REST-based sensor networks with OData

Matthias Thoma*†, Theofilos Kakantousis*, Torsten Braun†
matthias.thoma@sap.com, tkak@kth.se, braun@iam.unibe.ch
*SAP (Switzerland) Inc., Althardstrasse 80, 8105 Regensdorf, Switzerland
†Communication and Distributed Systems, University of Bern, Neubrückstrasse 10, 3012 Bern, Switzerland

***Abstract*—RESTful services gained a lot of attention recently, even in the enterprise world, which is traditionally more web-service centric. Data centric RESfFul services, as previously mainly known in web environments, established themselves as a second paradigm complementing functional WSDL-based SOA. In the Internet of Things, and in particular when talking about sensor motes, the Constraint Application Protocol (CoAP) is currently in the focus of both research and industry. In the enterprise world a protocol called OData (Open Data Protocol) is becoming the future RESTful data access standard. To integrate sensor motes seamlessly into enterprise networks, an embedded OData implementation on top of CoAP is desirable, not requiring an intermediary gateway device. In this paper we introduce and evaluate an embedded OData implementation. We evaluate the OData protocol in terms of performance and energy consumption, considering different data encodings, and compare it to a pure CoAP implementation. We were able to demonstrate that the additional resources needed for an OData/JSON implementation are reasonable when aiming for enterprise interoperability, where OData is suggested to solve both the semantic and technical interoperability problems we have today when connecting systems.**

I. Introduction

Recent advances in the typical protocol stack of Wireless Sensor Networks (WSNs), in particular the use of IP technology, and the demand of businesses for real-time monitoring and real-time decision support has increased the need of enterprise systems to interoperate directly with wireless sensor nodes. One of the major benefits that comes with 6LoWPAN based networking is the use of standard technologies and common and well-understood architectures to integrate smart objects into enterprise systems.

In a typical 6LoWPAN[1] based WSN protocol stack, there is a trend towards applying already existing application level protocols and paradigms. In a (networked) enterprise architecture as it exists today, one can observe two main paradigms: (Web-) services, for example, like SOAP and standardized by a variety of standards known as WS-* and REpresentational State Transfer (REST)[2]. REST architectures have become particularly important in Internet of Things applications, as sensors and actuators often can be naturally represented as resources identified by URIs.

The Open Data Protocol (OData)[3] is a data access protocol based on widely-used technologies (HTTP, AtomPub and JSON). Compared to the formerly predominant SOAP services, it follows a REST-based approach for a variety of data sources by defining a standardized interface.

The contribution of this work is: Introducing OData into sensor networks by presenting an implementation of the OData protocol over the Constraint Application Protocol (CoAP)[4], considering both standalone scenarios as well as utilizing an intermediary. We evaluate different deployment choices and deduce recommendations for the interaction between the motes, an intermediary and an enterprise system.

II. Related Work

Driving the World Wide Web, the best known RESTful protocol is HTTP[5]. HTTP uses a set of standardized verbs (for example GET, PUT, POST, DELETE) to create, retrieve, update or delete resources. These operations are known under the term CRUD. The HTTP protocol has been applied to sensor networks and smart objects (known as "Web of Things"). Guinard et al. [6] adapted patterns and technologies from the web towards smart objects. In particular, they suggest to use REST-style HTTP and ATOM[7]/JSON[8].

Due to the verbosity of HTTP, for Sensor Networks a protocol called Constrained Application Protocol (CoAP) has been suggested and is currently being standardized by the IETF[4]. CoAP shares many characteristics of HTTP like the verbs, but is more than a simple binary scaled-down version of HTTP. CoAP uses an interaction model similar to HTTP, but typically acts in server and client roles. As CoAP has been specifically designed for constrained environments it has low overhead and low parsing complexity. Compared to HTTP, it is designed to work with a non-reliable transport layer (UDP) and does not rely on the transmission control provided by TCP. The protocol, therefore, provides means for reliable transmission and message de-duplication. Colitti et al. [9] compare the performance of HTTP and COAP for constrained scenarios: They showed that CoAP has lower response time and less protocol overhead than HTTP. The number of bytes transmitted and received with CoAP (in a typical sensor application) was reduced to 17% compared to HTTP. Instead of seventeen 802.15.4 packets in the HTTP case, only two packets were needed for CoAP. The response time of CoAP was around 1/10 of a similar HTTP call. It has also been demonstrated that CoAP has significantly lower energy consumption compared to HTTP[10]. CoAP is often mentioned in conjunction with the CORE Link Format[11]. The Core Link Format is used by constrained servers to describe the resources they host, their attributes, and further relationships between links. Such discovery of resources is used to provide URIs (links) for the resources hosted by the

Theofilos Kakantousis' current affiliation is: Royal Institute of Technology (KTH), Stockholm, Sweden

978-1-4799-4937-3/14/$31.00

server. The CORE Link format shares some similarities with the Metadata information provided by OData. Webservices with CoAP and XML compression have been investigated [12] as an alternative to traditional SOAP-based webservices.

Webservices, as part of a Service Oriented Architecture (SOA), are commonly implemented using various W3C WS-* specifications. Glombitza et al. [13] demonstrate the use of SOAP within a sensor network as part of a business process. Moritz et al. [14] introduced a SOAP-over-CoAP binding, allowing to use SOAP in CoAP-based WSNs.

