University of Zurich[UZH]

Oleksiy Mazhelis
Martin Waldburger
Guilherme Sperb Machado
Burkhard Stiller
Pasi Tyrväinen

Extending Monitoring and Accounting Infrastructure Towards Constrained Devices in Internet-of-Things Applications

TECHNICAL REPORT – No. IFI-2013.02

April 2013

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

O. Mazhelis et al.: Extending Monitoring and Accounting Infrastructure Towards Constrained Devices in Internet-of-Things Applications
Technical Report No. IFI-2013.02, April 2013
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Extending Monitoring and Accounting Infrastructure Towards Constrained Devices in Internet-of-Things Applications

Oleksiy Mazhelis[1], Martin Waldburger[2], Guilherme Sperb Machado[2], Burkhard Stiller[2], and Pasi Tyrväinen[1]

[1] Department of Computer Science and Information Systems, University of Jyväskylä, FI-40014 Jyväskylä, Finland
[oleksiy.mazhelis|pasi.tyrvainen]@jyu.fi
[2] Department of Informatics (IFI), Communications Systems Group (CSG) University of Zürich, CH-8050 Zürich, Switzerland
[waldburger|machado|stiller]@ifi.uzh.ch

Abstract. Internet-of-Things (IoT) is envisioned to bring Internet connectivity to a vast number of sensing or actuating objects, many of which represent inexpensive devices with constrained computational and communication capabilities. For the organizations that manage these devices on an organization-individual or federated basis, the monitoring of each device's operational status and performance level, as well as the accounting of their resources usage are of great importance. Thus, monitoring and accounting support shall be built into future IoT platforms in terms of ready to use components as well as the relevant set of protocols and interfaces to access their functionality.
This paper undertakes a key step towards this goal of equipping IoT applications with monitoring and accounting capabilities by utilizing available infrastructure components and standard communication protocols. The contribution of the paper is two-fold. First, the paper studies the applicability of the Constrained Application Protocol (CoAP), a lightweight transfer protocol under development by IETF, for efficiently retrieving monitoring and accounting data from constrained devices. Second, the paper studies the feasibility of using AMAAIS, an infrastructure solution providing standard building blocks for the accounting and monitoring of IT services, in order to collect, pre-process, distribute, and persistently store this information. As a basis for this study, the necessary on-device and infrastructure components are prototypically implemented and empirically evaluated in a realistic simulation environment. Experiment results indicate that CoAP is suited for efficiently transferring monitoring and accounting data, both due to a small energy footprint and a memory-wise compact implementation being achieved. Likewise, the AMAAIS infrastructure is found suitable for pre-processing, distributing, and storing monitoring and accounting information, as it can be relatively easily adapted for a constrained environment. AMAAIS showed an additional benefit in its ability to define recipients of different subsets of the information in a flexible manner, which may be of great value in federated scenarios with organizations collectively using resources of constrained devices.

1 Introduction

Internet-of-Things (IoT) can be defined as a vision for the future of information and communications technology wherein a variety of attached or embedded real-world "things" surrounding us will be interacting and communicating with other virtual and physical entities through a global Internet infrastructure [5]. Such interconnections will enable fine-grained information about the state of physical things and/or our physical environment to be timely collected and dynamically supplied to applications over the Internet.

It is envisioned that up to 50 Billion of devices will be connected to the Internet by 2020 [2]. Many of these devices will be the so-called constrained devices, *i.e.*, devices with constraints on their memory size, computing power, communication capabilities, and/or available power [7]. In particular, a significant portion of these devices will be so-called Class-1 devices with circa 10 kByte of data space (RAM) and circa 100 kByte of code space (ROM, Flash). While being small and inexpensive, Class-1 devices are still capable to communicate with their peers through an IP (Internet Protocol) network [7, 8].

One could imagine an application scenario – such as the one envisioned by the A^4-Mesh project [47] – where a set of constrained devices are deployed as wireless sensor nodes in remote areas, and where these devices are responsible for measuring specific physical characteristics of their environment, *e.g.*, permafrost, glaciers, avalanches, landslip, radiation, soil humidity, rain, and solar radiation. While powered by batteries, these devices are supposed to run unattended for months or years, and communicate the sensor readings to external nodes in Internet.

Multiple organizations may be involved in the process of aggregating, storing, and processing the information from these devices. For instance, measurements of permafrost, radiation, and other environmental characteristics may be used by different research institutions, public organizations, and application providers – all sharing the access to resources of the same devices, but potentially accessing them at different times and with distinct access patterns. Besides, in order to attain high availability, performance, scalability, proximity, compliance with local regulations, or to achieve cost or energy efficiency, information from sensor nodes may be aggregated, stored, and processed by multiple cloud providers. In such a setting, constrained devices operate in a federated environment, with multiple organizations sharing either device resources [21, 44] or cloud resources for efficient processing [30].

Users of these constrained devices should have the possibility to monitor their operational status – including, *e.g.*, remaining battery energy, available memory, or detected errors – without the need to physically visit the deployment site. Resource usage may need to be accounted, too, both for network management purposes and, especially in case of multiple organizations requesting information from devices, for any potential charging and billing purpose. These information needs outlined necessitate the implementation of an infrastructure to support an efficient monitoring and accounting for constrained devices.

The physical constraints of wireless sensor nodes impose constraints on suitable monitoring and accounting approaches and communication protocols to be used. In particular, since monitoring and accounting involves on-device metering of a device's technical parameters and resource usage, this metering process should be conservative in memory, code space, and power expenditure, to accommodate limited capabilities and restricted battery power of the devices. This renders an application of traditional approaches – *e.g.*, those based on Simple Network Management Protocol (SNMP) [27] – suboptimal.

Hence, this paper aims at extending IoT applications, which are based on constrained devices, by monitoring and accounting functionality. Rather than introducing a new management framework and devising an additional, new communications protocol to implement it, this paper follows the approach to study the feasibility of using an available accounting and monitoring infrastructure, as well as the applicability of a standard communication protocol for constrained devices. Accordingly, two research questions are addressed in this paper:

- Is it feasible to utilize the Constrained Application Protocol (CoAP) – a transfer protocol that was introduced as a lightweight alternative to HTTP and optimized to work in constrained environments [8] – for retrieving the information needed for monitoring and accounting purposes as metered on constrained devices?
- Is it feasible to extend the functionality of a monitoring and accounting infrastructure not tailored to constrained environments – such as the infrastructure produced by the Accounting and Monitoring of Authentication and Authorization Infrastructure Services (AMAAIS) project [48] which provides components for accounting and monitoring of IT services and enables their integration with a federated authentication and authorization infrastructure – in order to allow the information from constrained devices to be utilized for monitoring and accounting purposes?

In response to these research questions, the design and implementation of a prototype has been undertaken to support the integration of monitoring and accounting capabilities in IoT applications. The functionality of AMAAIS is extended with (i) a metering component deployed on constrained devices in order to obtain relevant operational parameters and resource usage information, and (ii) a networked accounting application component. This networked accounting application is integrated with the core AMAAIS infrastructure. It communicates with the metering component using CoAP as a transfer protocol. For these purposes, the paper includes a detailed specificaiton of information to be metered on constrained devices as well as considerations on questions of architecture, protocols, and data formats to be used when transferring information from constrained devices to the core AMAAIS infrastructure. Finally, the implemented prototype is evaluated with respect to performance and discussed with respect to costs incurred due to the on-device metering process.

The reminder of this paper is organized as follows. In the next section, relevant terminology is introduced, and related work in the domain of constrained device management is overviewed. Key elements of the AMAAIS infrastructure are described in Section 3, along with determined changes needed to make it applicable to a constrained environment. Full details of the prototype implementation and its performance evaluation are reported and discussed in Section 4, followed by a discussion of costs incurred due to the acquisition of monitoring and accounting information in Section 5. Finally, in Section 6, obtained results are summarized, and directions to further work are outlined.

2 Background and related work

This section provides definitions for relevant terms, considers cloud computing platforms targeting IoT applications as well as the support for monitoring and accounting therein, and it overviews management mechanisms available for constrained devices in general and for retrieving monitoring and accounting information in particular.

