy consumption. Additionally, we implemented the TinyIPFIX-Aggregation protocol, which offers in-network aggregation mechanisms for data pre-processing. By leveraging in-network aggregation additional energy savings can be achieved through transmission reduction.

1.3. Syndication

The benefits of using IPv6 in sensor networks were detailed in previous work [10]. We choose to send TinyIPFIX packets via the BLIP [11] implementation of IPv6 and UDP, because it offers seamless integration into an existing IP-based network infrastructure.

1.4. Scalability

In Section 5.1 we discuss the flexibility of TinyIPFIX by showing how it could have been leveraged in other significant deployments presented at IPSN or SenSys over the past few years. We present the results of numerous real world test runs of TinyIPFIX assuming an office scenario (see Figs. 1 and 10) and in a large WSN deployment on the Harvard Sensor Network Testbed (Motelab) testbed [12].

Section 4 describes the integration of a TinyIPFIX based wireless sensor network into a cyber-physical system used for building automation. We evaluate the performance of the TinyIPFIX protocol concerning its hardware requirements and demonstrate the functionality of the whole system in Section 5 before concluding the paper in Section 6.

2. Related work

Widespread adaption of traditional web services in constrained networks is stymied by HTTP's verbosity. With the Constrained Application Protocol (CoAP) Shelby et al. introduced a lightweight, yet interoperable, alternative to HTTP that allows the adaption of the web service principle to constrained networks [8]. Compared to HTTP, CoAP's main benefits are a reduced header size and no requirement for reliable message transport (i.e., CoAP only requires UDP and not TCP). Motes that have data to expose implement a CoAP server and expose their data offerings to the data consumers via a discovery service. Similar to HTTP, CoAP does not specify the actual format in which data is transported but supports different content encodings. Our approach, which is centered on IPFIX, is more comparable to a content encoding format in the context of CoAP. CoAP and IPFIX for sensor networks therefore have different concerns: While CoAP's goal is to bring the full suite of features that is required for a web-like experience to constrained networks, we aim to offer a simple M2M application protocol with minimal implementation and network overhead that can be used where the full set of features offered by CoAP is not necessary or the implementation complexity cannot be afforded.

A more direct comparison can be drawn between IPFIX and different content encoding formats used with CoAP, HTTP or standalone: XML is arguably the most well known format for transferring structured data in a human readable way. However, the clear text format of XML results in very large message sizes and slow processing times – even in the field of traditional computing. JSON is a more compact format to transfer structured, human readable data but it still cannot achieve the same level of message compactness as binary formats. A large amount of effort has been undertaken to reduce the size of XML documents while simultaneously improving their processing speed. Two representative approaches are Fast Infoset [13] and Efficient XML Interchange (EXI) [7]. Compared to Fast Infoset, EXI achieves a higher rate of compression because it is able to take the structure information provided by an XML schema into account. However, both the encoding and decoding party needs to process the matching schema to leverage the increased rate of compression. A schema-less mode is available in EXI as well.

[1] http://www.iana.org.

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012

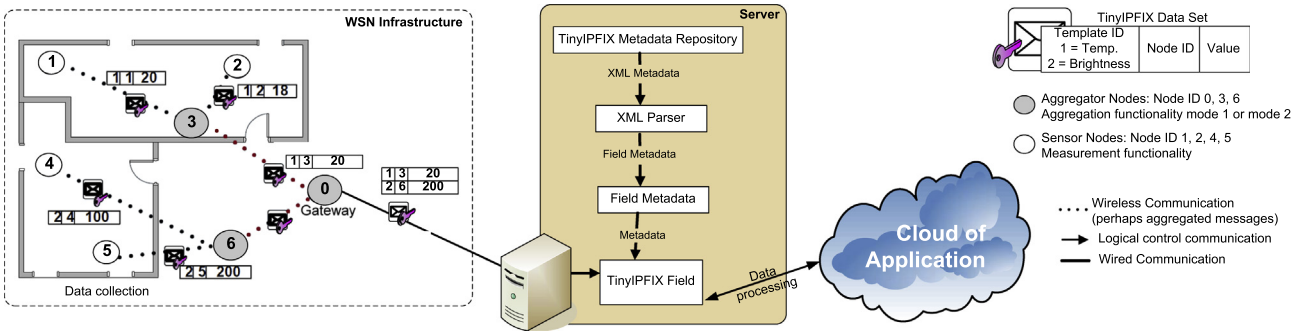

Fig. 1. Abstract system architecture.

Compared to our approach XML offers the capability to transfer structured (nested, repeating, value constrained, variable length, etc.) data whereas IPFIX as specified in RFC 5101 [14] and more recently in RFC 7011 [9] has no support for structured data types. Further extensions, especially RFC 6313 [15], can be applied to overcome this limitation. However, it is our observation that most structure data is used to describe the metadata surrounding a measurement value or to group multiple sensor measurements into one message. How IPFIX deals with this issue of metrology is discussed in Section 3.2. The separation of data into a Template message is comparable to a message containing an XML schema without which binary XML approaches struggle to reach IPFIX's level of compactness (c.f. Section 5.2.2). Overall, the implementation complexity of IPFIX is significantly lower than with any approach using XML or a binary XML format.

In the field of in-network aggregation two main approaches exist today: (1) message aggregation and (2) data aggregation, which is also referred to as pre-processing in literature. The first approach concatenates two or more messages into a newly generated aggregate message without data pre-processing. The second approach means to evaluate the data transmitted from one sensor node to another one and to compute aggregates on this data, based on requested types of values and aggregate functions. The naive approach consists of performing these aggregates on the data sink. A better strategy is to perform in-network aggregation on sensor nodes with more resources. In this paper we present an in-network aggregation technique called TinyIPFIX-Aggregation, which implements message and data aggregation features. The remainder of this section briefly compares our approach to TAG, AIDA, and SIA. Those three approaches are the most prominent examples in literature.

In 2002 Madden et al. presented a Tiny AGgregation (TAG) service for ad-hoc sensor networks [16]. This approach is based on in-network data pre-processing, because applications often depend more on data aggregations rather than raw sensor data. The underlying aggregation queries are formulated in a syntax that is similar to the well-known SQL.

Przydatek et al. presented a security framework called Secure Information Aggregation (SIA) for wireless sensor networks in 2003 [17]. In SIA, sensor nodes transmit raw sensor data unsecured and employ secured computation of aggregates. The computation of secure aggregates consists of three parts: (1) data collection and aggregation, (2) commitment of the collected data by the aggregator, and (3) establishing a communication protocol with the data sink. One example for the data collection and aggregation part is the implementation of a spanning tree in the network. Other algorithms can also be adapted.

One year later He et al. presented the AIDA implementation, which is an Adaptive Application-Independent Data Aggregation approach [18]. The AIDA component resides between the data link layer and network layer with no specific application dependent knowledge. The component includes an aggregation module, which combines messages into new frames of the AIDA protocol. Due to the independence from the application layer, the AIDA framework can provide aggregation mechanisms for a range of different applications. No data semantics are lost during the aggregation process. AIDA merely combines received messages in larger frames for a better utilization of the communication channel, and, therefore, does not reduce the amount of the transmitted bytes, but as a result of less packets the amount of control messages can be reduced when congestion in the network occurs. Another advantage is the dynamically controlled degree of aggregation (DoA) in accordance with changing traffic conditions.

3. TinyIPFIX design

We start this section by giving a short overview of the original IPFIX protocol [9] before showing how it can be applied to sensor networks.

3.1. IPFIX overview

IPFIX was developed by the Internet Engineering Task Force (IETF) for transmitting flow information between different instances in the network [9]. Communication takes place between an *Exporter* and a *Collector*. IPFIX is specified as a PUSH-protocol with an Exporter periodically transmitting data to one or more Collectors. This makes IPFIX an attractive choice for WSNs, because they often rely heavily on Collection traffic. This traffic pattern means that information flows from many source nodes to only a few information sinks such as the gateway nodes.

A template-based design is used to exchange measurement data while minimizing overhead. Measurement data is exchanged in *Records*. The protocol distinguishes, among others, between *Template Records* and *Data Records* as shown in Fig. 2. Together with the corresponding header those messages form a *Template Set* or *Data Set*. A detailed description of the data exchange and special requirements for sensor measurements can be found in reference [1].

Data Records contain the measurement data while Template Records contain the meta information of Data Records. The meta information covers the semantics, data type, and length of the measurement data. An Exporter sends a Template Record to its Collector to announce the structure of the upcoming Data Records. This Template record is only required once but may be repeated periodically. The Collector for decoding incoming Data Records stores the Template Record. A *Template ID*, which is unique for every Exporter and the templates it uses, is assigned to every Template Record sent to a Collector. Further Data Records will reference this ID.