Alternatives to OData include GData and RDF from the W3C. GData is a protocol from Google which shares fundamental ideas with OData but failed to gain widespread adoption outside of Google. Furthermore, even inside Google not all APIs, especially the more recent ones, follow the GData approach.

III. ODATA

A. Overview

The Open Data Protocol (OData) is a data access protocol based on widely-used technologies (HTTP, AtomPub and JSON). Compared to the formerly predominant SOAP services, it follows a REST-based approach for a variety of data sources by defining a standardized interface. OData consists of the following four main parts:

- **OData protocol**: OData specifies a protocol defining how clients can query and manipulate data sources. It supports CRUD operations and different serialization formats: Atom Syndication Format and JSON. OData defines a query language as an extension of the URI. This query language provides a set of query options that allow clients to specify the data they are interested in.
- **OData data model**: The structure of the data is defined by an abstract data model called Entity Data Model (EDM). It can be seen as realization of the well known entity relationship model, where data is modelled as entities and associations among those entities. An OData service provides a Service Metadata Document that defines the EDM-based model of the service in the XML-based Conceptual Schema Definition Language (CSDL).
- **OData service**: An OData service exposes a callable endpoint that allows accessing data or calling functions. It implements the OData protocol and uses the OData data model.
- **OData client**: An OData client accesses an OData service through the OData protocol and the known OData data model.

B. Services, Resources and Filters

OData uses URIs to reference resources and to specify queries. An URI as used in OData can consist of three different parts: A service root URI, the resource path and a query. The service root URI identifies the root of an OData service. The resource path identifies the resource the service consumer wants to interact with (for example a specific temperature sensor, or some actor). Commonly, such a resource addresses a collection of entities, e. g. several sensors, or a single entity, like one specific temperature sensor.

Op	Description	Op	Description
Eq	Equal	Not	Logical Negation
Ne	Not equal	Add	Arithmetic Addition
Gt	Greater than	Sub	Arithmetic Subtraction
Ge	Greater than or equal	Mul	Arithmetic Multiplication
And	Logical and	Div	Arithmetic Division
Or	Logical or	Mod	Arithmetic Modulo

TABLE I
ODATA OPERATORS (EXCERPT)

Function	Description
bool startswith(string p0, string p1)	Checks if string p0, starts with the string p1
int length(string p0)	Length of string
string trim(string p0)	Removes whitespaces at beginning and end
string toupper(string p0)	Transforms to upper case
string tolower(string p0)	Transforms to lower case
double round(double p0)	Arithmetic rounding
double floor(double p0)	next lowest integer value by rounding down

TABLE II
ODATA FUNCTIONS (EXCERPT)

A typical OData URI looks as follows:

$$\underbrace{\text{http://services.sap.com/service.svc}}_{\text{service root URI}}\underbrace{\text{/sensor/temp?}}_{\text{resource path}}\underbrace{\text{\$filter=temperature gt 20}}_{\text{query}}$$

The query can consist of one or more pre-defined options (called *system query options*), user defined *custom query options*, or *service operation parameters*. Service operations are functions exposed by an OData service in a RESTful style. These operations might require zero or more parameters, which are passed as part of the query string. In this work we will mainly concentrate on the built-in system query options. We briefly introduce the most important system query options:

- *orderby* allows clients to request resources in a particular order. This is comparable to an SQL orderby clause.
- *top* allows to retrieve only the first n-results of a result set.
- *expand* allows clients to request related resources when a resource that satisfies a particular request is retrieved.
- *select* (projection) is used to select certain properties only.
- *filter* identifies a subset of the entries from the collection of entries identified by the resource path. The subset is determined by filtering out the Entries that satisfy the expression specified by the filter query option. Some of the operators are listed in Table I and Table II. For a complete list please refer to [3].
- *format* is used to identify the data format requested by the client.

Discovering the capabilities of an OData service is possible through the *Service Document* and the *$metadata* information. The service document allows to discover the locations of the available collections of resources. It is returned when doing a get request on the service URI. This is a *must*

have feature according to the OData protocol specification. Additionally, every service should present information about the structure and organization of all the resources. This is done by appending a $metadata path segment to the path. The result is in Common Schema Definition Language (CSDL) [15] format.

IV. Deployment options

For exposing smart items or sensor networks towards enterprise systems with OData we identified several deployment options. The three options are illustrated in Figure 1. In this section we will discuss deployment considerations of implementing an OData-enabled system.

The first option is illustrated in Figure 1a: OData can be used to access whole sensor networks (Entity Model), with a gateway as entry point. The sensor network itself is communicating internally with a different protocol stack, that is typically tailored towards low energy consumption or low latency. The application logic for making requests and setting up the sensor network to satisfy a request resides in the gateway. The OData gateway can either be a sensor node itself, or a platform with more computing power. This approach is not different from any other gateway or proxy approach, as the sensor network is completely independent from the enterprise communication. We will not explore this option further and instead consider only scenarios were the motes themselves communicate through OData.

The second option is shown in Figure 1b. The enterprise system interacts directly with single motes over the OData protocol. This pattern is usually applied when single board sensor platforms or embedded devices are used. While out of scope in this paper, this is also the pattern used when mobile phones are used as sensing devices.