2.1 Terminology

Constrained devices can be categorized into (i) Class-0 devices ($\ll$10 kByte of RAM and $\ll$100 kByte of ROM) only capable of engaging in simple communication scenarios with the help of a proxy or gateway, (ii) Class-1 devices ($\approx$10 kByte of RAM and $\approx$100 kByte of ROM) powerful enough to directly communicate with their peers in the Internet via lightweight protocols, and (iii) Class-2 devices ($\approx$50 kByte of RAM and $\approx$250 kByte of ROM) whose capabilities are sufficient to support full-fledged protocols used in conventional network nodes [7]. Among these three categories, Class-1 devices, being both inexpensive and capable of communicating with their peers in the Internet, are believed to play an important role in emerging IoT applications [8]. Class-1 devices therefore determine the main focus of this work.

The management of constrained devices requires a broad set of functionality to be implemented to efficiently configure, monitor, and control them. As these devices are managed as a part of the network to which they belong, the core operational network management functions that are conventionally grouped along the five functional areas of fault, configuration, accounting, performance and security [1], are also relevant for managing constrained devices and networks thereof [39]. These functional areas can be defined as follows:

- Fault management functions address the problem of detecting, isolating, and correcting abnormal operation of the network and its elements.
- Configuration management deals with identifying, provisioning, and controlling network elements.
- Accounting management represents the set of functions needed for gathering the information about resource usage by customers, as well as for summarizing it into accounting records, *e.g.*, for the purposes of charging and billing.
- Performance management functions are responsible for continuously gathering and analyzing statistical data describing the performance of network elements, for the purposes of monitoring and timely correcting the behavior of a network.
- Finally, security management encompasses the functions for providing confidentiality and integrity of data, authentication, authorization, access control, and non-repudiation services, as well as provisions to detect intrusions and recover from them.

Many of these functions are enabled by the so-called *metering* process whereby technical parameters of a particular resource are identified and current usage of the resource is determined. The metering process can be triggered by signaling or other external polling events, or alternatively it can be performed periodically or according to a statistical sampling scheme [24].

In the context of this paper, the focus is restrained to accounting management and performance management (specifically, performance monitoring) functions. Extensions to include other management functions are outside of the scope of this paper – they are left for further work.

2.2 Management architecture and protocols for constrained IoT devices

The physical constraints of wireless sensor nodes impose constraints on suited communication protocols and management methods. This renders the application of traditional monitoring and configuration approaches suboptimal.

In order to cope with the specifics of networked constrained devices, Ruiz et al. have introduced MANNA [39], a management architecture for Wireless Sensor Networks (WSN) based on manager-agent interactions. The authors specified relevant management functions and considered the WSN information model [38], although empirical evaluation of the proposed architecture has not been reported. Likewise, an agent-manager interaction is assumed in the WSN management system proposed by Ma et al. [31] whose WSNManagement System relies on SNMP and implements configuration, performance, and fault management functions for TinyOS devices. The system has been prototypically implemented; however, the performance of the management architecture and its overhead are not reported in the paper.

A number of studies were aimed at applying standard IP tools, such as SNMP and NETCONF [18] – a protocol used for manipulating the configuration of a network device – for device management purposes. Under the assumption that standard tools require a large Management Information Base (MIB) and result in noticeable message overhead, applying them directly is not usually seen practical. Therefore, attempts are being made at tailoring these protocols, *e.g.*, by limiting the set of functionality to be implemented, or by introducing a gateway device to mediate between standard protocols and tailor-made monitoring agents to run on constrained devices. In particular, this approach is followed by the LoWPAN Network Management Protocol (LNMP) [35], 6LoWPAN-SNMP [12], and EmNetS Management Protocol (EMP) [11]. These approaches introduce optimizations to the SNMP architecture to better match the constraints of the devices being managed.

Rather than introducing modifications to standard protocols, Kuryla and Schönwälder [27] have studied the feasibility of implementing SNMPv1 and SNMPv3 message processing models as an SNMP agent for Contiki OS. To meet the hardware constraints, only Get, GetNext and Set operations were implemented, and simplifications to the modular SNMP architecture were made. The implementation reportedly has a relatively modest memory footprint and a short processing delay of 40–120 ms; however, it does not support notifications (Trap and Inform), and thus implies the need for the manager component to explicitly request information from an on-device agent.

Sehgal et al. [43] have empirically compared SNMP against NETCONF in the context of constrained devices. A lightweight version of NETCONF, without subtree filtering and the edit-config operation, has been implemented by the authors. As NETCONF relies on the exchange of relatively large XML messages, its use in constrained devices results in longer processing time, as well as in an increased memory footprint [43].

For event notification and logging, also the SYSLOG protocol [20] can be used [42]. It can run on UDP [36] incurring neither significant data communications overhead nor large processing delay. The protocol, however, assumes that the configuration of a syslog application is performed through other channels, and after that the syslog application is autonomously generating and transmitting the information needed. It should also be noted that this is a pure simplex protocol and as such it does not provide acknowledgment of message delivery.

Instead of tailoring traditional protocols designed for non-constrained devices, an alternative approach is to re-use CoAP for device management purposes. CoAP is a new transfer protocol introduced as a lightweight alternative to HTTP. It is optimized to work in constrained environments [8, 45]. CoAP relies on UDP as a transport, and offers a simple in-built stop-and-wait reliability mechanism. It uses a compact four-byte binary header with a total header size of 10 to 20 Byte, and defines four methods – GET, POST, PUT, and DELETE – enabling a RESTful architectural style. Importantly, the protocol supports an asynchronous retrieval of information by using the "Observe" option: by issuing specially crafted GET messages, observing clients subscribe to the updates of a particular resource of interest; after that, a constrained device asynchronously notifies its observers about resource changes, without the need for explicit polling requests. Finally, CoAP's similarity to HTTP simplifies the implementation of CoAP-to-HTTP proxies for interworking with HTTP-based systems. All that makes messages compact, minimizes the overall volume of the transferred data, and reduces the complexity of implementation – all crucial characteristics in constrained environments.

In comparison to the protocols above, the use of CoAP for obtaining metering information from constrained devices offers numerous benefits, such as the possibility of re-using the CoAP protocol stack implementation without any need for implementing other management-specific protocols. The avoidance of any TCP session establishment and related overheads reduces the application-level message size, and adding security to a CoAP-based implementation by using DTLS will have the same added footprint as in the NETCONF/DTLS implementation reported in [43].

To the best knowledge of the authors, the use of CoAP for device management purposes has not been tried in practice yet. Sehgal et al. [43] suggest the suitability of CoAP for accessing on-device configuration data, while also indicating the lack of practical experience.

Another alternative for transferring the metering information is the Compressed IPFIX protocol[3] [40] which compresses the data and the template records of the IP Flow Information Export (IPFIX, [13]) protocol. When compared with CoAP, Compressed IPFIX offers similar benefits in terms of a compact header and the possibility to push information updates without explicitly requesting them, but lacks the flexibility of the former manifested in the support for RESTful interactions, as well as its in-built reliability mechanism.

[3] `http://tools.ietf.org/html/draft-braun-core-compressed-ipfix-03`

For these considerations, the applicability of CoAP for the purpose of retrieving metering information from constrained devices is investigated in this paper. The motivation for focusing on CoAP as a transport protocol is two-fold. First, this protocol is used as it is originally aimed at constrained devices. Though the management of constrained devices was not among the set of use cases considered during protocol specifications, the specific protocol characteristics (as outlined previously) indicate that CoAP may be successfully used for implementing (a subset of) device management functions. Second, as CoAP becomes available on constrained devices for application purposes, re-using the protocol for monitoring and accounting purposes instead of using a dedicated protocol allows the code footprint on the device to be decreased, which is of prime importance for constrained Class-1 devices.

2.3 Metering on constrained devices

The constrained nature of Class-1 devices also imposes constraints on the metering process. Since metering consumes the scarce energy resource on devices, it should be computationally lightweight and should avoid the transmission of large messages. These and other requirements for managing networks with constrained devices are being defined by the Constrained Management (COMAN) – a recently established IETF activity [19] – and are summarized below.