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012

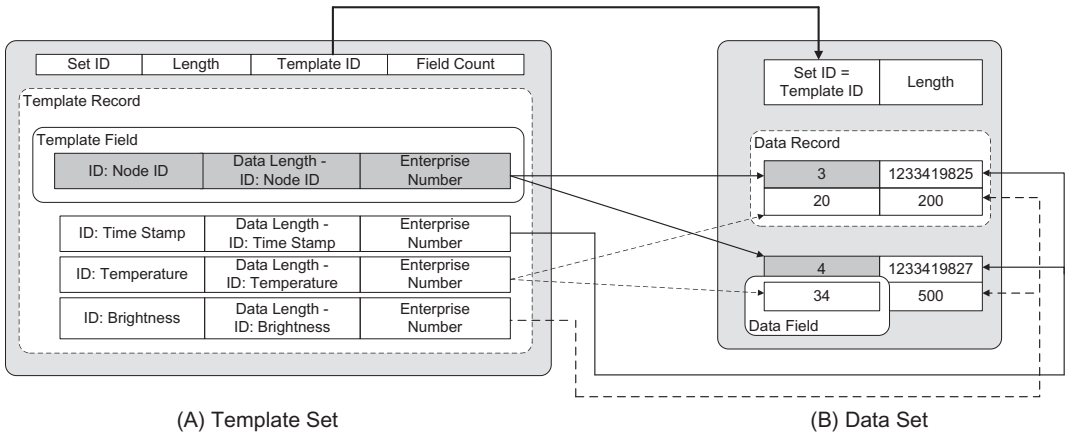

Fig. 2. Components of the IPFIX protocol showing decoding of the data.

As shown in Fig. 2, each Template Record describes the encoding of the transmitted sensor measurement values in a Data Record. A Template Record may contain several *Template Fields* where a field corresponds to a measurement type, e.g. brightness or humidity. Each field in the Template Record describes the type and length of the corresponding field in the Data Record (Fig. 2 shows four template fields). The type is uniquely described by a tuple consisting of a *Type ID* (2 bytes), a *length* statement (2 bytes), and an *Enterprise Number* (4 bytes) in a Template Field. The Type ID specifies the type of data while the Enterprise ID denotes the organization, which issued the Type ID.

Sensor nodes act as Exporters and transmit their measurement data using IPFIX. When a sensor node boots up, it has to announce its Template Record before sending its measurement data to the Collector. This has to be done only once since a Collector buffers the Template Record to decode incoming Data Records. They do not have to contain anything but the measurement data as all meta information has been already sent in the Template Records. A short header containing the number of transmitted values and the referenced Template ID only accompanies Data Records. Several Data Records can be put into a single message.

Section 4 characterizes the implementation of IPFIX for WSNs in detail. From this point on we mean TinyIPFIX if we use IPFIX in WSN context. The computation costs for creating a template are very cheap and the number and size of different templates and messages used in parallel bound the additional memory requirements of TinyIPFIX. Therefore, TinyIPFIX can be used on nodes with constrained resources when they limit themselves to a few templates of small size, which will be demonstrated by our implementation in Section 5.2.1. Multiple templates can be used if the nodes have more resources. They may also analyze the data directly instead of shifting this task to a server.

3.2. Metrology

The *Type ID* and the *Enterprise Number* (EID) identify Sensor measurement data. As described in the introduction, each measurement of a sensor should be assigned a globally unique combination of Enterprise and Type ID. For example, the Sensirion SHT11 temperature and humidity sensor would be assigned two different combinations, e.g. Enterprise ID 12345 and Type ID 32768 for its temperature channel and Enterprise ID 12345 and Type ID 32769 for its humidity channel. A public repository would allow an application that receives a template with an Enterprise and Type ID combination that it has not seen before to obtain the semantics of that measurement from the Internet. The semantic information includes what kind of data (temperature), the data type (16-bit integer), how to convert to a sensor independent format (formula to convert to °C), and any other required information. This would allow flexible deployments of motes with different sensors, for example, if an existing deployment using Sensirion SHT11 sensors was augmented with several new nodes that use a Sensirion SHT15 instead, there would be no need for extensive reconfiguration within the WSN or the converter application if both use IPFIX to send their measurement data. Nodes can simply transmit the raw measurements without having to convert them into other formats via potentially complex formulas. The receiving application on a PC can convert both values to scientific units and combine the measurements based on semantic information obtained from the repository. An example is shown in Section 5.4.

3.3. Adaptation to WSNs

Since IPFIX was designed for conventional networks for monitoring tasks [19], some extensions and changes have to be introduced to increase its efficiency in the context of WSNs. The implementation for WSNs is called TinyIPFIX. One of the problems when deploying IPFIX in sensor networks is the overhead introduced by the relatively large header, as shown in Fig. 3. In order to address this issue, a header compression scheme was developed, which is part of the TinyIPFIX protocol. Apart from header compression, TinyIPFIX is fully compliant with the IPFIX protocol. TinyIPFIX template and data set contents are identical to those of standard IPFIX.

Because the header size has large influence on the overall transmission efficiency of TinyIPFIX, we have chosen an aggressive approach to header compression [1]. It starts by limiting the capabilities of IPFIX to those required in WSNs. The IPFIX packet length is limited to 1024 bytes, which exceeds the maximum packet length defined by the IEEE 802.15.4 standard and, thus, requires packet fragmentation [20], which may be provided by the underlying network layer. For example, it is possible to send IPv6 messages of up to 1280 bytes length with BLIP [11]. With header compression, only one set of templates or data is transmitted. Fig. 4 shows the header of the aggressive approach where the *SetID* field is moved to the front of the header and shortened to four bits. It acts as a lookup field for the SetIDs and provides shortcuts to often used SetIDs [21]. The bits marked *E1* and *E2* control the presence of the extended SetID field and the length of the Sequence Number field respectively. Typical WSN installations are sending messages in intervals of multiple seconds or more. Therefore, the Sequence Number field has been shortened to 1 byte, which should cover a sufficiently long timespan. Since IPFIX packets are always transported via an underlying network protocol, which specifies the

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012

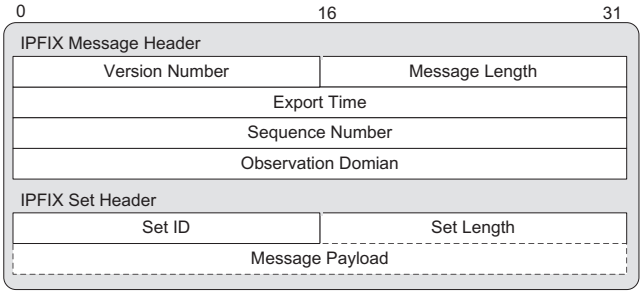

Fig. 3. General structure of IPFIX headers [bits].

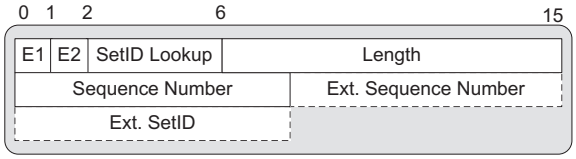

Fig. 4. Header used by aggressive TinyIPFIX approach [bits].

source of the packet, the *Observation Domain* can be equated with the source of an IPFIX packet and the field can be dropped from the header. The specification of a 32-bit time stamp in seconds would require time synchronization across the WSN and produce additional protocol overhead. Thus, the *Export Time* is dropped in TinyIPFIX. Applications requiring an exact time stamp for their measurements can define an according field in the template and send the measurement time stamp along with the sensor data in a data packet. By using the header compression technique the IPFIX Message Header (16 bytes) and the first Set Header (4 bytes) can be reduced from in total 20 bytes to only 3 bytes.

3.4. In-network aggregation

As mentioned at the beginning of the paper the resources are very limited for wireless sensor devices, which makes saving resources important. One technique for saving energy and computational capacities is in-network aggregation [22]. Aggregation is usually performed on aggregator motes, which have more resources and are located at selected positions within the network. The following two aggregation techniques are common and implemented in our TinyIPFIX-Aggregation framework:

1. *Message Aggregation (mode 1)*: Aggregation of several data messages in one packet:
   - Type A: Data Records refer same Template.
   - Type B: Data Records refer different Templates.
2. *Data Aggregation (mode 2)*: Data pre-processing within the transmission path to the gateway through aggregate functions.

The first aggregation type (mode 1) can easily be performed if the link layer MTU is big enough. In the case of TinyIPFIX we can split this case into two subclasses. In case 1.A several Data Records, which refer to the same Template, are transmitted in one message as shown in Fig. 2 where two Data Records are transmitted in one packet. The second possibility (1.B) is the combination of two different Data Records in one message, which refer to different Template Sets. In this case the combined Data Set must refer all needed Template Sets for decoding. An example is shown in Fig. 5. Assuming Fig. 1 setting and looking at the transmitted messages between the aggregator nodes (node ID = 3, 6, 0) towards the server, the number of transmissions is reduced by one.