As OData queries can be rather complex and might involve time and energy consuming processing, we evaluated an assisted mode. Here the OData enabled smart item can request support from an external, most likely more powerful, processing unit. This is illustrated in Figure 1c. The assisted approach allows the service consumer to send arbitrary complex requests to the OData-enabled mote and also makes the service transparent, as the consumer only communicates with a very constrained device and does not notice any differences compared to simple OData queries. Constraining the type or size of requests would limit the potential number of service consumers and would make it necessary for them to ensure certain preconditions before issuing requests to the motes.

V. OData Stack on the Mote

OData was originally designed to work with HTTP. Nonetheless every HTTP-like CRUD based protocol is a suitable option for OData, as long as it can be mapped to HTTP. A CoAP to HTTP mapping has been demonstrated by [16]. In this work, we will base our implementation on CoAP as a protocol to communicate with the motes directly and HTTP only for communication with an intermediary.

In Figure 2 the standard OData stack is compared with our CoAP-based stack. The network layer is in both cases IP-based: IPv4/IPv6 vs. 6LoWPAN in IoT. The transport layer in

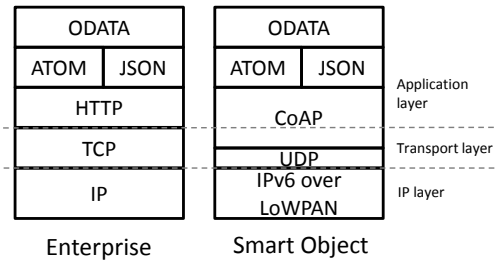


Fig. 2. OData stack on enterprise system vs. OData stack on mote

TABLE III
TECHNICAL DETAILS OF AN IRIS MOTE

CPU	ATmega1281 (8Mhz)
Serial Flash	512k Bytes
Program Flash	128k bytes
RAM	8k bytes
Current	8mA(act), 8μA(sleep)
RF power	3 dBm (typ)

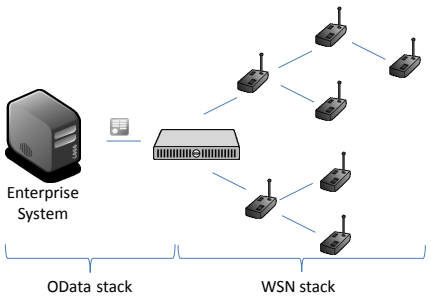
a typical enterprise OData stack is the TCP protocol, while in our IoT-Stack it is a combination of UDP for transport and (parts of) CoAP for transmission control. On top of that there is HTTP or CoAP, respectively. Considering the traditional ISO/OSI stack, CoAP can be seen as a cross-layer protocol being on one hand above the transport layer providing more or less the same functionality as HTTP while on the other hand it incorporates elements of TCP (transmission control, message deduplication). On top of HTTP/CoAP the stack is identical. Data is transported in either ATOM/XML or JSON format while the data and resource handling is done as specified by the OData protocol itself. An OData query in CoAP notation looks as follows:

$$\underbrace{\text{coap://}}_{\text{protocol}}\ \underbrace{\text{[a:b:c:d:e:f]/service.svc}}_{\text{IPv6 address of mote and service}}\ \underbrace{\text{/sensors/temperature?}}_{\text{resource}}\ \underbrace{\$\text{filter=temp gt 20}}_{\text{query}}$$

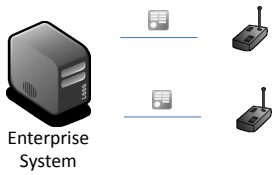
Instead of HTTP, the CoAP protocol is used. We directly address the mote through its IPv6 address which is then routed over 6LoWPAN. The service caller is not aware that its request goes to a mote. All other parts of the request stay the same. The only limitation is that each request needs to fit into one CoAP and thus one UDP packet.

VI. Implementation

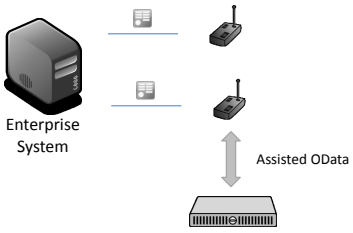
We built a prototypical implementation of our system. All software was written in Java. We are using the Moterunner [17][18] operating system from IBM Research for our evaluation. The Moterunner system allows to run java bytecode on the motes. It provides a custom compiler optimized for embedded systems, with a good energy profile [19], and a custom runtime system. We are using a custom written CoAP-18 compliant Java-based implementation on the motes. For our experimentation we are using IRIS motes with technical details as in Table III. It supports the base CoAP-18, as well the blockwise transfer and observe extensions. In cases where an assisted system is used, we were running on a Raspberry Pi with 512MB RAM and an ARM1176JZF-S 700 Mhz CPU.



(a) Intermediate OData Gateway



(b) Direct access to single motes



(c) Assisted OData processing

Fig. 1. Deployment Options

VII. Evaluation

In this section we perform a quantitative analysis of our OData implementation. We are using the software and hardware as described in Section VI. First, we will present the experimental setting we used to perform our experiments (Section VII-A). As outlined in Section IV we are currently considering two scenarios: Direct access from the Enterprise System to the Mote, which we evaluate in section VII-B. Afterwards, in section VII-C we evaluate the assisted mode by adding a more powerful system to which the motes can delegate work.

A. Experimental Setting

All the experiments were performed with IRIS motes, with technical details as in Table III. The motes were running the Moterunner operating system with a 6LoWPAN network stack, with CDMA MAC access, in a 2-hop setting. Energy measurements were performed within the Moterunner simulation environment.