Requirements for managing networks with constrained devices. On a general level of a management architecture/system, the set of requirements relevant to monitoring and accounting functions include the following needs: (i) to minimize the state maintained on constrained devices, (ii) to support devices that are not always on-line, (iii) to support lossy and unreliable links, through in-built resilience mechanisms and through a limited data rate, (iv) to keep the encoding of management data compact, and (v) to optionally compress management data or complete messages. At the implementation-level, with the aim of minimizing communication overhead, two mandatory requirements are stated:

- Avoiding complex application layer transactions with large messages, since they require large memory buffers and increase the volume of information re-transmitted in case of transaction failure.
- Avoiding the fragmentation and reassembly of messages at multiple protocol stack layers, *e.g.*, by limiting the size of application layer messages.

Table 1 lists requirements with respect to the monitoring functionality on Class-1 devices, as defined in [19], and indicates whether these requirements are considered mandatory or optional. As can be seen from the table, only the monitoring of device status and energy level are considered mandatory, whereas all other requirements are optional to implement depending on the device type and depending on the respective management needs.

For the energy management use cases, additional requirements are stated on the management of energy resources, *e.g.*, by adjusting the sampling rate and by reducing transmission power. Finally, the document mandates a number of security functions, including the need for authentication and access control (both on constrained devices and in the management system) as well as the need for a security bootstrapping mechanism.

Static vs. dynamic management information. Management information can be categorized as static or dynamic [39]. *Static* information describes the configuration of constrained device as well as the service based on them. This information may encompass the device identifier, type, its sensing capabilities in terms of types of available sensors and their accuracy, available connectivity alternatives, and their properties, etc. Static information is changed infrequently, and therefore can be stored or cached outside of a constrained device and/or retrieved out-of-band.

Dealing with such static information is facilitated by utilizing standard languages for describing sensors, their capabilities, and measurements. For instance, the Sensor Web Enablement (SWE) standards by the Open Geospatial Consortium (OGC) provide standard models and XML schemas for describing sensors, as well as observations and measurements from them [9, 15]. In addition, ontologies and reasoning mechanisms for semantic interoperability and integration of heterogeneous sensor networks are available; Ref. [29] provides a good overview of the ontologies produced for (semantic) sensor networks, and introduces the Semantic Sensor Network (SSN) Ontology as the standard ontology promoted by the W3C.

On the other hand, the management information gathered for monitoring and accounting purposes represents *dynamic* information, *i.e.*, information about the current status of a constrained node, and may include the sensing and communication coverage map, residual energy in the constrained device, or usage

Table 1. Requirements on monitoring functionality of networks with constrained Class-1 devices [19]

Requirement title	Description	Priority
Device status monitoring	Collect and expose information about device status.	Mandatory
Energy status monitoring	Collect and expose information about device energy parameters, such as battery level or communication power.	Mandatory (for energy reporting devices)
Monitoring of current and estimated device availability	Collect and expose information about estimation of remaining available resources including, *e.g.*, energy, memory, computing power, and queue buffers	Optional
Network status monitoring	Collect and expose information on the status of connected network segments.	Optional
Network topology discovery	Collect and expose information about the network topology by using a network topology discovery capability.	Optional
Self-monitoring	Provide local fault detection capability.	Optional for Class-1 devices
Neighbor-monitoring	Provide the capability of fault detection in local network.	Optional
Notifications	Provide the capability of sending notifications on critical events and faults.	Optional
Performance Monitoring	Collect and expose information about the performance of a device.	Optional
Fault detection monitoring	Collect information about network state and use it for providing fault detection monitoring.	Optional

statistics [39]. Dynamic information changes frequently or continuously, and, thus, needs to be acquired through the metering process triggered by signaling or other external polling events, or alternatively needs to be performed periodically or according to a statistical sampling scheme [24].

While static properties of constrained devices, their capabilities and produced measurements are specified in various standards such as the mentioned OGC SWE standards and W3C SSN Ontology, dynamic information appears to be standardized to a lesser degree. Some of the dynamic properties of constrained devices are however specified or exemplified in [3, 6, 23].

RFC 3433 [6] defines an extension to the Entity MIB (RFC 2737) wherein the information about physical sensors is defined. Among other properties, the document defines several objects that are relevant for accounting and monitoring purposes, namely:

- entPhySensorOperStatus – the current operational status of the sensor.
- entPhySensorValueTimeStamp – the time when the information in an entry was updated.
- entPhySensorValueUpdateRate – the time elapsing between polling for information updates.

The hierarchical management information model introduced by the IoT-BUTLER project [3] classifies constrained device parameters into families of general, network-related, system-related, device/vendor-specific parameters, and the set of operations allowed by a device. Out of these, dynamic properties are reflected in the families of system- and network-related parameters, including energy level, CPU and memory usage, sampling rate, neighbor table, alarm threshold, received signal strength, packets sent, and uptime.

Finally, the specifications of the Sensor Markup Language (SenML) define a data model and media types for transmitting sensor information on top of a transfer protocol (*e.g.*, HTTP or CoAP) [23]. This specification targets devices with constrained capabilities and therefore limits the amount of metadata to be included in the messages sent. Besides dynamic sensor measurement information, the standard specifies certain dynamic management information, such as remaining battery energy level and update time, plus it allows further dynamic management properties to be defined.

To make the encoding of management data compact, in SenML, the measurements and metadata are packaged into a single object that encapsulates an array of entries. The specifications define three serialization alternatives for the data model: JavaScript Object Notation (JSON), XML, and Efficient XML Interchange (EXI). Among these, EXI allows the messages to be compressed the most efficiently, often allowing the total message size to be under 80 Byte, at the cost of a slightly more complex implementa-

tion. Alternatively, it is possible to achieve a similar degree of compactness by encoding the data using comma-separated values (CSV) format, the negative side here being a lack of metadata in the message.

Relevance to this work. The focus in this paper is on the acquisition of dynamic information for monitoring and accounting purposes, and an assumption is taken that static management information can be obtained out of band by other means. Specifically, this paper aims at equipping IoT applications with the mandatory monitoring functionality set in [19], including device status monitoring and energy status monitoring, while also satisfying the respective other architectural- and implementation-level requirements. Meanwhile, the implementation of security mechanisms has been left outside of the scope of this work due to the fact that the CoAP DTLS security specifications are still under work at the time of writing. The prototype provides, in addition, the optional capability to inform about remaining resources (in particular, dynamic memory) of a device, whereas implementing the respective other optional capabilities has been left for further work.

2.4 Leveraging cloud computing services in IoT applications

Due to limited capabilities of constrained devices, it is often reasonable to implement the aggregation, filtering, storage, correlation, and further processing of acquired information remotely, outside of constrained devices. Cloud infrastructure can be used to complement the limited resources of constrained devices with practically unlimited computing and storage capacity which is shared among multiple users, provisioned to them on-demand, and billed on a per-usage basis [33]. Practically unlimited capacity and other benefits of cloud computing, including reduced capital expenditures, rapid deployment, global availability, and on-demand scalability make the use of cloud computing for IoT applications a justifiable option [4, 46].

Both cloud computing infrastructure resources and constrained device resources can be shared among multiple organizations. In particular, multiple organizations may utilize information from constrained devices in different applications, while sharing constrained devices and potentially collectively managing them [22] as a part of a federated sensor network [21]. Also, even within a single application, for the purposes of achieving high availability, performance, coping with legal domains, or meeting other requirements, information from a constrained device may be aggregated, stored, and/or processed in multiple clouds working in concert as a cloud federation [30]. Thus, constrained devices may need to operate in a federation which can be defined as a set of agreements, standards, and technologies used by multiple organizations for collectively operating a shared resource – the resources of constrained devices and/or cloud resources to efficiently process the information therefrom [28, 32].

The adoption of cloud infrastructure has also implications to monitoring and accounting functions. On one hand, in applications utilizing cloud infrastructure, operational state and data regarding usage of cloud infrastructure resources possibly need to be monitored and accounted for. Moreover, whenever a device or cloud resource is shared by multiple organizations, the monitoring and accounting infrastructure in place has to support such a federated environment. On the other hand, cloud infrastructure resources may be utilized for hosting components of the monitoring and accounting infrastructure.