The second type of aggregation (mode 2) uses the aggregate functions $f = max\{a, b\}, f = avg\{a, b\}$ and $f = min\{a, b\}$. The chosen aggregation function depends on the application and is applied to contemporary measurements rather than temporal series of measurements. For example, if only the maximum temperature in a room is relevant, as shown in Fig. 1 in the upper room, we can use $f = max\{a, b\}$. In this case the number of transmitted messages to the gateway node (node ID = 0) can be reduced by one in total.

As pointed out by Krishnamachar et al. [22], in-network aggregation leads to an increase in data latency, which depends on the performed aggregation mechanism.

4. An end-to-end cyber-physical system implementation

In this section we show the design and implementation of a 6LoWPAN/TinyIPFIX based end-to-end cyber-physical system. It consists of the WSN infrastructure, the server, and a connection to the cloud. We use the term *cloud* to refer to a pool of applications and services, which require and access the collected data of the WSN as illustrated in Fig. 1). For example, the project Autonomic Home Networking (AutHoNe) is a representative for a CPS in the cloud, which was tested in an office scenario [27]. All components in AutHoNe communicate over IPv6 and require different information of all components. In the case of the WSN the AutHoNe infrastructure requests, among others temperature and humidity values in order to regulate the climate control automatically by comparing monitored data with preset thresholds for individual rooms as briefly illustrated in Fig. 6 [23]. In order to allow this data transfer, parsers and interfaces are required, which are partly implemented in AutHoNe and in the WSN. A brief description is presented in parts of Section 5.

4.1. General design decisions on the WSN side

The choice of UDP over IPv6 as network and transport protocols allows the simple integration of motes into the network. A mature implementation of UDP/IPv6 for sensor networks exists in BLIP [11]. Fig. 7 shows the used network stack.

A matching application level protocol, which would facilitate the aforementioned properties while introducing limited protocol overhead, was still needed. We decided to use TinyIPFIX for this purpose since it is highly flexible while maintaining high transmission efficiency through the separation of meta data and sensor measurements into different messages. The Enterprise/Type ID metrology described in Section 3.2 allows the user to add new motes with sensors that were not known during the initial deployment, because the semantic data can be dynamically obtained, thereby fulfilling

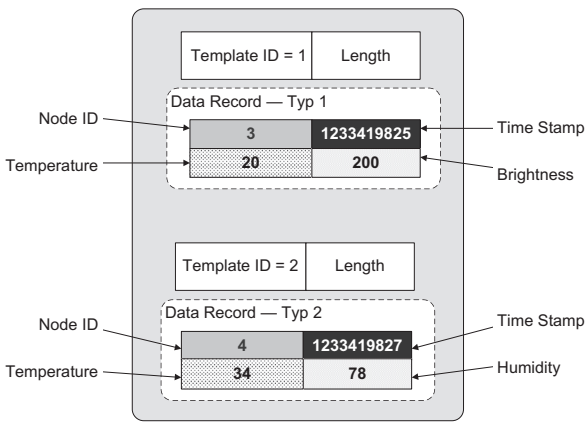

Fig. 5. Data Set for in-network aggregation of mode 1.B.

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012

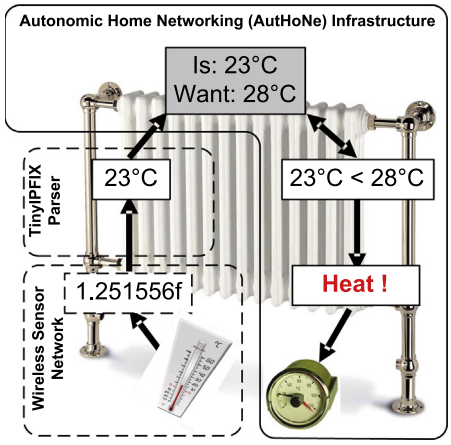

Fig. 6. Examplary information flow in AutHoNe.

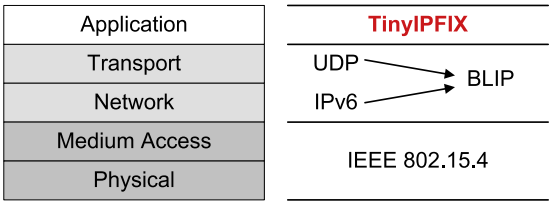

Fig. 7. Structure of network stack.

our requirement for easy configuration. Other protocols such as the XML based COAP [8] or the JSON based sMAP [24] also meet the aforementioned requirements. However we wanted a simpler protocol that introduces very little computational, message and implementation overhead. We decided in favor of TinyIPFIX, because it does not need to perform compression to achieve high transmission efficiency as shown in Section 5.2.2.

4.2. Data preprocessing by TinyIPFIX-Aggregation

Due to application requirements and limited resources of WSN components aggregation techniques are an attractive addition to cyber-physical system similar to those presented in this paper.

Our TinyIPFIX-Aggregation framework offers the user two modes of aggregation: (1) message aggregation and (2) data aggregation. Both techniques work with the TinyIPFIX message format as it is shown in the example of Fig. 8 [25,2].

The functionality of the protocol for message aggregation (mode 1) is shown in the lower room in Fig. 8. Sensor messages up to a certain amount are aggregated into newly generated appropriate messages. The number of message aggregated is referred to as the degree of aggregation (DoA), which depends on the aggregator's available memory as well as the number of sensor nodes in range of the aggregator, the acceptable message delay and application constraints. In the message aggregation mode information about the source sensor node of template and data sets is essential for reconstructing the data and must not be lost during the aggregation process. We address this requirement in our message aggregation algorithm. It is shown in Fig. 9 on the left side and consists of the following steps [25,2]:

1. **Buffer template sets:** Before data can be transmitted the IPFIX protocol requires the announcement of the related template. Those are buffered by the aggregator to a maximum amount, the degree of aggregation (DoA).

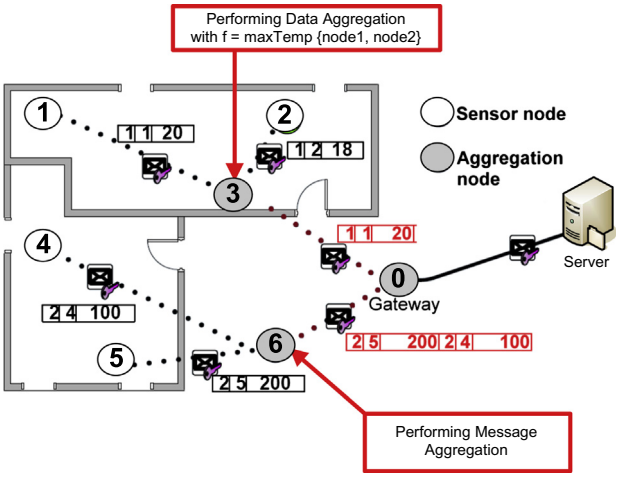

Fig. 8. Testbed 1: Overview of aggregation features. Black marked messages are original IPFIX records transmitted by sensor nodes. Red marked messages are IPFIX records transmitted by aggregators as result of aggregation functionality. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

2. **Buffer data sets:** Incoming data sets are also buffered and allocated to the corresponding template sets stored in the step before. If the requested template set is unknown the data set is dropped because it cannot be interpreted.
3. **Transmit the aggregated template set:** If the maximum amount of buffered template sets is reached, the aggregator announces the upcoming data with the aggregated template set. The template can be announced independently of whether or not the associated data sets have been received.
4. **Transmit the aggregated data set:** The aggregated data set is now transmitted.
5. **Update buffered data sets:** The updated data sets are assigned to their buffered template sets and send periodically after the degree of aggregation has been reached. The timespan for receiving new data information is generally shorter than the interval for receiving new template information.
6. **Update buffered template sets:** If the aggregator receives a new template definition the buffer for the template set is updated. As a result, the conditions for the aggregation of new incoming data sets change which requires the received data to be reallocated according to the updated template definitions.