The OData service we are exposing consists of one mote with three different sensors (temperature, humidity and light), where each has a unique ID, a name and can return a value (data). As outlined in Section III-A, service discovery is done through the service description and the metadata information. The service description of our IoT-Service looks as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<service xml:base="http://tmpsvc.sap.com/OData.svc/" xmlns
     ="http://www.w3.org/2007/app" xmlns:atom="http://www.w3
     .org/2005/Atom">
  <workspace>
    <atom:title>Default</atom:title>
-   <collection href="Sensors">
      <atom:title>Sensors</atom:title>
    </collection>
  </workspace>
</service>
```

The JSON representation is more compact and therefore more suitable for constraint devices:

```
{"odata.metadata":"http://tmpsvc.sap.com/OData.svc/\
     $metadata","value":[{"name":"Sensors","url":"Sensors
     "}]}
```

The $metadata keyword, providing information about the sensors looks as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.
     microsoft.com/ado/2007/06/edmx">
<edmx:DataServices m:DataServiceVersion="3.0" m:
     MaxDataServiceVersion="3.0" xmlns:m="http://schemas.
     microsoft.com/ado/2007/08/dataservices/metadata">
<Schema Namespace="Mote" xmlns="http://schemas.microsoft.
     com/ado/2009/11/edm">
<EntityType Name="Sensor">
<Key><PropertyRef Name="ID" /></Key>
<Property Name="ID" Type="Edm.Int32" Nullable="false" />
<Property Name="Name" Type="Edm.String" m:FC_TargetPath="
     SyndicationTitle" m:FC_ContentKind="text" m:
     FC_KeepInContent="false" />
<Property Name="Data" Type="Edm.String" m:FC_TargetPath="
     SyndicationSummary" m:FC_ContentKind="text" m:
     FC_KeepInContent="false" />

<EntityContainer Name="APPService" m:
     IsDefaultEntityContainer="true">
<EntitySet Name="Sensors" EntityType="Mote.Sensor" />
</EntityContainer>
```

The metadata encodes that our mote has entities of type *Mote.Sensor*, which have certain properties (ID, Name and Data) as well as corresponding datatypes. Furthermore, the set Sensors as specified in the service description is further defined to be type Mote.Sensor.

As baseline for a system that is not OData enabled we choose pure CoAP. The message payload is in that case encoded as comma delimited property:value pairs, such as shown in the following code fragment:

```
id:0,name:temperature,data:42
```

A similar means of discovery as the OData service discovery is the CoRE Link Format[11], accessible through the .well-known/core interface. It should be noted, that CoRE Link

Format is performing *Resource Discovery*, while OData aims for *Service Discovery*. The CoRE Link Format provides Web Linking as specified in RFC5988[20] and can be used to discover the links hosted by a CoAP server. It returns information in link-header style format [20]. A minimal resource description for a similar resource based access to the mote could look as follows:

```
</temp>;rt="temperature";ct=0;if="sensor"</hum>;rt="
    humidity";ct=0;if=sensor,</light>;rt="light";ct=0;if="
    sensor"
```

First, the resource is named (e. g. /temp) then the resource type (rt) and the ct. The rt attribute is string used to assign an application-specific semantic type to a resource [11]. It specifies the interface to be used. This is also an application specific string. The ct attribute specifies the content type as described in the CoAP specification [4].

We evaluated both compressed and uncompressed responses for ATOM and JSON. Previous research [21] has shown that standard compression schemes do work well on very constrained devices, for that reason we are also applying a LZW compression algorithm. The CoAP response did not shrink in size, so the uncompressed version was used as baseline. We did not use special compression schemes for XML like EXI, as they are not supported by OData clients and furthermore the memory was too constrained to run it on our platform. [22] describe such a platform for Contiki and 8kb platforms, nonetheless without the support of EXI schema encoding or any decoding. Furthermore [22] do give only one data point, no information on further compression restrictions nor about the energy or memory consumption. Our offline experiments with EXIficient, also in non schema mode, showed that the compression of XMI was for our result set not significantly better than the one provided by LZW. Only Q_1^A was significantly better with EXI (20%), than with traditional compression schemes. As this calculation was done offline it is unclear if the additional EXI-overhead on mote would actually provide a better performance or energy profile. Furthermore, it is unclear if it would actually fit (resource-wise) on a 8kb mote if an application level protocol like OData is used, and/or further application logic is running on the mote.

Q_1	`GET coap://[]:1024/OData.svc/sensors`
Q_2	`GET coap://[]:1024/OData.svc/sensors(0)`
Q_3	`GET coap://[]:1024/OData.svc/sensors(0)/ID`
Q_4	`GET coap://[]:1024/OData.svc/sensors(0)/Name`
Q_5	`GET coap://[]:1024/OData.svc/sensors(0)/Data`
Q_6	`GET coap://[]:1024/OData.svc/sensors?$filter=Name ne barometric`
Q_7	`GET coap://[]:1024/OData.svc/sensors?$filter=Data gt 42`
Q_8	`GET coap://[]:1024/OData.svc/sensors?$filter=Data eq 60 and Name eq humidity`

TABLE IV
QUERIES

B. Direct Access

In the following we investigate a typical IoT-scenario in which a backend system is directly communicating with the mote. The queries used to evaluate the properties of the system are listed in Table IV. The computational complexity increases with each Q_i. In the rest of the paper we will reference to these queries as Q_i^m, where i is the query number as listed in Table IV and m being either A for OData/ATOM, J for OData/JSON and C for CoAP, with or without the compression suffix CP

The resource consumption of the on-mote implementation is shown in Table V. The available space is the free space left on a IRIS mote running with its runtime system, the 6LoWPAN assembly for communication and our CoAP/OData implementation, including code for sensing temperature and humidity. The flash usage consists of program code itself (bytecode) and temporary data stored on the mote. As the XML was too large to be kept in RAM, most parts are stored on Flash and loaded into RAM on demand. Furthermore, the service metadata document is also stored in flash.