A number of IoT cloud platforms are available for storing and processing information originating from constrained devices. Table 2 lists major platforms and summarizes their support for monitoring and accounting functionality. As the table indicates, these platforms usually monitor and account for the intensity of platform usage, measured mainly by means of the API (Application Programming Interface) request rate. In some platforms, also the volume of stored data (SensorCloud, iDigi) and the number of devices (OnePlatform, iDigi) are accounted for. Meanwhile, the support for monitoring and accounting of constrained device resources seems to be practically absent; among the platforms listed, only iDigi provides an in-built API for requesting the status of constrained devices.[4] Therefore, retrieval of information about relevant parameters and usage data of constrained devices has to be implemented as a part of the software application to run on top of a platform – *e.g.*, by making constrained devices transmit such information as a part of a data feed. To the best knowledge of the authors, no support is provided by these platform for a federated environment either.

Thus, in summary, contemporary cloud IoT platforms provide little or no support for retrieving monitoring and accounting information from constrained devices. Therefore, there is an identified gap for a platform that offers such functionality for applications involving constrained devices, that provides (cloud) infrastructure to deploy this functionality, and that supports federated environments.

[4] Note that, while monitoring is often advertised by a platform vendor, the subjects of monitoring determine typically external parameters, such as structural health monitoring or environmental monitoring, not parameters of constrained nodes themselves.

Table 2. Accounting and monitoring provisions in IoT Platform-as-a-Service (PaaS) offerings

Platform	Website	Support for monitoring and accounting
Thingspeak	`https://www.thingspeak.com/`	Rate of updates per channel is monitored for enforcing the maximum allowed limit (5 updates per minute).
Nimbits	`http://www.nimbits.com/`	API calls are accounted for billing purposes; a free quota of 1000 API calls a day.
Evrythng	`http://www.evrythng.com/`	Requests are accounted for enforcing the free quota (5000 requests a day).
Cosm	`http://cosm.com`	Requests are accounted for enforcing the free quota (100 requests per minute).
NanoService	`http://www.sensinode.com/EN/products/nanoservice.html`	No information at the website.
OnePlatform	`http://exosite.com/products/onep`	Number of devices and user accounts is accounted for billing purposes.
SensorCloud	`http://www.sensorcloud.com/`	Data points and API requests are accounted for billing purposes.
Thingworx	`http://www.thingworx.com/`	No information at the website.
Arkessa	`http://www.arkessa.com`	No information at the website.
iDigi	`http://www.idigi.com/`	Number of devices, transactions, data streams (storage in GB) are accounted. Device status (*i.e.*, inactive vs. active) is monitored.

3 Adapting AMAAIS infrastructure to constrained environment

The AMAAIS project aims at supporting the accounting and monitoring of IT services, by offering a set of enabling components and facilitating their integration with an authentication and authorization infrastructure [48]. The infrastructure produced by AMAAIS is well suited for federatively aggregating, processing, and storing accounting and monitoring information. This infrastructure, however, has not been designed for constrained devices, and hence needs to be tailored to be suitable in constrained environments. This section overviews the AMAAIS infrastructure and considers the changes that are needed for making it work in applications with Class-1 devices.

AMAAIS provides federated (cross-university) accounting and monitoring functionality and integrates with a Shibboleth-based Authentication and Authorization Infrastructure (AAI). It includes functionality for acquiring relevant information by parsing log files (or databases) for services offered by a federation member, for creating accounting and monitoring events, as well as for delivering these events to a predefined set of recipients for further processing or persistent storage. An event is a measurable real-life event represented as a set of attributes, *e.g.*, metered battery voltage, available memory, or uptime.

The interplay of AMAAIS accounting applications and AMAAIS core components is visualized in Figure 1. In the AMAAIS core software package, the Accounting Client (AC) and Accounting Server (AS) contain a set of so-called Sources and Sinks that constitute a pipeline (not depicted in Figure 1) in which events flow through: A pipeline is composed of a set of steps (each step being either a Source or a Sink). Sources and Sinks are software components that have a common interface in order to be easily connected to each other. Sinks are used to receive and process events, and Sources are components that produce events.

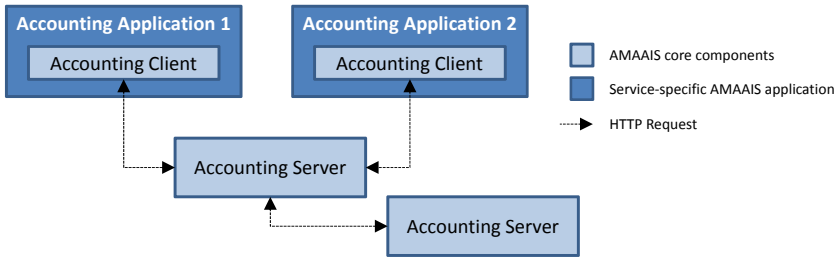


Fig. 1. AMAAIS Components and Applications

AMAAIS provides purpose-specific sub-types of Sinks and Sources. A Sink can be a Stage or a Channel. On the other hand, a Source can be just a simple Source or a SourceStage. A Stage is a component that assumes the role to perform operations on an event. For example, a Stage can modify an accounted event in a particular manner. A Channel is a component responsible for routing accounted events it receives to a Sink – which usually is a Stage. The difference between a Stage and a SourceStage is merely conceptual: a SourceStage is a Stage that also produces events and, therefore, can be considered as a Sink and a Source (it implements both interfaces).

In the underlying implementation, AMAAIS utilizes the Shibboleth implementation framework [34] which simplifies its use in a federated environment and allows the functionality of the AC to be integrated with the functionality of an Identity Provider (IdP) and a Service Provider (SP) as defined in the Shibboleth project. In line with the principles of Shibboleth, and in order to enable interoperability, communication between AC and AS is based on exchanging Security Assertion Markup Language (SAML) messages.

Based on the outlined concept of Sources and Sinks, AC pipelines as well as AS pipelines can be constructed by combining the respective needed Source and Sink components in any wished for order. Within the AMAAIS core, the following are the main Source and Sink components implemented: BroadcastChannel, QueueChannel, FilterAttributesStage, PersistEventsToDBStage, ResolveAttributesStage, TransmitSAMLStage, and LookupRelyingPartyMetadataStage. Based on these components, the following functionalities are enabled:

- Event broadcasting: Accounted events can be forwarded to one or more AS. The BroadcastChannel component can be used to clone events and forward them to different pipelines.
- Queuing: Accounted events can be queued until a release rule is met. A release rule can be a time period for which events remain in a queue, or a maximum queue size.
- Filtering: Accounted events can be filtered either on an AC or AS. For example, for certain event types, anonymization may be needed in order to delete sensitive event data. Event attribute resolving: Each event has one or more attributes that should be associated with a proper attribute encoder. For example, for a certain event type, attributes may have to be encoded using UTF-8.
- Metadata: Information about AC and AS within a Shibboleth-based federation are represented in a metadata file. If a component needs any specific information about other entities within a federation (*e.g.*, public certificates of a given AS), LookupRelyingPartyMetadataStage is able to retrieve the respective information from a metadata file.
- Event persistence in database: Accounted events should be properly persisted, possibly using different databases (*e.g.*, MySQL, PostgreSQL). Therefore, PersistEventsToDBStage has the knowledge of how to generate a database schema for specific event types.
- SAML-based protocol: The TransmitSAMLStage component implements a SAML-based protocol that marshals/unmarshals events to/from a SAML message. This protocol determines a key asset to enable interoperability between federations that use AMAAIS.

Pipelines can be configured through an XML file. This XML file may be modified at any time, since AMAAIS has a configuration reload capability. The configuration reloading process first detects that a configuration file was modified, validates it, and then reconfigures the pipeline at execution time – without the need of, *e.g.*, restarting the AC or AS. This enables a higher availability and lower chances of event losses.

In turn, any service-specific Accounting Application (AA) is responsible for generating events, *e.g.*, by parsing system log files and extracting any information of interest from them. Once a new event is created, the AC's API is called. Both AA and AC are running on the same host as a daemon. Once an event generated by an AA is pushed to the AC, it enters the AC pipeline. The event will pass along the AC's pipeline, will be then transmitted to one or multiple AS, pass along the respective AS pipeline(s), and will be persisted to database eventually.