The upper room shown in Fig. 8 is showing data aggregation (mode 2) in contrast to message aggregation (mode 1). Resulting messages transmitted by the aggregators are shorter than in normal message aggregation mode. The idea behind this implementation is that the aggregator computes aggregates on the received sensor readings from the sensor nodes by applying aggregate functions on specific values such as MIN, MAX or AVG, i.e. the aggregator performs semantic aggregation in mode 2 whereas only syntactic aggregation is performed in mode 1. Because bidirectional communication is provided, the aggregate functions and the sensor reading types can additionally be selected and changed during operation. Selecting the aggregate function and value type via UDP-Shell commands does this. The underlying algorithm for data aggregation consists of the following steps and is shown in Fig. 9 on the right side [25,2]:

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012

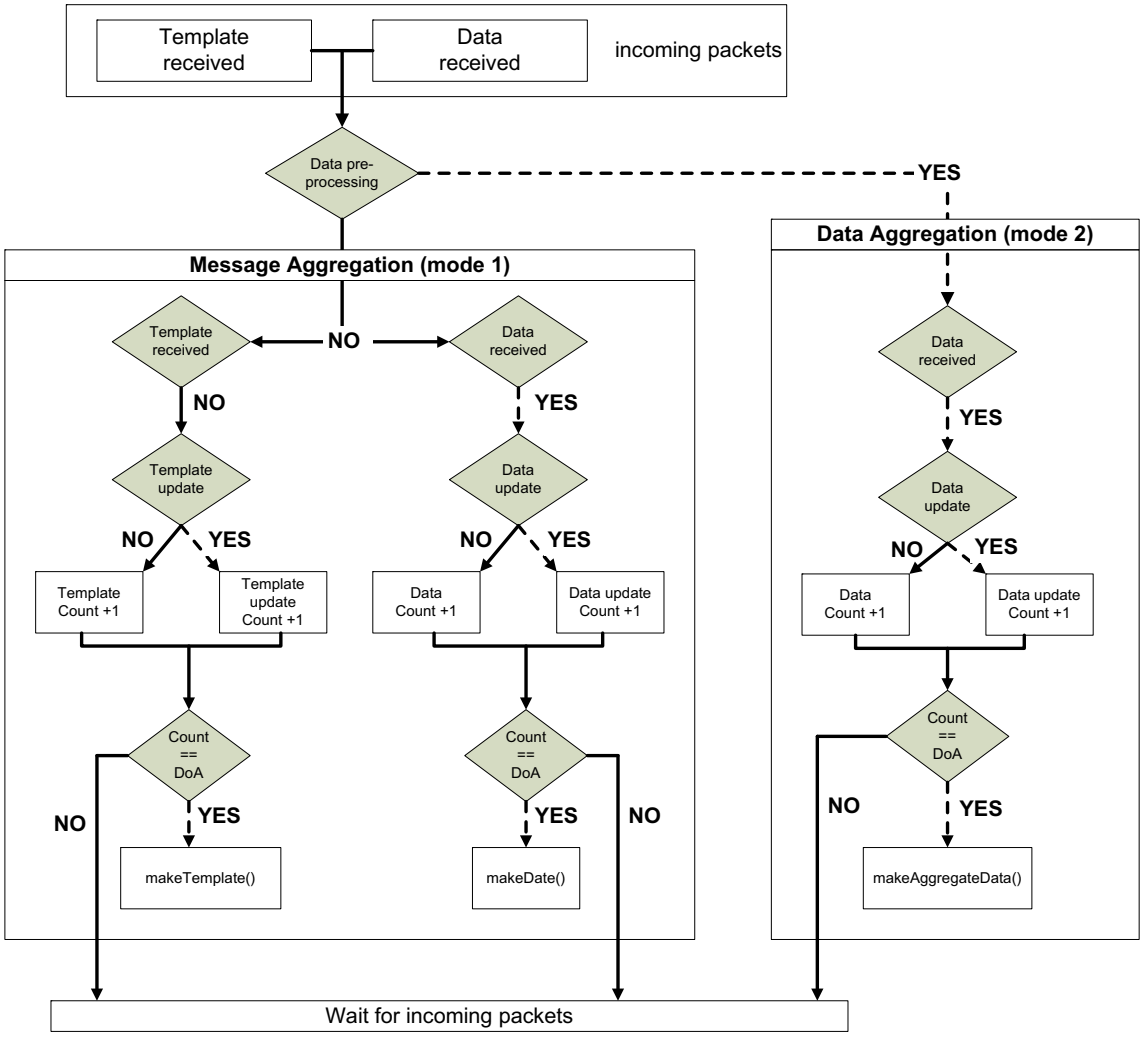

Fig. 9. Decision tree for TinyIPFIX-Aggregation.

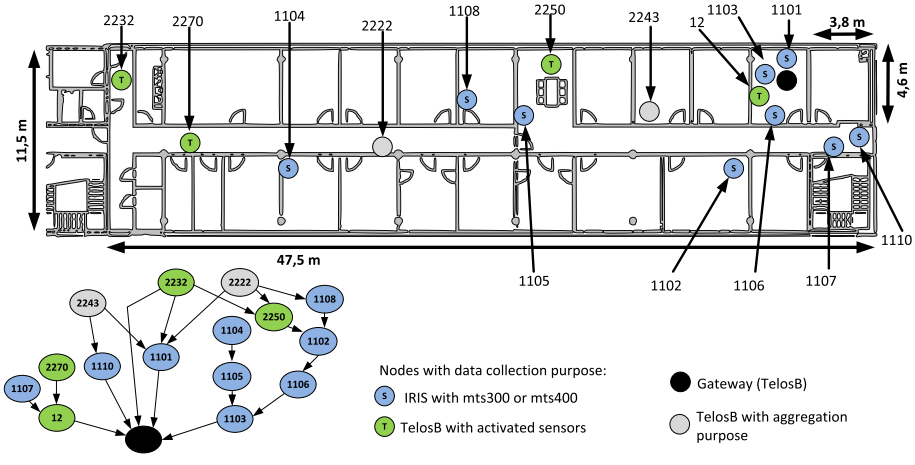

Fig. 10. Overview and routing tree of deployed network with heterogeneous node hardware.

1. **Transmit new template set:** Since the data pre-processing aggregation is driven by user requests for specific sensor readings, and, therefore, the recipient of the aggregated data already has knowledge of the meaning of the expected data, the transmission of a new template by the aggregator can be omitted.
2. **Buffer template set:** The aggregator buffers all incoming template sets from the sensor nodes till the buffer limit or the degree of aggregation is reached.
3. **Buffer data set:** Incoming data sets are associated with the matching template sets, which have been received earlier. After

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012

reaching the buffer limit or degree of aggregation for the data sets, the selected aggregate function is performed on the stored data sets.

4. **Compute the aggregation function:** In order to compute the selected aggregate function, sensor readings have to be converted from the sensors' encoding of the measurement to an universal format. The aggregator holds a lookup table for the data values announced by the according template set, with which it can identify the desired value type. It then computes the aggregate function on all buffered values of that type.
5. **Transmit the aggregated data set:** After computing the aggregate function, a newly generated data set for the demanded value type with the aggregated values is generated and transmitted to the next aggregator or gateway. Additional information on the aggregated value can also be transmitted in the data set, assuming that the recipient can decode the information properly.

5. Evaluation

In this section we evaluate TinyIPFIX extensively using the following metrics:

- *Metrology* ⟶ *Completeness and Generality*: We discuss how TinyIPFIX can be used in different application scenarios and for different kinds of transmitted data.
- *Resource Efficiency*: We evaluate the resource (RAM and ROM) consumption of the TinyIPFIX protocol, as well as the transmission efficiency in comparison to a common Type–Length–Value approach. Additionally we evaluate its system performance in a large WSN.
- *Scalability* ⟶ *Practical Implementation*: We show that TinyIPFIX is easy to program and present an example in TinyOS. The protocol can also be transferred to other operating systems if the implementation follows the description as published in reference [26].
- *Syndication* ⟶ *Application use*: We integrated a WSN performing TinyIPFIX in the home infrastructure management project AutHoNe [27]. The included components in AutHoNe can use the measured data for controlling purposes (e.g. climate control). By having an interchange format via XML, other applications are "portable" from a collection of similar sensors.

5.1. Complete and general

Due to the separation of measurement data and corresponding meta information in two different message types, the TinyIPFIX protocol is very flexible. A suiting Template Set can be defined for almost any application scenario and may be changed by the motes as needed. It must only be ensured that the modified Template Set is announced to the recipients of the Data Sets. An optional periodic rebroadcast of the Template helps to address the issue of unreliable communication protocols and also gives Collectors, which join the network at a later time, the possibility to parse the incoming sensor data. The TinyIPFIX protocol is also independent from the mote's hardware. Little to no changes (e.g. only changing the sensor code or adapting the maximum size of IPFIX packets) are needed to use TinyIPFIX on other platforms supported by TinyOS. So far, TinyIPFIX has been successfully tested on the *IRIS* and *TelosB* platforms.

So far we have focused on an office application scenario. In order to widen the scope we discuss how TinyIPFIX may be applied to some major deployments from IPSN/SenSys from the past few years. These deployments were chosen, because each of them differs in the way that they send data to the gateway nodes and, therefore, represent a wide spectrum of possible deployments. In the remainder of this section IPFIX describes the unaltered protocol while TinyIPFIX refers to the implementation with activated header compression (cf. Section 3.3). For optimization purposes the TinyIPFIX-Aggregation framework introduced in Section 4.2 is included in the upcoming evaluation.