	COAP			JSON			ATOM		
	Stack	Heap	Flash	Stack	Heap	Flash	Stack	Heap	Flash
Q_1	232	3284	3850	292	3460	3931	292	3552	4255
Q_2	232	3252	3850	292	3432	3931	292	3476	4255
Q_3	232	3244	3850	292	3436	3931	292	3408	4255
Q_4	232	3248	3850	292	3440	3931	292	3496	4255
Q_5	232	3248	3850	292	3448	3931	292	3416	4255
Q_6	232	3364	3850	292	3524	3931	292	3616	4255
Q_7	232	3316	3850	292	3500	3931	292	3524	4255
Q_8	232	3404	3850	292	3528	3931	292	3548	4255

TABLE V
MEMORY CONSUMPTION (MAXIMUM, IN BYTES)

It can be seen the memory consumption of the OData/ATOM implementation is larger than the memory consumption of JSON. Nonetheless, the JSON implementation is comparable to a pure CoAP implementation in terms of memory. The actual payload of each of the queries is shown in Table VI.

Query	ATOM	ATOM/CP	JSON	JSON/CP	COAP
Q_1	1643	906	191	156	63
Q_2	604	443	110	105	23
Q_3	99	96	74	74	4
Q_4	94	88	85	84	13
Q_5	87	82	76	76	4
Q_6	1643	908	191	127	63
Q_7	1199	734	148	100	39
Q_8	761	523	108	108	18

TABLE VI
PAYLOAD SIZE (IN BYTES)

For each query $Q_1 ... Q_8$ we measured the service access time, i. e., the time from issuing a request by the service consumer until the answer has been decoded. The results (av-

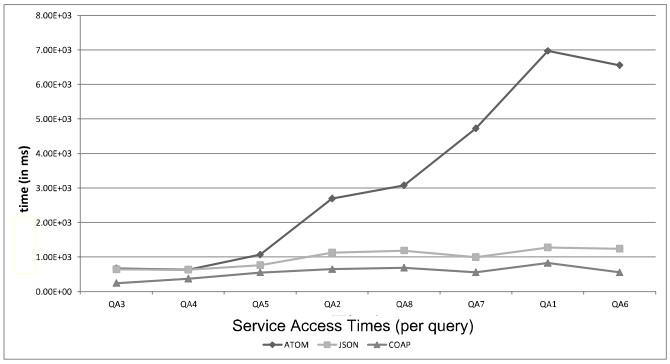


Fig. 3. Service access time (in ms, blockwise-transfer with block size 64 Byte) for queries Q_1 to Q_8 sorted by Q_n^A payload size

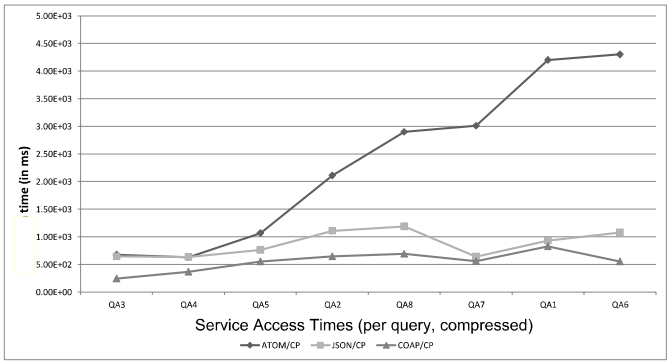


Fig. 4. Service access time (in ms, blockwise-transfer with block size 64 Byte) for queries Q_1 to Q_8 (compressed) sorted by Q_n^A payload size

eraged over 100 runs) are shown in Figure 3 for uncompressed data and in Figure 4 for compressed data. For small result sets the difference between the three formats is negligible. Larger data sets change the situation. Compression decreases the service access time considerably. CoAP and OData/JSON stay at low service access times, but the ATOM format increases the service access time very fast because of the amount of data to be transmitted. Compression does not help much in case of ATOM. The CoAP block size also affects the service access time. A 6LoWPAN packet, even if fragmented, has shown to be more efficient than the CoAP block option in our experimental setting, which did not suffer from high packet loss rates. Nonetheless, in cases of packet loss or when fragmentation is not available or exceeds the fragmentation capabilities of the system, CoAP blockwise transfer is to be used. The relationship between CoAP block sizes and the resulting service access times are shown in Figure 5. In case of a pure CoAP response, the response always fitted into one IEEE 802.15.4 frame, so there was no fragmentation of pure CoAP packets.