The main advantage of a highly configurable pipeline is the immense flexibility this may mean to each service scenario. Moreover, having such flexibility at disposal will show its benefits in federated environments, where different services are managed by different organizations, but with a need for a joint, inter-operable monitoring and accounting system.

In principle, it is possible to enable monitoring and accounting functionality on constrained devices by deploying AA and AC components directly on the device. However, for Class-1 devices (which determine the main focus in this paper), this is not feasible due to memory constraints and the lack of Java support, not to mention a large computational and data communication overhead while being engaged in an exchange of SAML messages.

Hence, in order to cope with the limitations of constrained devices, it is reasonable to implement the AA and AC outside of any constrained device, for instance, on a gateway between constrained and unconstrained networks, or on another unconstrained node, as shown in Figure 2. This way, the functionality to be kept on constrained device can be limited to a lightweight metering agent – a *meter* – responsible for executing the metering process and transmitting the metered information to the AA executed on a remote host by using a suitable communication protocol (thus, called Networked AA in Figure 2).

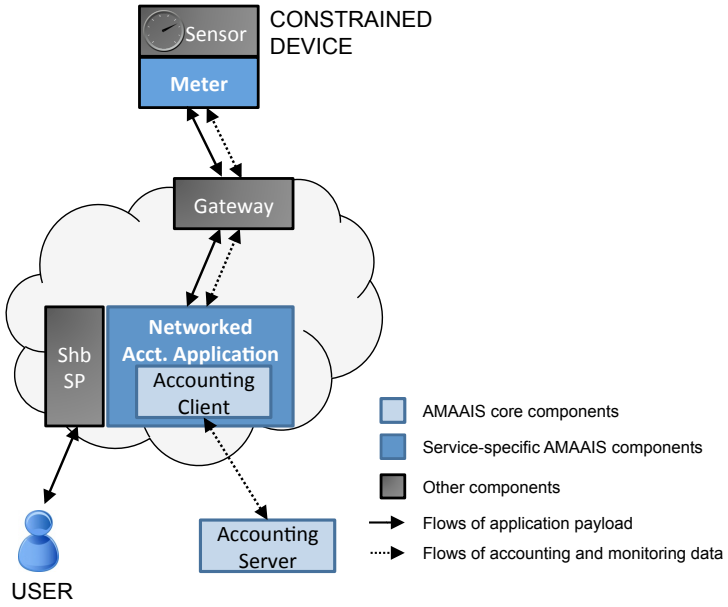


Fig. 2. Adapting the AMAAIS components to work in constrained environment

CoAP is used as a data transfer protocol, due to its compact header and the use of UDP as transport, both allowing data communication overheads, and hence the related energy consumption, to be minimized. Another important feature of CoAP is its subscription mode (triggered by the "Observe" option in a GET request) which allows a device to notify the set of subscribers about events of interests without the need to request them explicitly. This effectively replaces the pulling mode of HTTP with an asynchronous push mode, allowing further reduction of data transfer volume. Last but not least, the CoAP protocol implementation can be reused for transmitting both application payload and metered management data, thereby avoiding the need for a dedicated management-specific protocol and hence reducing the memory footprint of the code on a device.

The AA and AC components can be implemented either on a gateway node that connects constrained devices with the unconstrained network, or on another remote host outside of the constrained network. In the latter case, depending on the capabilities of a remote host running the AA and AC components, a gateway may simply forward CoAP messages, or it may alternatively act as a proxy converting CoAP messages to HTTP.

The meter on a constrained device can collect and dispatch accounting- and monitoring-related information periodically, upon request, or whenever a certain condition is met, *e.g.*, whenever battery level drops below 30%. The meter implementation depends on the type of constrained device, the application service it contributes to, and the type of required monitoring and accounting information. As will be discussed later, the delivery of metered information may be done separately from or along with application payload. Therefore, the meter is seen as a service-specific component in the figure.

Based on the synthesis of related work overviewed in Section 2.3, the following information attributes may be metered for monitoring and accounting purposes (the list should by no means be seen as complete):

- Current operational status of the sensor [6, 23]
- System uptime [3]
- Time between information updates [3, 6]
- Current energy level [3, 23]

– Size of available memory [3, 23]
– Packets sent and received, packets dropped, etc. [3]

Using JSON representation with the syntax defined by SenML [23], a message with metered information can be encoded as shown in Listing 1.1. The listing exemplifies a message originating from a node with base name "http://[2001:db8::2]/" created at base time 213220 (node uptime) and with message id 35, which informs that the current energy level of the device is at 97%, that its operational status is normal, that 2048 Byte of dynamic memory are still available, that 228 packets were transmitted, 36 packets received, and that 28 packets were dropped.

```
{"e":[
      { "n": "eventId", "v": 35},
      { "v": 97.0, "u": "%EL"},
      { "n": "status", "sv": "OK" },
      { "n": "memory", "v": 2048, "u": "Byte" },
      { "n": "tx", "v": 228, "u": "Packet" },
      { "n": "rx", "v": 36, "u": "Packet" },
      { "n": "drop", "v": 28, "u": "Packet" },
      { "n": "temperature", "v": 27.2, "u": "Cel" },
      { "n": "light", "v": 80, "u": "lx" }
 ],
 "bn": "http://[2001:db8::2]/",
 "bt": 213220
}
```

Listing 1.1. Example of a message with metered data in JSON representation

Note that this example message also contains the application-specific payload, namely, readings from temperature and light sensors. Integrating the application payload with the accounting and monitoring information within a single message allows data communications overhead to be minimized and hence battery lifetime at a device to be prolonged; this will be discussed later in the paper in further detail.

The message in the above JSON representation requires 318 Bytes. Thus, if this representation is used to encode messages, message size will be significantly longer than the 80 Byte limit suggested by Jennings et al. [23] in order to avoid packet fragmentation and reassembly. This is important, since the "Observe" option of CoAP does not assume block transfer when delivering notifications, thus, making it cumbersome to implement the delivery of long notification messages spanning multiple packets.

A more compact message size can be achieved by using EXI representation which relies on principles from information and formal language theories for compressing an XML stream into a binary format [41]. In schema-informed mode, the use of EXI encoding allows messages to be compressed to as little as 3% of the original XML message [45]. Important for constrained devices, EXI encoding is computationally lightweight, too, and can be efficiently implemented on constrained devices [10]. If only encoding needs to be performed at a constrained device, one can further simplify the implementation by hardcoding a binary EXI message and replacing the bytes corresponding to the attributes in question with updated information, as exemplified in [23].

4 Empirical evaluation

In order to empirically evaluate the feasibility of providing IoT applications with monitoring and accounting functionality by using an available lightweight protocol (CoAP) and an available accounting and monitoring infrastructure (AMAAIS), a prototype has been implemented as an extension to the AMAAIS core infrastructure, utilizing CoAP for communication with an on-device meter component. Details of this prototype implementation as well as results of its empirical evaluation are described in this section.

4.1 Prototype implementation details

For prototyping purposes, the existing AMAAIS components have been extended by a networked AA and an on-device meter, whose implementation details are provided below.

The meter software agent has been implemented for TmoteSky motes, a sensor mote platform featuring an MSP430 16-bit 3.9 MHz CPU, a CC2420 radio chip, as well as 48 kByte of program flash and 10 kByte of RAM. A ready set of software components, including Contiki OS and networking libraries were

utilized in the implementation. For evaluation purposes, the meter software has been deployed in the Cooja simulation environment, which emulates TmoteSky motes and enables a simultaneous simulation at network level, operating system level, and machine code instruction set level [37].

As the CoAP engine for Contiki OS, the Erbium CoAP implementation [25] has been used. The meter agent is responsible for gathering and transmitting the following attributes: event identifier, remaining battery level, operational status of the device, available memory, as well as the name of the device and the uptime at which the metering process has been performed. The number of packets sent, received, or dropped has not been metered in the prototype, as these attributes are considered optional for constrained devices.

Along with the management attributes listed above, also temperature and light sensor readings are transferred. In order to simplify the implementation, a simple CSV format is used for message encoding. While EXI representation may offer a more compact encoding, CSV encoding was also found sufficiently compact, allowing the message size to be below 80 Byte.

The metering process was implemented to be executed periodically, with a configurable *metering interval*. Both subscribing to periodic metering updates and setting of the metering interval are done in a RESTful manner by issuing GET (with the "Observer" option set) and PUT requests, respectively.