HydroWatch is a sensor network deployed to monitor the "life cycle of water as it progresses through a forest ecosystem" [28]. It is based on the TMote-Sky, which is compatible with TelosB motes. It periodically samples the photo synthetically active radiation (Hamamatsu S1087) and the total solar radiation (Hamamatsu S1087-01), as well as temperature and humidity from a Sensirion SHT15. The encoding of these measurements in an IPFIX-Template are straightforward since the data flow model (periodic, single measurements, no back channel) is very similar to our home networking environment. Therefore, TinyIPFIX could have been used in this scenario without any changes to the protocol. Refs. [29–31] described a similar data flow pattern.

The goal of the project PermaSense is to collect geophysical data via 15 nodes installed in an alpine environment [32]. It samples rock temperature and electrical resistance at four different depths per node, as well as the internal voltage, temperature and humidity inside the enclosure of each node. One interesting point arises for how to encode having more than one measurement of the same type in one IPFIX packet. The basic idea of IPFIX for WSNs is to assign the Enterprise and Type IDs based on the manufacturer and model of the sensors used. In this case, one would have to deviate from this principle by defining an Enterprise ID for the project and separate Type IDs for each of the sensors at different depths. The semantic information would have to be supplied via an indirection, as an automatic lookup in a public repository via the Enterprise/Type ID combination would not yield the correct response. Instead one would have to instruct the program that queries the repository not to use the combination specified in the IPFIX template but the Enterprise and Type ID that corresponds to the correct sensor. Another point of interest comes from the properties of the data flow. PermaSense transmits data periodically as single measurements when a link between the gateway node and the sensor node exists. When the link is broken, e.g. when the mote is covered in snow, measurement data is logged and transmitted in bulk when connectivity is re-established. Assuming bulk transmission, filling the IPFIX data packet with data records until the maximum packet size allowed by the transmission protocol is reached, can increase the relative transmission efficiency of IPFIX.

The system Lance collects seismological data [33]. As such, it samples data at high rates (100*Hz* or higher) and performs event detection on the motes to decide, which parts of the recorded data are of potential interest since it is not feasible to transmit all measurements. The gateway node requests the bulk transfer of the data based on a cost model. Other application areas with similar modes of operation include habitat and structural integrity monitoring.

There is only one point where plain TinyIPFIX is struggling to achieve the required functionality. Currently it does not support a back channel as would be needed for the gateway node to request event records from the motes. This limitation could be fixed by introducing another template type for instructions: each node announces an instruction template after boot, which is periodically retransmitted. Any entity that wants to give a command to a node sends an IPFIX data packet that is in compliance with the previously announced instruction template. How to assign the Enterprise and Type IDs still remains an issue. However, IPFIX is fundamentally a PUSH-protocol and, therefore, does not natively support pulling data.

Transmission of binary data of variable length is supported by the IPFIX specification [14,9], although our implementation does currently not support this feature.

5.2. Resource efficiency

This section deals with evaluation of resource consumption of the TinyIPFIX implementation with aggregation support. It

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012

Table 1
Memory usage of BLIP and TinyIPFIX [bytes].

Component	RAM	ROM	TinyIPFIX packet
Scaffold	46	2826	–
BLIP	4738	23,012	–
TinyIPFIX	57	2972	0
	261	3182	102
	2105	3012	1024
Total	4841–6889	29,020	0–1024

is important to focus on resource consumption for long life support of the WSN, because the used sensor nodes (IRIS and TelosB) are constrained in memory, computational, and energy capacity.

5.2.1. Embedded implementation

We implemented TinyIPFIX on top of the BLIP IPv6 and UDP implementation, which comes as a part of TinyOS-2.1.1.[2]

In order to put the memory usage in perspective, each major component is considered separately. Since they build upon each other, the memory consumption has been measured in an incremental fashion, starting with a basic scaffold for obtaining measurements and expanding upon that by adding BLIP and TinyIPFIX. The results are shown in Table 1, which shows a maximum RAM consumption of 6889 bytes when using BLIP and setting the maximum TinyIPFIX package size to 1024 bytes. The memory consumption of the TinyIPFIX component is only 57 bytes plus twice the maximum defined IPFIX packet size. The memory consumption on IRIS is similar to that on TelosB.

5.2.2. Comparison to approaches using binary XML

XML [8] or JSON [24] based formats send meta data together with every sensor measurement and, thus, have a relatively low transmission efficiency. Previous work has shown that XML-data should be compressed and sent as *binary XML* [5], with XML schema aware techniques such as EXI [7] outperforming general compression algorithms such as gzip. At their core, schema aware built a dictionary of the tags used in the schema and prefix data items with indices into that dictionary. This approach is comparable to a *Type–Length–Value* (TLV) approach where a sensor measurement would always be prefixed with an index into a (perfect) dictionary, describing the type of a value and the length of the value that will follow in bytes. The entries in an IPFIX template boil down to the same approach; therefore we will use TLV with a length of 1 byte for the type field (TLV^8) and with 6 bytes for the type field (TLV^{48}) as the baseline. TLV^8 represents the minimal example whereas TLV^{48} represents a TLV approach offering the same range for the type field as IPFIX.

In order to give some concrete examples of how TinyIPFIX would perform compared to XML compression techniques we use the examples in Listings 1 and 2 provided by Ref. [34].

The encodings used in [34] were EXIficient v0.3 [7] and FastInfoset v1.1.9 [13] as well as a project specific binary XML implementation [35]. We are comparing this to IPFIX and TinyIPFIX data packets. The data costs for the template packet have been added proportionally to the size of the data packet for a data to template packet ratio of 32 (we will show in Table 4 later that this ratio achieves very high end-to-end data usable rates, e.g., approximately 97%, in a large multiple hop sensor network testbed). As can be seen in Table 2 the transmission efficiency of IPFIX matches

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <simpleMetering>
3   <CurrentSummationDelivered>0</
        CurrentSummationDelivered>
4   <Status>0</Status>
5   <UnitOfMeasure>0</UnitOfMeasure>
6   <SummationFormatting>85</SummationFormatting>
7   <MeteringDeviceType>0</MeteringDeviceType>
8   <InstantaneousDemand>1234</
        InstantaneousDemand>
9   <CurrentDayConsumptionReceived>23</
        CurrentDayConsumptionReceived>
10 </simpleMetering>
```

Listing 1. XML content used for a Smart Energy Meter. We assume a total size of 9 bytes for the values (five 8-bit integer and two 16-bit integer).

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <GPS_Temperature_System>
3  <TimeOfSample>2009-04-08T09:30:10+02:00</
       TimeOfSample>
4  <Longitude>6404.48220N</Longitude>
5  <Latitude>02430.91188E</Latitude>
6  <Temperature>20.50</Temperature>
7 </GPS_Temperature_System>
```

Listing 2. XML content used for a GPS and Temperature System. We assume a total size of 16 bytes for the values (one 32-bit time stamp and four 32-bit floating point numbers).

Table 2
Packet size of various encodings compared to a XML encoding. The relative size is given in percent. TLV^8 and TLV^{48} refer to one and 6 bytes for the length of the "Type" field.

Encoding	Listing 1	Listing 2
XML	409B	300B
EXIficient	13B (3%)	57B (19%)
Binary XML	210B (51%)	117B (59%)
FastInfoset	200B (49%)	185B (62%)
IPFIX	31B (8%)	34B (11%)
TinyIPFIX	13B (3%)	16B (5%)
TLV 2^8	16B (4%)	16B (5%)
TLV 2^{48}	51B (12%)	40B (13%)

or outperforms those of comparable approaches. Furthermore, some of the listed approaches struggle to reduce the transmission size to one that fits in a single IEEE 802.15.4 packet, thus requiring fragmentation and more energy for transmitting.

5.2.3. Energy efficiency

In order to obtain measurements of the energy spent for transmitting packets we connected an oscilloscope across a resistor (10 Ω) in the circuit of an external power source to a TelosB mote. This methodology is also described in detail in reference [36]. The TelosB features a CC2420 Radio chip, which is rated at 17.4 mA current draw when transmitting.[3] We then proceeded to measure the average transmission time over 128 samples of IPFIX, TinyIPFIX and TLV packets as they would occur when transmitting a time stamp (4 bytes), the node ID (2 bytes) and two sensor measurements (2 bytes each). The size of each packet, including meta data, is given in Table 3. Low power listening was disabled during our measurements.

[2] http://www.tinyos.net.

[3] Datasheet avaliable at http://www.ti.com/product/cc2420.

Table 3
Average transmission times and energy consumption. TLV^8 and TLV^{48} refer to one and 6 bytes for the length of the "Type" field.

Packet type	t_{send} (ms)	Payload (bytes)	Energy (μJ)
empty	10.48	0	699
TLV 2^8	10.93	14	730
TLV 2^{48}	11.55	34	778
IPFIX Data	11.69	30	779
IPFIX Template	12.3	48	820
TinyIPFIX Data	10.9	13	727
TinyIPFIX Template	11.71	31	780

Table 4
Percentage of packets that successfully arrived at a PC connected to the edge router along with the total energy consumed by the WSN. Retransmission intervals for data/template packets are specified in seconds. TLV^8 and TLV^{48} refer to one and 6 bytes for the length of the "Type" field.