In Figure 6 and Figure 7 we show the energy consumption of OData/ATOM, OData/JSON and CoAP once for a blocksize of 64 bytes and 256 bytes. The energy consumption evaluations were performed within the moterunner simulation environment. It is measured in mAs serving 100 requests each. The energy consumption of JSON is comparable to CoAP, when reasonable block sizes are chosen. Smaller blocksizes

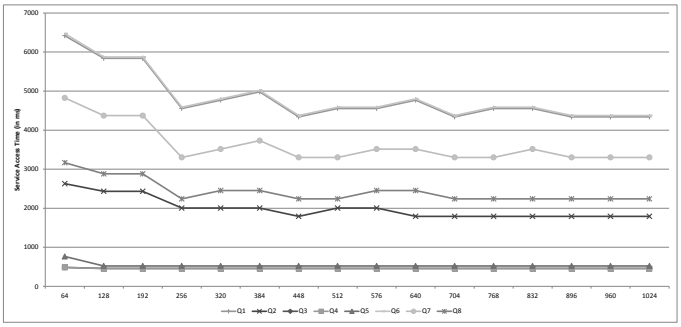


Fig. 5. Service access times of queries Q_1^A to Q_8^A with different CoAP block sizes (in simulation)

lead to more data transfer for the block requests and responses, as well as more computation to process these requests. The advantage of a smaller blocksize is increased reliability and in case of very lossy network increased throughput. The ATOM response needs most energy, for the obvious reason of more air time, but also as there are many transfers from flash memory to RAM.

C. Assisted Mode

OData requests can be very computation and memory intensive, or might even be beyond the capabilities of the mote, like complex floating point operations. As a countermeasure to such exceptional cases or uncommon uses of the system with arbitrary complex requests we implemented an "assisted mode", that means that we provide a dedicated system with more processing power. Without this assisted mode the mote would otherwise have to return an error code, indicating that the request exceeds its resources or the service consumer would have to wait for the request beyond reasonable duration and most likely outside of any timeout period.

Q_9	`GET coap://[]:1024/OData.svc/sensors?$filter=substringof("temp",Name) eq true`
Q_{10}	`GET coap://[]:1024/OData.svc/sensors?$filter=filter=concat(concat(Name,','),Area) eq temperature,Bern`
Q_{11}	`GET coap://[]:1024/OData.svc/sensors?$filter=filter=concat(concat(concat(Name,',',),Name ,',',),Area) eq temperature,humidity,Bern=3`

TABLE VII
COMPLEX STRING QUERIES

We evaluated more complex queries, including string operations and storing information on the mote's persistent storage, but still within reasonable limits for the mote to execute. Table VII shows the three additional queries. Q_{11} utilized the persistent memory for string operations.

The energy consumption measurements of the more complex queries are shown in Figures 8 and 9. As can be seen, apart from reducing the computational complexity which might not even allow the mote to execute the query, the energy consumption of the assisted mode is only slightly higher due to the additional communication needed.

The service access times (averaged over 100 runs) are shown in Table VIII. As shown, from a service access time point-

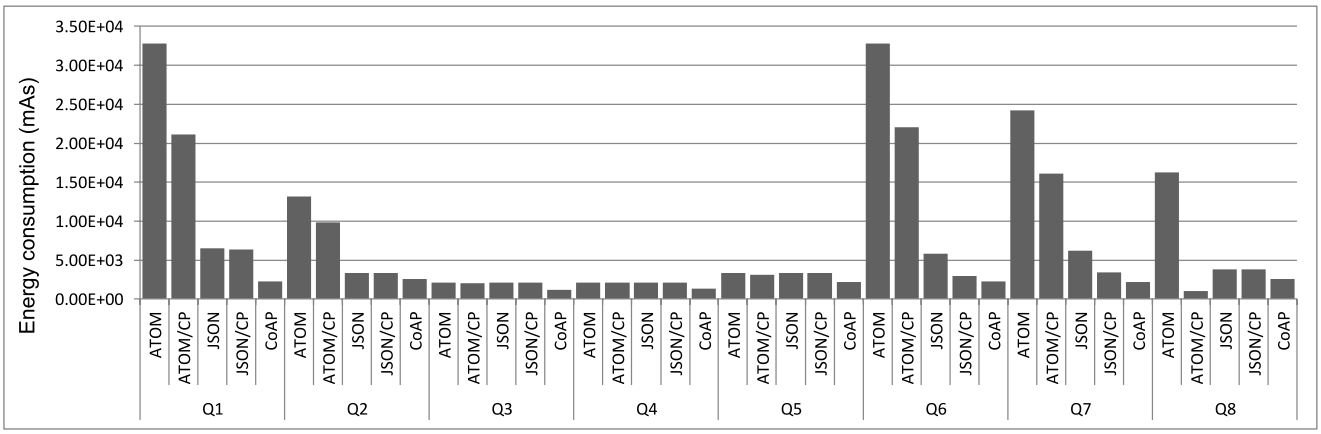


Fig. 6. Energy consumption (in mAs, 100 requests, blocksize 64 Byte)