In the prototype, metered data is transmitted by the meter as soon as it is acquired. This minimizes the size of memory needed for temporally storing data, and allows the transmitted message to remain compact, thus, avoiding packet fragmentation. Alternatively, in applications where the volume of metered data is low and where delays in acquiring this data can be tolerated, several measurements can be buffered first and transmitted at a later point in time, *e.g.*, when data volume approaches 80 Byte. This alternative, however, has not been explored further in this study.

In order to reduce energy consumption by the transmitter, the meter uses the ContikiMAC radio duty cycling (RDC) protocol [16]. Two versions of the meter were implemented: one with radio (or more specifically, RDC) enabled all the time, and the other with radio being periodically disabled and switched on for a relatively short duration of time (10 s) to acquire and transmit metered information, as well as to re-configure a constrained device if necessary.

The former version of the meter is aimed at application scenarios where a device needs to be constantly available, for retrieving metered information, for device configuration, or for actuating events in its environment or a system under control. For instance, a weather monitoring station may need to be ready to deliver instant measurements of wind strength and direction whenever requested. The latter version is suitable for scenarios where energy efficiency is of prime concern, and where periods of unavailability can be tolerated. For example, a device responsible for periodically delivering permafrost measurements could utilize this version of the meter, thereby allowing it to be put to sleep in between communication sessions in order to minimize energy consumption.

The gateway has been made responsible for reasons of simplicity solely for routing packets between meter and networked AA (NAA). Similarly to the meter component, the gateway has been implemented on a TmoteSky platform, and emulated within the Cooja environment, whereas the NAA and the other AMAAIS components are deployed on separate nodes outside of the constrained network.

The NAA has been implemented in Java and integrated with the AMAAIS core components[5]. For communication with constrained devices, the NAA employs the Californium Java framework [26] that implements CoAP communication primitives. With the help of the Spring framework [50], the NAA is populated with information about constrained devices to communicate with, as well as with other operational parameters, such as the name of the metering resource on the device, the length of a metering interval, and the configuration of the event processing pipeline.

Once launched, the NAA retrieves the list of constrained devices and the metering interval as well as the parameters of the event processing pipeline from the configuration file. After that, it connects to the constrained devices it is in charge of, sets the metering interval, and sends GET requests with the "Observe" option to subscribe to metering information updates. This effectively initiates a periodic retrieval of metering information from the devices.

From that point on, the NAA periodically receives messages with metered information, parses them, extracts relevant attributes, and forms accounting events which are then dispatched, through the AC API, to a pre-defined processing pipeline. For prototypical purposes, a simple pipeline was configured consisting of a filtering Stage and two persistence Stages as well as the respective communication Channels between them. The filtering Stage allows individual attributes to be excluded from further processing; for simplicity, the filter in the prototype was configured to let all the event attributes pass. In turn, the persistence Stages are responsible for storing newly created events in two separate databases. In order

[5] Available at `http://www.csg.uzh.ch/research/amaais.html`

to support an efficient persistence of application-specific event attributes, two application-specific classes have been implemented, by extending and implementing the BaseEventDAOFactory and BaseEventDAO interfaces of AMAAIS.

Only a periodically executed metering process has been implemented in the prototype. Implementing the other forms of triggering the metering process, such as a change in environmental characteristics or a critical decline of energy level, have been left for further work.

4.2 Performance evaluation

The implemented prototype has been evaluated in order to assess the feasibility and performance of the proposed approach to support monitoring and accounting capabilities in IoT applications. Both qualitative and quantitative criteria were used in the assessment. First, the prototype's functionality and properties were qualitatively matched against the requirements for the metering process in constrained environment, as discussed in Section 2.3. After that, some properties – in particular, resource consumption at a constrained device – were assessed quantitatively. Finally, the efforts needed for extending, integrating, and configuring AMAAIS core components were considered.

Meeting the requirements for metering in constrained environments. As was discussed in Section 2.3, in applications dealing with constrained devices, the management functionality shall acquire and expose at the very minimum information about device status and about device energy parameters. This functionality should require a minimal state to be maintained on the devices, support lossy and unreliable links, and keep the encoding of management data compact. Further, meeting these requirements shall be facilitated by keeping application layer transactions small and simple, and by avoiding fragmentation and reassembly of messages. Finally, both constrained devices and the management system should be authenticated, and an access control as well as a security bootstrapping mechanism shall be provided.

These requirements have been taken into account when designing and implementing the prototype. In particular, the prototype enables acquiring and delivering mandatory information, *i.e.*, device status and energy level, as well as some of the optional information (namely, available memory). By means of resilience mechanisms built into CoAP, support for lossy and unreliable network connection is provided. The use of CSV as data encoding format allows the encoding to be compact and hence helps avoid any need for fragmentation and message reassembly. A compact representation, along with the RESTful methods of CoAP, also enable simple and small application level transactions.

It shall be noted that security mechanisms were omitted when implementing the prototype. Implementing them has been postponed until the CoAP DTLS security specifications are finalized.

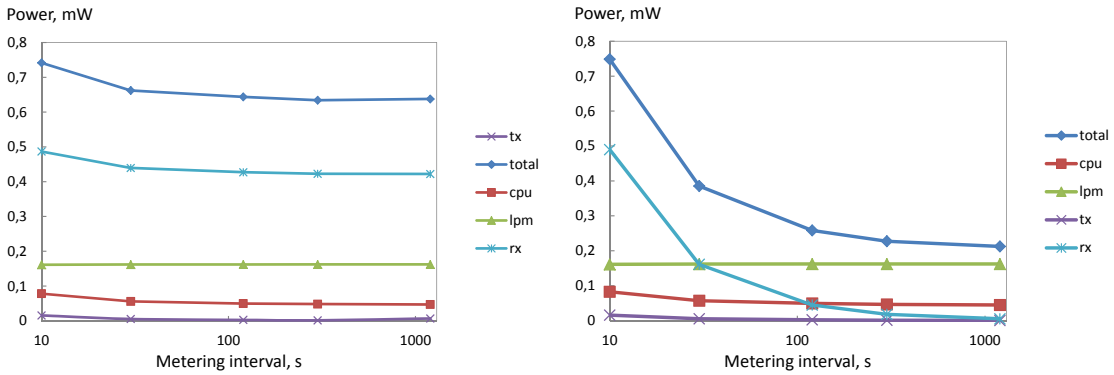
Resource consumption on constrained devices. The implementation of the meter is quite compact and requires only 1286 Byte of ROM and 158 Byte of RAM. Together with the operating system and other necessary libraries (RPL, uIP, ContikiMAC, Erbium, etc.), the overall application memory footprint is 48638 Byte of ROM and 8732 Byte of RAM. Thus, the metering application fits into TmoteSky, even though this particular platform could be seen as a lower end of the Class-1 devices.

The energy and power consumption of the on-device metering process depends on the metering interval. Therefore, when estimating the energy footprint, five different metering intervals were used: 10 s, 30 s, 120 s, 300 s, and 1200 s. For each value of the metering interval, the operation of the meter was simulated over a duration of 50 metering intervals. Power consumption during each interval has been measured, and medians of obtained measurements have been calculated.

Both versions of the meter software – with RDC always or periodically enabled – were experimented with. Power consumption at a constrained device has been estimated with the help of the Energest module built into Contiki OS [17], which traces the time a constrained device spends in different modes of operation. Using Energest traces, power consumption for both versions of meter implementations have been estimated, as visualized in Fig. 3.

As can be seen from the figure, power consumption of both versions of the meter declines as the metering interval grows. This reflects the lower amount of data that needs to be transmitted per unit of time, making the transmission-related power consumption to decline ($\tau \to \infty \Rightarrow P_{\text{tx}} \to 0$, where τ is the length of the metering interval).