Packets...	...Readable (%)	...Sent (kB)	Energy (J)
TLV^{48} 5 s	99.42	1179.27	66.188
IPFIX 5 s/180 s	98.41	827.8	68.246
TinyIPFIX 5 s/180 s	99.42	400.7	63.024
TLV^{48} 15 s	99.68	390.9	24.491
IPFIX 15 s/480 s	98.60	280.9	24.254
TinyIPFIX 15 s/480 s	97.23	135.8	23.046

After recording the measurements shown in Table 3 we found that the time, when energy is consumed, is largely dominated by a constant factor which stems from the medium access protocol and the time it takes to switch the radio from receiving to sending mode and back. When activating low power listening the variance and average duration of the transmission time increased up to one order of magnitude. Therefore, the employment of IPFIX, TLV or any other application layer protocol does not have a large impact on overall energy usage when using the default TinyOS settings and relatively small packages. Nevertheless, the settings for the back-off period can be changed as detailed in [37], potentially leading to shorter overall transmission times. Furthermore, other medium access protocols, such as TDMA, can also increase the impact of transmission efficiency.

5.2.4. System level energy consumption

In order to test if TinyIPFIX scales to large networks with more complex routing paths and longer distances between nodes compared to our functional experiments (see Fig. 10), we tested TinyIPFIX on Motelab[4] [12], a sensor network testbed provided by Harvard University. Of the 184 motes installed 77 were functional at the time of testing, meaning we could run our experiments with 76 motes sending sensor measurements to a single gateway node over a maximum number of six hops. Each experiment lasted for 30 min with a start up phase of two minutes in which no sensor measurements were sent to allow BLIP to establish the routes in the network. Afterwards the nodes would take temperature, humidity and internal voltage measurements and send them with their mote ID and the ticks since they booted to the edge router. Every time a mote tried to send an IPFIX or TLV packet it would log the attempt to a database. This figure was then compared with the number of packets captured in Wireshark on the receiving end to determine the percentage of packets that had arrived successfully. If an IPFIX packet containing sensor data could not be parsed because it was referencing an unknown template it was counted as unreadable. In order to calculate the amount of energy spent by the whole WSN during a test run we collected statistics on the number of sent or forwarded packets from each node and multiplied this number with our measurements for energy usage from Table 3. Our aim was to get a realistic estimate for the amount of energy consumed, which does not include the lower layer, e.g., routing and overhead. Therefore, any additional packets that could not be traced back to an IPFIX Template or Data packet, such as topology maintenance packets, were treated like empty packets from Table 3.

The figures in Table 4 show a high overall percentage of data packets being delivered and readable when arriving at the processing application although IPFIX has a slightly lower success rate than TLV. This is due to some Template Packets being lost during the initial announcement rendering the following data packets unreadable. The high savings in the amount of transferred data – TinyIPFIX only transmits around 35% of the amount that TLV sends with our settings – did not translate to similar savings in energy consumption. The WSN is consuming 5% less energy when using TinyIPFIX compared to using TLV, which could be expected given the slim difference between the amounts of energy needed to send packets of different length. These results encourage using aggregation and bulk forwarding, an area in which compressed IPFIX should have an advantage over a TLV approach: By limiting the amount of meta data needed more sensor data can be fitted in a data packet leading to less packets being transmitted and, thus, an overall reduction in energy consumption.

5.2.5. Impacts of in-network aggregation

In-network aggregation has different impacts on the performance of the implementation presented in this paper. In Section 3.4 we described the implemented aggregation modes in detail [2,25].

For the upcoming evaluation on reduction of transmitted messages we assume a modified testbed, called tesbed 1, with node deployment as shown in Fig. 8. Fig. 11 shows the data packet capture. The testbed consists of several nodes where nodes marked in gray work as aggregators. Further we assume that the nodes with IDs 1, 2, 4, and 5 transmit their sensor readings on a regular basis in accordance with the TinyIPFIX protocol described in Section 3.

In order to analyze the impact of aggregation the following cases are considered [2,25]:

1. No TinyIPFIX aggregation is performed on any node in testbed 1. The gray marked nodes just forward the received data down to the gateway without any modification on the data. Twelve messages are transmitted in total in this case.
2. TinyIPFIX aggregation is performed on three nodes (gray marked) in the testbed 1, which only results in seven messages being transmitted in total.

As a result of the described setup the aggregation functionality reduces the number of transmitted messages by 42%. Here a degree of aggregation (*DoA*) of two messages per aggregate was used and the simple message aggregation (mode 1) was performed. If additionally the data aggregation functionality (mode 2) is performed the same result can be shown together with a reduction of the transmitted packet size, due to the computation of the aggregate function on the sensors' values. The reduction of transmitted messages in this example is only related to the reduction of transmitted TinyIPFIX messages. We did not take the impact of the message reduction on the amount of control messages into account. However, we expect that it should be reduced as well because fewer packets are transmitted overall which leads to less congestion in the network. [2,25]

We also observed energy savings in parallel to the message reduction. In order to achieve energy savings on radio transmissions for the aggregator, the additional transmission time for the aggregated packets must be compared to the transmission time that is necessary to forward the unaggregated packets. We assume

[4] http://motelab.eecs.harvard.edu.

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012

```
|+--[fec0:0:0:0:0:0:0:8ae]:4740[2222], Data: 256 received Jun 22, 2012 11:56:50 AM
|
|----- Sound (MTS300)[2] (3844 - 32769): 466
|----- Temperature[2] (3843 - 32771): 27.8 °C
|----- NodeTime[4] (1 - 32770): 73.24 sec
|----- NodeID[2] (1 - 32769): 1212
|----- Temperature[2] (3847 - 32769): 27.67 °C
|----- Humidity (Sensiron SHT11)[2] (3841 - 32770): 35 %
|----- Light (TAOS TSL2550)[2] (3845 - 32769): 65535 LUX
|----- Voltage MTS400[2] (3846 - 32769): 0.14 V
|----- NodeTime[4] (1 - 32770): 78.13 sec
|----- NodeID[2] (1 - 32769): 2202

|+--[fec0:0:0:0:0:0:0:89a]:20679[2202], Data: 256 received Jun 22, 2012 11:56:54 AM
|
|----- Temperature[2] (3847 - 32769): 27.67 °C
|----- Humidity (Sensiron SHT11)[2] (3841 - 32770): 35 %
|----- Light (TAOS TSL2550)[2] (3845 - 32769): 65535 LUX
|----- Voltage MTS400[2] (3846 - 32769): 0.14 V
|----- NodeTime[4] (1 - 32770): 83.16 sec
|----- NodeID[2] (1 - 32769): 2202


|+--[fec0:0:0:0:0:0:0:4bc]:20679[12152], Data: 256 received Jun 22, 2012 11:56:55 AM
|
|----- Sound (MTS300)[2] (3844 - 32769): 465
|----- Temperature[2] (3843 - 32771): 27.9 °C
|----- NodeTime[4] (1 - 32770): 83.01 sec
|----- NodeID[2] (1 - 32769): 1212
```

Fig. 11. Transmitted data packets captured by TinyOS-Listener.

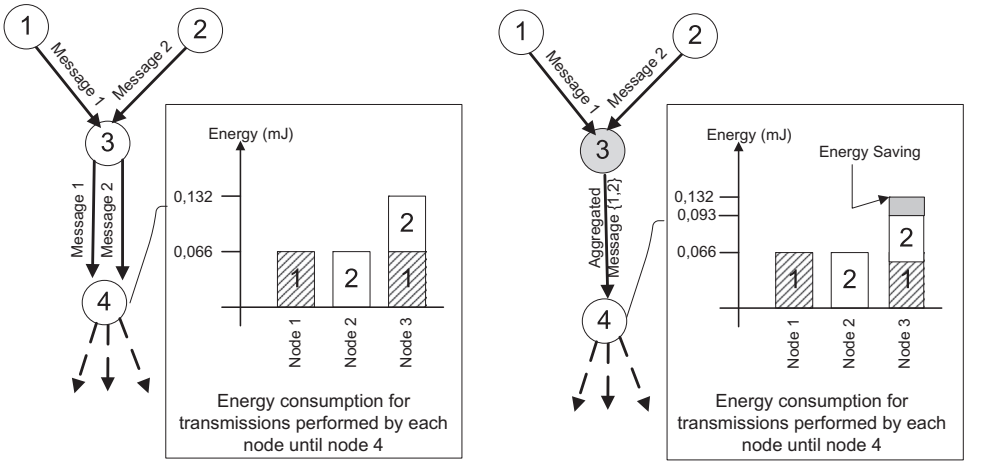

Fig. 12. Average CC2420 energy consumption per mote transmitting TinyIPFIX packets.