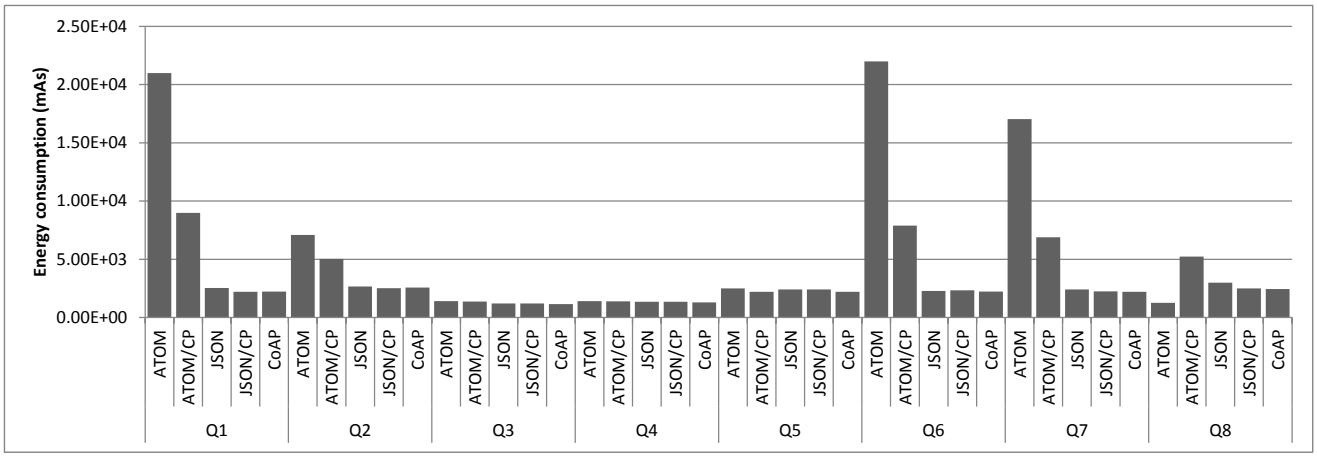


Fig. 7. Energy consumption (in mAs, 100 requests, blocksize 256 Byte)

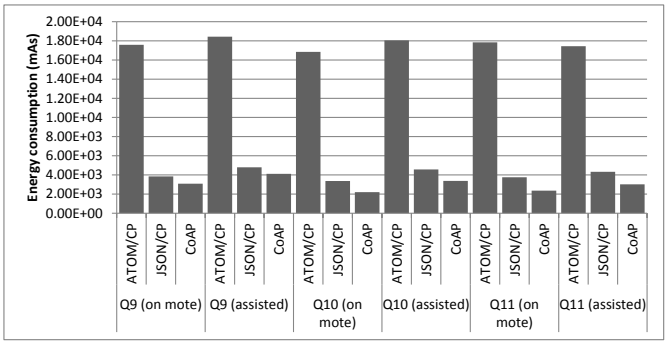


Fig. 8. Energy consumption (in mAs, 100 requests, blocksize 64 Byte)

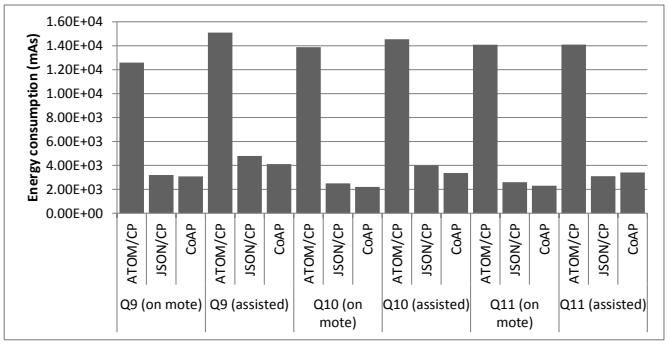


Fig. 9. Energy consumption (in mAs, 100 requests, blocksize 256 Byte)

of-view, the assisted mode was faster for Q_{11} and nearly as performant as some OData operations that have a high computational complexity (like string operations) with a lot of memory accesses. When ATOM and JSON are used, which usually is the case in such deployments, string operations utilizing the mote's persistence storage is a costly operation which the server can effortlessly handle with a negligent overhead, enabling the client to experience faster service access time. The small overhead introduced by the roundtrip times for sending a request to the assistance system, processing it there, and sending the response back compared to performing all operations on the mote, shows that using an assistant system can be firmly used for processing arbitrary queries. This approach alleviates both the computational load on the mote and increases the QoS provided to the service consumer as the mote is less likely to experience performance issues.

VIII. Conclusions

We evaluated the OData protocol as an end-to-end solution for the integration of REST-based sensor networks into enter-

	ATOM/CP		JSON/CP		CoAP	
	Avg	σ	Avg	σ	Avg	σ
Q_9 (mote)	2392.22	29.27	781.72	28.78	521.40	33.94
Q_9 (assisted)	2521.35	25.21	795.12	19.39	648.36	46.31
Q_{10} (mote)	2602.92	41.13	780.56	21.35	524.42	32.46
Q_{10} (assisted)	2805.13	22.13	826.54	24.89	642.60	17.79
Q_{11} (on mote)	2115.16	39.21	893.62	22.75	579.28	17.77
Q_{11} (assisted)	2095.65	31.42	824.13	27.32	532.02	19.93

TABLE VIII
SERVICE ACCESS TIMES (IN MS), AVERAGE AND STANDARD DEVIATION σ

prise IT systems. We concentrated on direct communication with the Mote, because a gateway solution is no different than any other gateway-based proxy. Nonetheless, it is notable that the OData entity abstraction fits a typical sensor network usage very well, including abstracting the topology and the sensors (or actors) as entities. This makes OData an appropriate abstraction for systems that can be seen as sensor network databases. Querying the sensor network becomes easily possible, as well as accessing temporal data, and can be done in a standardized RESTful way, which is immediately accessible by other applications.