With the version of the meter with the RDC always on, the radio receiver periodically awakes even when nothing is transmitted, by default with a frequency of 8Hz, to check for radio activity. This is valuable when the device needs to be always available for metering and/or reconfiguration. However, as a result, even for longer metering intervals, the incurred reception-related energy footprint remains



(a) RDC always on

(b) RDC on periodically

Fig. 3. Power consumption at a constrained device, both total and according to modes of operation including CPU active (cpu), low power mode (lpm), transmission (tx), and reception (rx): (a) RDC always on; (b) RDC on periodically. Note that a logarithmic scale is used for the metering interval shown on the x-axis

significant ($\tau \to \infty \Rightarrow P_{\text{rx}} \to \text{const} > 0$), becoming the greatest contributor to the energy footprint, as visible in Fig. 3(a).

On the other hand, for the second version of the meter, which disables the radio completely in between metering and communications sessions, the reception-related energy drainage is minimized ($\tau \to \infty \Rightarrow P_{\text{rx}} \to 0$, cf. Fig. 3(b)). Thus, power consumption at a device can be decreased by increasing the metering interval, at the expense of the device being unavailable between communication sessions.

Assuming that a constrained device is powered by a pair of AA Zync-carbon batteries, the expected battery lifetime has been estimated and is reported in Table 3. As the table indicates, even with short metering intervals of 10 s, battery lifetime is approaching half a year. With RDC always enabled, battery lifetime increases insignificantly with the metering interval, extending the battery lifetime maximum by less than a month. On the other hand, when RDC is enabled periodically, battery lifetime can be extended by up to 1.5 years and above (with metering intervals of 300 s and more), thus, approaching the self-discharge time of a battery.

Table 3. Constrained device's power consumption and battery lifetime depending on the metering interval

Radio mode	Interval, s				
	10	30	120	300	1200
Power consumption, mW					
RDC is always enabled	0,74	0,66	0,64	0,63	0,64
RDC is on periodically	0,75	0,39	0,26	0,23	0,21
Estimated battery lifetime, days					
RDC is always enabled	169	189	194	197	196
RDC is on periodically	167	324	484	550	588

It shall be noted that the RDC implementation on constrained devices consumes memory: in case more functionality needs to be put on equally constrained Class-1 devices (10 kByte of RAM, 48 kByte of ROM), less efficient RDC or nullRDC may be used, at the cost of shorter battery lifetime. It shall be also noted that energy consumption at a constrained device is affected by multiple factors, including the overhead of the routing protocol, the type of low power mode used, the time needed for sensors to initialize after being turned on again, etc. Therefore, the absolute numbers of energy consumption estimates reported above shall be used with care, as they are likely to differ to a certain extent depending on case-specific implementation details.

Efforts for extending and integrating with AMAAIS core components. The activities needed for extending, integrating, and configuring the AMAAIS core components to operate with constrained devices included (i) the implementation of the NAA, (ii) the implementation of persistence interfaces, and (iii) the provision of the NAA with on-device meter parameters and the configuration of the event processing pipeline.

The difficulty of implementing the NAA depends on the set of monitoring and accounting capabilities to be supported. The prototype application has been relatively small and simple, making the NAA implementation relatively easy too, requiring some days of efforts. Due to a limited number of event attributes, also implementing the persistence interfaces required only some hours of work. Likewise, the integration and configuration relied on using existing configuration templates, making the process straightforward.

It should be noted that AMAAIS is still under development, and therefore, the available documentation is incomplete at the time of writing. However, in the process of implementing the prototype, the deficiency of documentation was compensated by a close interaction with the research group implementing the core components of AMAAIS.

The flexibility of AMAAIS pipeline configurations makes it suitable especially in a federated environment, where the metered information or subsets of it shall be received by different organizations. For instance, by adjusting a pipeline, subscriber organizations can be configured to receive measurements from sensors on constrained devices, whereas the organization managing the constrained devices can be configured to receive management (monitoring and accounting) information, too. Both types of information would pass through the AMAAIS core which would be responsible, among other activities, for extracting the necessary subsets of attributes and for forwarding related events to the respective recipient organizations. In such an environment, the Stages implemented by AMAAIS play a role similar to telecommunications mediation software aggregating and pre-processing metered network- or service-related data, *e.g.*, for billing or performance assurance purposes [49].

5 Metering-related overheads and incurred costs

The acquisition of dynamic management information for monitoring and accounting purposes incurs additional costs to organizations deploying and operating applications based on constrained devices. In this section, costs incurred due to the introduction of monitoring and accounting capabilities to constrained devices are considered. Other costs that may be incurred on the infrastructure side (*e.g.*, additional databases may need to be integrated into the AMAAIS infrastructure), are likely to be the same irrespective of where monitoring and accounting information comes from, and are hence left outside of consideration.

First, due to the additional energy consumption at a constrained device, device batteries need to be recharged or replaced more often which may require additional physical visits on-site. Even if energy harvesting technologies are used, energy usage for metering purposes reduces the amount of energy available for other tasks that a constrained device may be assigned to.

Second, gathering and transmitting the management information may consume notable memory at a device, either for data storage and transmission buffers or for code implementing it. In this case, the resources of a device may prove insufficient, and as a result Class-1 devices may need to be replaced with more expensive Class-2 devices.

Finally, as the volume of management information being transmitted grows, the available short-range radio network capacity may become exhausted – for instance, if hundreds of devices are transmitting their information at approximately the same point in time. Should this be the case, additional access points may need to be introduced in order to handle the radio traffic, thereby also inducing additional cost.[6]

A number of factors determining metering-related costs have been identified. In particular, these costs depend on:

- How many variables are metered. The more variables need to be monitored, the longer the metering process executes, the larger the volume of information to be transmitted, the greater the chance of fragmentation and packet reassembly. As a result, memory consumption and data communications overhead increase with the number of variables being metered.
- How often metering is performed. More frequent metering implies that a device needs to be awake to acquire information, which has an associated energy footprint. Besides, in case metered information is not transmitted immediately but accumulated first at the device, a more frequent metering also necessitates greater memory buffers to be allocated at the device for storing intermediate metered data.

[6] It shall be noted that the exhaustion of radio network capacity greatly depends on the respective traffic pattern, which in turn is application-specific. Therefore, according to domain experts, the radio network capacity offered by a single access point in the proximity of constrained devices is sufficient in many application scenarios.

– How often metered data is communicated. More frequent delivery of metering information incurs a data communications overhead that drains the device battery.
– What protocols and mode of communications are used. Communication protocols and encoding formats bring additional overhead due to maintaining communication sessions, handling verbose headers, etc. This may make processing more complex and increase the data communications' energy footprint. For instance, for short messages, the message size when using TCP and HTTP is several times greater than the message size when using UDP and CoAP [14]. Likewise, when the communication mode is periodic with regular intervals, the energy footprint at a constrained device can be reduced significantly by putting the device to sleep between communications sessions, as compared with the ad-hoc mode of communication where a device is engaged in radio duty cycling, to be ready to respond to incoming requests.
– Whether security features of communication protocols are used. When using authentication and encryption mechanisms, a constrained device needs to perform additional computations and transmit additional data, as a part of an authentication and encryption procedure, or due to the need to include security-specific headers into transmitted messages. This will have a negative effect on battery lifetime. Besides, collateral additional memory requirements may imply a need to acquire more expensive devices, thus, further increasing hardware costs.

While assessing the absolute value of extra costs induced by the metering process is important, it is also useful to analyze how large they are in comparison with costs induced by the core functionality of the application. Let us consider three possible scenarios which differ in the costs induced by the metering process. For simplicity, let us assume that in all scenarios the capabilities of Class-1 devices would suffice both for application-specific functions and for management functions related to accounting and monitoring data acquisition and transfer. Let us also assume for simplicity, that available access points and/or gateways are sufficient for processing both application-specific data as well as management data, *i.e.*, that neither new access points nor gateways are needed. Therefore, costs of the memory footprint and access point overload may be ingnored, and all subsequent consideration may focus solely on the cost of the energy footprint induced by the metering process.