$DoA = 2$ as above. The trade off between the necessary average transmission energy for forwarding two TinyIPFIX packets and the necessary average transmission energy for the aggregated packet for TelosB in comparison to just forwarding functionality without any aggregation mode is shown in Fig. 12. If data is transmitted in an aggregated format we gain a saving of 30% compared to transmission of the same number of packets in individual transmissions over the CC2420 radio of TelosB. [2,25].

Similar results were also achieved in bigger testbeds as shown in Fig. 10 and in runs on Motelab consisting of 40 TelosB nodes during our test-runs [12].

The implemented aggregation techniques lead to increased end-to-end transmission latency. This occurs because a $DoA > 1$ requires the aggregator to wait for more than one incoming packet before performing the chosen aggregation. In our testbeds the measurement and transmission intervals were configured to relatively high frequencies. Thus, no latency could be observed if aggregation in both modes were performed. A more detailed analysis of latency is currently a work in progress.

5.3. Scalability

The existing TinyIPFIX implementation is based on TinyOS 2.x, and has been tested successfully for TelosB and IRIS motes. We believe that it could also support other hardware platforms, which feature IEEE 802.15.4 radios with little modification.

Due to the intuitive structure of IPFIX it is easy for a programmer to tailor the Template/Data set pair to the application requirements. For example, to add a new temperature sensor to the sensor node's programming, the following changes are

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012

Table 5
Total RAM and ROM consumption for TinyIPFIX including aggregation and BLIP as the underlying communication stack.

Components	RAM (bytes)	ROM (bytes)
Boot, Leds, Timer, Blip	4766	25344
UDP Socket	2	296
UDP Shell	288	4074
TinyIPFIX	559	2160
Aggregation	404	3600
Total	6019	35474