Within a very constrained environment, as provided by the IRIS platform with its limited RAM, the differences between the resource usage in terms of memory, processing time and energy consumption of OData/JSON, compared to a pure CoAP solution, was small. XML processing should be avoided whenever possible though. JSON was superior to ATOM in almost every aspect. Transmission of large ATOM files should be even if compressed, whenever possible, redirected to a system with more performance capabilities. For really limited devices, when further business processes are running on the mote or as a countermeasure for arbitrary complex queries, we evaluated adding an assistance system. From a resource consumption point of view and from a general IoT point of view the assisted mode should be avoided. However, we were able to show that the penalty of using such a system is often comparable to do the entire processing on the mote, when feasible. In some cases, especially when long running or energy intense operations would have to be executed on the mote, the assisted mode has a slight advantage from a resource and response time point of view. Furthermore, the compression schemes on these platforms should have only a very small fingerprint. When running business processes on motes it is not feasible to spend most of the RAM on compression or decompression, especially when also intermote communication should be possible as an option. OData/JSON proved to be a valid alternative to pure CoAP solutions, providing enterprise and other systems direct access to data-driven IoT-devices. In contrast to a pure CoAP solution, direct and transparent access is possible. Now, for an enterprise system a sensor node or a wireless sensor network is nothing more than just another datasource. It can be used in business processes like any other datasource, without a need for adapters or special low-level information.

ACKNOWLEDGMENT

We would like to thank the Moterunner Team at IBM Research Zurich for their support, especially Marcus Oestreicher. Furthermore Klaus Sperner and Carsten Magerkurth for fruitful discussions, as well Martin Zabel for his help in finalizing the paper.

REFERENCES

[1] Z. Shelby and C. Bormann, *6LoWPAN: the wireless embedded internet*. Wiley, 2011, vol. 43.
[2] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.
[3] Microsoft Corporation, "Open Data Protocol (OData) Version 3.0," Microsoft, 2013.
[4] Z. Shelby, K. Hartke *et al.*, "Constrained Application Protocol (CoAP), Internet-Draft," 2011. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-coap-18
[5] R. Fielding, J. Gettys *et al.*, "Hypertext transfer protocol–http/1.1," 1999.
[6] D. Guinard, V. Trifa *et al.*, "A resource oriented architecture for the web of things," *Proc. of IoT*, 2010.
[7] M. Nottingham and R. Sayre, "The atom syndication format," 2005. [Online]. Available: http://tools.ietf.org/html/rfc4287
[8] D. Crockford, "The application/json media type for javascript object notation (json)," 2006. [Online]. Available: http://tools.ietf.org/html/rfc4627
[9] W. Colitti, K. Steenhaut *et al.*, "Evaluation of constrained application protocol for wireless sensor networks," in *Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*. IEEE, 2011, pp. 1–6.
[10] ——, "Integrating wireless sensor networks with the web," *Extending the Internet to Low powerand Lossy Networks (IP+ SN 2011)*, 2011.
[11] Z. Shelby, "Constrained restful environments (core) link format," 2012. [Online]. Available: http://tools.ietf.org/html/rfc6690
[12] A. P. Castellani, M. Gheda *et al.*, "Web services for the internet of things through coap and exi," in *Communications Workshops (ICC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–6.
[13] N. Glombitza, D. Pfisterer *et al.*, "Integrating wireless sensor networks into web service-based business processes," in *4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*. New York, USA: ACM, 2009, pp. 25–30.
[14] G. Moritz, F. Golatowski *et al.*, "A lightweight soap over coap transport binding for resource constraint networks," in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*. IEEE, 2011, pp. 861–866.
[15] Microsoft Corporation, "Open Data Protocol (OData) Version 3.0 — Common Schema Definition Language (CSDL)," Microsoft, 2013.
[16] A. P. Castellani, A. Rahman *et al.*, "Best practices for http-coap mapping implementation," 2013. [Online]. Available: http://tools.ietf.org/html/draft-castellani-core-http-mapping-07
[17] A. Caracas, T. Kramp *et al.*, "Mote runner: A multi-language virtual machine for small embedded devices," in *SENSORCOMM'09*. IEEE, 2009.
[18] A. Caracas and T. Kramp, "On the Expressiveness of BPMN for Modeling Wireless Sensor Networks Applications," in *Business Process Model and Notation*, ser. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2011, vol. 95, pp. 16–30.
[19] A. Caracas, C. Lombriser *et al.*, "Energy-efficiency through micro-managing communication and optimizing sleep," in *8th International Conference on Sensor, Mesh and Ad Hoc Comm. and Networks*. IEEE, 2011.
[20] M. Nottingham, "Web linking," 2010. [Online]. Available: http://tools.ietf.org/html/rfc5988?chocaid=397
[21] K. Dolfus and T. Braun, "An evaluation of compression schemes for wireless networks," in *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010 International Congress on*. IEEE, 2010, pp. 1183–1188.
[22] D. Caputo, L. Mainetti *et al.*, "Implementation of the exi schema on wireless sensor nodes using contiki," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 770–774.