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <match>
    <field nosend="FALSE">
      <name>Temperature</name>
      <fieldID>0x80A0</fieldID>
      <enterpriseNumber>0xF0AA00AA</
          enterpriseNumber>
      <dataType>int16</dataType>
      <expression> (1024 - x) / (1+ log(x))</
          expression>
    </field>
    <field nosend="FALSE">...</field>
  </match>
  <match source="fec::800">
    <field nosend="TRUE">
      <name>Sound from Node 2048</name>
      <fieldID>0x80A1</fieldID>
      <enterpriseNumber>0xF0AA00AA</
          enterpriseNumber>
      <dataType>int16</dataType>
    </field>
  </match>
</configuration
```

Listing 3. XML parser example for AutHoNe.

needed: Define IPFIXDataSampler interface and instantiate it with the desired Type ID and enterprise ID. In a next step instantiate the fitting Sensor and wire it to the Sensor interface of the IPFIXDataSampler. Finally, wire all IPFIXDataSamplers to the Sampler interface of the main Application by performing the following three steps:

1. Create a new instance of IPFIXDataSampler, a wrapper interface that annotates a sensor reading with the matching Enterprise and Type ID: `components new IPFIXDataSamplerl6C(uintl6_t type_id, uint32_t enterprise_id) as Temp;`.
2. Wire the sensor interface to the IPFIXDataSampler: `Temp.Sensor → Shtll.Temperature;`.
3. Add the IPFIXDataSampler instance to the application: `App.Sampler → Temp;`.

The program can now automatically generate templates for all connected sensors and obtain their measurement data, which is then automatically encapsulated in a format that complies with the previously generated template. To allow for more fine grained control over the creation of a TinyIPFIX packet the programmer can also use the methods provided by the TinyIPFIX implementation such as `tinyIPFIX.start_template_record (uint8_t buffer_no, uintl6_t template_id)` to start handcrafting a template record or `tinyIPFIX.start_data_record (uint8_t buffer_no)` for writing multiple data records into a single packet. Reference [1] outlines the features of the TinyIPFIX implementation in more detail.

The consumed ROM and RAM space of the presented work is shown in Table 5. The reported figures make the protocol viable for use on constrained hardware. Due to the modular structure of TinyOS different parts of the implemented protocol can be excluded for very limited hardware and included on nodes with more resources in a heterogeneous network as shown in Fig. 10 [2]. This advantage makes the TinyIPFIX protocol attractive for all common node platforms.

5.4. TinyIPFIX applications and syndications

In our application scenario the first task was to seamlessly integrate a WSN into an already existing smart home infrastructure (AutHoNe) that supports IPv6. As pointed out before in Section 4, the requirements are an XML parser (cf. example in Listing 3 and the reference to a previously announced Template (cf. Section 3.1). Additionally, due to the idea of the Internet of Things IPv6

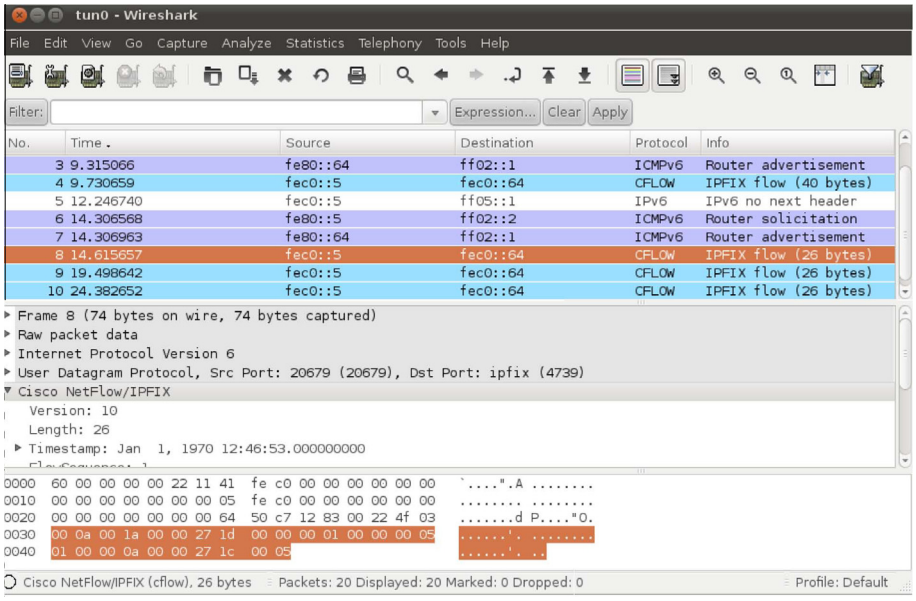

Fig. 13. IPv6 packets arriving from the WSN.

communication is required. The TinyIPFIX data received at the server must be translated and forwarded to the cloud of application as illustrated in Fig. 1. Responsible for the translation is the XML parser included (cf. Fig. 1 server part). When a packet arrives at the server, the payload including all required information is extracted from the messages and with the help of the XML parser, which specifies the interpretation of the data, translated to readable information for the upcoming data processing (cf. Fig. 6). AutHoNe can be such an application where all components access data of different sources using IPv6 communication as briefly indicated in the beginning of Section 4 [27].

Fig. 13 shows a capture of incoming IPv6 packets by Wireshark. A network tunnel with IP-Address fec0::64, running on the gateway, is functioning as destination address for the packets from the WSN. The gateway also runs a conversion program between IPFIX and TinyIPFIX. To receive IPFIX packets, an application program listens on the appropriate UDP port (e.g., 4739, as defined by the IETF) and handles the arriving payload, which is marked in Fig. 13. One can quickly see the version field of the uncompressed header ($0 \times 000a$), followed by the length ($0 \times 001a$ = 26) and all other elements of the valid IPFIX message.

6. Conclusion

Cyber-physical systems, such as intelligent building control systems like AutHoNe, are needed to achieve both a raised level of comfort for the inhabitants along with a high overall energy efficiency of the building. Wireless sensor networks as part of a smart meter infrastructure provide the required level of data quality with temporally and spatially fine-grained measurements.

In order too facilitate the efficient transfer of sensor data through a heterogeneous WSN with minimal configuration upon deployment we utilize TinyIPFIX – a versatile and lightweight application level protocol for data exchange. In this paper we briefly described the modifications performed to adapt standard IPFIX to the resource constrained environment of sensor networks and evaluated the performance of the TinyIPFIX implementation with regards to transmission efficiency, system level energy consumption and memory requirements. The rate of successfully transmitted measurement was similar to conventional approaches even though the separation of meta data and measurement data into Template and Data Records in different packets increases the risk of unreadable data. The savings in the amount of transmitted data did not translate directly into energy savings on the system level although a reduction of about 5% could be achieved. Additional energy savings of up to 30% can be gained with the integration of aggregation functionality in TinyAggregation. If mode 1 of the message aggregation is used, 0039 mJ can be saved per aggregated transmission. If the data pre-processing of mode 2 is performed, more energy can be saved due to reduced message sizes within the network. The aggregation functionality can also be changed during the system run on the fly, which is a big advantage to react directly on system changes.

TinyIPFIX is demonstrated to be a suitable choice for the studied CPS and we believe that it could be leveraged in other scenarios and deployments as we have outlined. Based on the shown results an exploration of how TinyIPFIX and IPFIX could be bridged started that investigated the development of an IPFIX Mediator like device analog to RFC 6183 [38]. The idea of such a mediator is to take measurements from a wireless sensor network and send them across a network using IPFIX with full functionality. Especially as the benefits of TinyIPFIX disappear at higher levels of spatial and temporal aggregation the request for an adapted mediator exist. Thus parts of the authors decided for standardization under IETF and published a draft including this topic [39]. Further investigations will take place within the FLAMINGO project at University of Zurich [40].

References

[1] T. Kothmayr, C. Schmitt, L. Braun, G. Carle, Gathering sensor data in home networks with IPFIX, in: Proceeding of the European Conference on Wireless Sensor Networks, EWSN, Springer, 2010, pp. 131–146.
[2] C. Schmitt, Secure data transmission in wireless sensor networks, net 2013-07-2, department computer science, Technische Universität München (Jul. 2013), http://dx.doi.org/10.2313/NET-2013-07-2.
[3] Global Mobile Data Traffic Forecast Update, 20122017, White Paper, Cisco Systems, Inc., February 2013.
[4] L. Atzori, A. Iera, G. Morabito, The internet of things: a survey, Comput. Networks 54 (15) (2010) 2787–2805.
[5] N.B. Priyantha, A. Kansal, M. Goraczko, F. Zhao, Tiny web services: design and implementation of interoperable and evolvable sensor networks, in: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems, SenSys, ACM, 2008, pp. 253–266.
[6] C.J. Augeri, D.A. Bulutoglu, B.E. Mullins, R.O. Baldwin, L.C. Baird III, An analysis of XML compression efficiency, in: Proceedings of the Workshop on Experimental Computer Science, ExpCS, ACM, New York, NY, USA, 2007.
[7] J. Schneider, T. Kamiya, Efficient XML Interchange (EXI) Format 1.0, W3C Working Draft 19.
[8] Z. Shelby, K. Hartke, C. Bormann, B. Frank, Constrained Application Protocol (CoAP), IETF Draft, Work in Progress, RFC Editor, March 2013. <http://tools.ietf.org/html/draft-ietf-core-coap-14>.
[9] B. Claise, B. Trammell, P. Aitken, Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information, RFC 7011 (INTERNET STANDARD), September 2013. <http://www.ietf.org/rfc/rfc7011.txt>.
[10] J.W. Hui, D.E. Culler, IPv6 in low-power wireless networks, Proc. IEEE 98 (11) (2010) 1865–1878.
[11] S. Dawson-Haggerty, A. Tavakoli, D. Culler, Hydro: a hybrid routing protocol for low-power and lossy networks, in: First IEEE International Conference on Smart Grid Communications, SmartGridComm, 2010, pp. 268–273.
[12] G. Werner-Allen, P. Swieskowski, M. Welsh, MoteLab: a wireless sensor network testbed, in: 4th International Symposium on Information Processing in Sensor Networks, IPSN, 2005, pp. 483–488.
[13] P. Sandoz, A. Triglia, S. Pericas-Geertsen, Fast Infoset, Sun Developer Network.
[14] B. Claise, Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information, RFC 5101 (Proposed Standard), obsoleted by RFC 7011, January 2008. <http://www.ietf.org/rfc/rfc5101.txt>.
[15] B. Claise, G. Dhandapani, P. Aitken, S. Yates, Export of Structured Data in IP Flow Information Export (IPFIX), RFC 6313 (Proposed Standard), July 2011. <http://www.ietf.org/rfc/rfc6313.txt.>
[16] S. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, TAG: a tiny aggregation service for ad-hoc sensor networks, SIGOPS Oper. Syst. Rev. 36 (SI) (2002) 131–146.
[17] B. Przydatek, D. Song, A. Perrig, SIA: secure information aggregation in sensor networks, in: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys, ACM, 2003, pp. 255–265.
[18] T. He, B.M. Blum, J.A. Stankovic, T. Abdelzaher, AIDA: adaptive application independent data aggregation in wireless sensor networks, ACM Trans. Embed. Comput. Syst. Special Issue Dyn. Adaptable Embed. Syst. 3 (2) (2004) 426–457.
[19] M. Johnson, Network Monitoring: What You Need to Know for It Operations Management, Tebbo, 2011.
[20] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler, Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC 4944, September 2007. <http://tools.ietf.org/html/rfc4944>.
[21] T. Kothmayr, Data Collection in Wireless sensor Networks for Autonomic Home Networking, Master's thesis, Department Computer Science, Technische Universität München, 2010.
[22] B. Krishnamachari, D. Estrin, S.B. Wicker, The impact of data aggregation in wireless sensor networks, in: Proceedings of the 22nd International Conference on Distributed Computing Systems, ICDCSW, IEEE Computer Society, 2002, pp. 575–578.
[23] M.-O. Pahl, C. Niedermeier, M. Schuster, A. Müller, G. Carle, Knowledge-based middleware for future home networks, in: IEEE IFIP Wireless Days Conference Paris, Paris, France, 2009.
[24] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, D. Culler, sMAP: a simple measurement and actuation profile for physical information, in: Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, SenSys, ACM, 2010, pp. 197–210.
[25] B. Ertl, Data Aggregation using TinyIPFIX in Wireless Sensor Networks, Bachelor Thesis, Department Computer Science, Technische Universität München (Aug. 2011).
[26] C. Schmitt, L. Braun, G. Carle, IPFIX for Wireless Sensors, IETF Draft, Work in Progress, IETF, October 2009. <http://tools.ietf.org/html/draft-schmitt-6lowapp-ipfix-ws-00>.
[27] Autonomic Home Networking DE Project, 2012. <http://www.authone.de>.
[28] J. Taneja, J. Jeong, D. Culler, Design, modeling, and capacity planning for micro-solar power sensor networks, in: Proceedings of the 7th International Conference on Information Processing in Sensor Networks, IPSN, IEEE Computer Society, 2008, pp. 407–418.

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012

[29] L. Mo, Y. He, Y. Liu, J. Zhao, S.-J. Tang, X.-Y. Li, G. Dai, Canopy closure estimates with GreenOrbs: sustainable sensing in the forest, in: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys, ACM, 2009, pp. 99–112.
[30] C.-J.M. Liang, J. Liu, L. Luo, A. Terzis, F. Zhao, RACNet: a high-fidelity data center sensing network, in: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys, ACM, 2009, pp. 15–28.
[31] O. Chipara, C. Lu, T.C. Bailey, G.-C. Roman, Reliable clinical monitoring using wireless sensor networks: experiences in a step-down hospital unit, in: Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems- SenSys, ACM, 2010, pp. 155–168.
[32] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, M. Yuecel, PermaDAQ: a scientific instrument for precision sensing and data recovery in environmental extremes, in: Proceedings of the International Conference on Information Processing in Sensor Networks, IPSN, IEEE Computer Society, 2009, pp. 265–276.
[33] G. Werner-Allen, S. Dawson-Haggerty, M. Welsh, Lance: optimizing high-resolution signal collection in wireless sensor networks, in: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems, SenSys, ACM, 2008, pp. 169–182.
[34] Z. Shelby, M. Luimula, D. Peintner, Efficient XML Encoding and 6LowApp, IETF Draft, Work in Progress, RFC Editor, October 2009. <http://tools.ietf.org/html/draft-shelby-6lowapp-encoding-00>.
[35] M. Luimula, Z. Shelby, J. Tervonen, J. Markkula, P. Weckström, P. Verronen, Developing geosensor network support for locawe platform: application of standards in low-rate communication context, in: Proceedings of the 2009 International Conference on Pervasive Services, ICPS, 2009, pp. 73–82.
[36] Y. Panthachai, P. Keeratiwintakorn, An energy model for transmission in telos-based wireless sensor networks, in: International Joint Conference on Computer Science and Software Engineering, JCSSE, 2007.
[37] David Moss, Jonathan Hui, Philip Levis, Jung Il Choi, 2007. <http://www.tinyos.net/tinyos-2.x/doc/html/tep126.html>.
[38] A. Kobayashi, B. Claise, G. Muenz, K. Ishibashi, IP Flow Information Export (IPFIX) Mediation: Framework, RFC 6183 (Informational), April 2011. <http://www.ietf.org/rfc/rfc6183.txt>.
[39] L. Braun, C. Schmitt, B. Claise, G. Carle, Compressed IPFIX for Smart Meters in Constrained Networks, IETF Draft, Work in Progress, September 2011. <http://tools.ietf.org/html/draft-braun-core-compressed-ipfix-03.txt>.
[40] FLAMINGO Consortium, FLAMINGO: Management of the Future Internet, March 2014. <http://www.fp7-flamingo.eu/>.

Please cite this article in press as: C. Schmitt et al., TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems, Comput. Commun. (2014), http://dx.doi.org/10.1016/j.comcom.2014.05.012