Let E_{A} and E_{M} denote the energy consumed for acquiring and transmitting application-specific (A) data and metered management (M) data for monitoring and accounting purposes, respectively. Three cases could be distinguished:

Minor overhead. Let us first consider a scenario where acquiring and transmitting both application-specific data and metered management information does not consume significant resources. For example, one could imagine an application where snow level is measured and transmitted to pre-defined recipients every five minutes, along with management information. In this scenario, metered management information is acquired and transmitted along with the application-specific payload, with the same interval, similar to the prototype implementation described in the previous section. Furthermore, both application-specific and management information fit into a single frame, thus, avoiding fragmentation and packet reassembly. In between data acquisition and transmission sessions, the device is put to sleep mode wherein energy consumption is negligible. In this scenario, the energy footprint is minimal, *i.e.*, $E_{\mathrm{A}} \to 0$ and $E_{\mathrm{M}} \to 0$, and battery lifetime is determined by the self-discharge process rather than by the application-specific and management functionality. Thus, in this scenario, any metering-related overhead can be ignored when analyzing costs and benefits of implementing the metering functionality of the device.

Medium overhead. Let us now turn to a scenario where the energy footprint of the application and management data acquisition and transmission is not negligible, either because of more verbose data being acquired, or because it needs to be acquired frequently, say, once per minute or more often. Metered data is also assumed to be transmitted separately from application-specific payload, as it needs to be transmitted at different intervals or since application or protocol implementation details, security requirements, or other considerations demand management-related information to be transmitted separately from application payload. For example, in an environment monitoring application, sensing devices can be shared by multiple organizations. Environmental measurements can be delivered to the customer organizations upon request, whereas metered data for monitoring and accounting purposes can be periodically delivered to the management system.

In this scenario, the volume of data metered for monitoring and accounting purposes is still assumed to be comparable to the volume of the application payload being sent. Accordingly, both types of data are assumed to generate data communication overhead – and hence an energy footprint – of the same

order of magnitude, *i.e.*, $E_A \approx E_M > 0$. The impact of the metering process on a device's battery lifetime grows as the frequency of the metering process execution increases (as visible in Table 3). Therefore, the frequency of management information acquisition needs to be justified by the benefits it is expected to bring, such as

- Better granularity of charging and accounting information
- Better information for status monitoring and troubleshooting
- Better information for performance optimization, network planning

Large overhead. Finally, in the third scenario, management data is not only transmitted separately from the application payload, but is also transmitted more frequently than the payload, and possibly also with greater message size. For instance, in applications where a specific environmental condition is to be detected, such as avalanches or earthquakes, the notification of a detected event may need to be dispatched rarely (note that, depending on the application, a device still may have to awake frequently or be constantly on in order to timely detect an event of interest). Meanwhile, since ensuring proper operational status of devices is critical, metered information for monitoring and accounting purposes needs to be acquired and transmitted regularly.

In this scenario, the acquisition and transmission of management information is likely to generate a notably greater energy footprint than the core application does, *i.e.*, $E_A \ll E_M$. Still, the additional costs caused by the acquisition of management information are justified by strategic needs of the application, which shall be reflected in its non-functional requirements, such as a high availability requirement.

These three scenarios differ in how great an impact on battery lifetime the acquisition and transmission of management data has, in comparison to core functions of the respective application. This impact varies from being a negligible factor in the first scenario to becoming the main contributor of energy consumption in the last scenario. In summary, the additional costs of acquiring management data can be ignored if it is acquired and transmitted infrequently (once per five minutes or less frequently), or if this data can be compactly represented as a part of an application message. On the other hand, if the data metered for monitoring and accounting purposes is acquired and transmitted more often and/or in greater volume than application-specific data, then these extra costs become noticeable and therefore need to be justified by the expected benefits and/or a need to satisfy certain (non-functional) requirements.

6 Conclusions

The management of constrained devices in general, and in particular, the monitoring of their operational status and performance level, as well as the accounting of resource usage of these devices are of great importance for applications in the emerging IoT field. For organizations implementing or deploying IoT applications, the availability of monitoring and accounting information offers numerous benefits, such as better granularity of charging and billing information, support for device status monitoring and troubleshooting, and support for performance optimization and network planning.

Therefore, the capability of acquiring monitoring and accounting information shall be built in future IoT applications. The implementation of such a capability may be facilitated if the necessary building blocks are provided by an implementation framework that uses standard components and protocols and offers a publicly available API to access its functionality.

This work has studied the applicability of CoAP, a lightweight IETF protocol under development, for acquiring and transmitting management information from the constrained devices, and it has also investigated the applicability of AMAAIS, which provides standard building blocks for the accounting and monitoring of IT services, for storing, processing, and distributing management information retrieved from constrained devices. As a part of the study, a meter component responsible for acquiring and transmitting the management information has been prototypically implemented, and the AMAAIS infrastructure has been extended with components enabling the retrieval of information from constrained devices. CoAP was used as a transfer protocol for the communication link between the meter and AMAAIS.

Based on the results obtained from experiments with the implemented prototype, CoAP has been found suitable for transferring monitoring and accounting information from constrained devices. Its strong sides include the simplicity and compactness of the implementation, as well as a small energy footprint attributed to a small size of messages being transferred and the use of the observer design pattern. The protocol is especially suitable in the application scenarios where it is also utilized for exchanging application-specific information, since in this case the protocol implementation code can be reused and hence the size of memory required can be minimized.

Due to the use of standard libraries implementing CoAP-based communication, the memory footprint of the meter is small, thus, allowing it to run on relatively low-power Class-1 devices, such as TmoteSky. Furthermore, owing to the use of CoAP, the impact of the metering process on device battery lifetime is relatively modest, too. The latter depends on (i) whether the metered data is acquired infrequently and transmitted along with application-specific data within the same frame, and (ii) whether a constrained device needs to be constantly available. In case a single frame suffices to transfer all the data, and in case periods of unavailability can be tolerated, battery lifetime is rather long, reaching 1.5 years in the experiements conducted. On the other hand, making a constrained device available all the time shortens the battery lifetime considerably, with the maximum of circa six months according to estimates obtained in the experiments.

The use of CoAP also imposes some constraints on the management functionality implementation. Specifically, it restricts the volume of information that can be retrieved without making the implementation more complex and/or less energy efficient. This is due to the fact that the use of the "Observe" option of the protocol assumes that information is transmitted within a single CoAP package, which, to avoid packet fragmentation and reassembly at the lower levels in the protocol stack, effectively limits the size of a message to circa 80 Byte. As a result, the use of JSON representation for encoding messages becomes infeasible in all but trivial cases due to its considerable size.

The use of the "Observe" option also brings the need to confirm successful message reception by the implemented networked Accounting Application, as otherwise, according to CoAP specifications, the subscription to metered information updates is canceled. This in turn implies that a connection loss will nullify a subscription, and its re-initiation shall be implemented at the application level by the networked Accounting Application. It should be mentioned that the CoAP standardization process is not completed yet, and therefore some changes may still be introduced, possibly making these side effects less restrictive.

The obtained results also suggest that the AMAAIS infrastructure can be relatively easily adapted for IoT applications, and that it is well suited for processing, storing, and distributing metered accounting and monitoring information. With the help of attribute-based filtering components, it is possible to define recipients for particular subsets of information: for instance, subscriber organizations could be configured to receive and process sensor measurement updates, while a management center could be configured to receive and persist accounting and monitoring information – both types of information could be transmitted from a device within a single CoAP message. From a perspective of implementation efforts required, the integration with AMAAIS is relatively straightforward and requires days to a few weeks rather than months of efforts: the application developer only has to implement the networked Accounting Application for retrieving metered data from constrained devices, implement two Java classes to support efficient persistency, and specify recipients of metered information as well as other details of the event processing pipeline by editing configuration files.

The work reported in this paper has several limitations that shall be addressed in future work. First, the prototype implementation shall be expanded to allow additional ways of triggering the metering process, include additional attributes to be metered at constrained devices, and allow a reconfiguration of metering process parameters – in addition to the currently reconfigurable metering interval. Second, the EXI representation shall be used to encode metered data, either with the help of available libraries, or by editing specific places in a hardcoded binary EXI messages. Once the specification of CoAP DTLS security is finalized, the prototype shall be equipped with the required security mechanisms. Finally, in future work, performance characteristics of the prototype shall be further empirically evaluated using field experiments accompanied by analytical modeling. As a part of such evaluation, costs and benefits associated with the acquisition of monitoring and accounting information shall be compared, in order to assess how well the expected benefits justify the costs.

Acknowledgements

This work has been performed partially in the framework of the AMAAIS project as part of the "AAA/ SWITCH e-infrastructure for e-science" programme under the leadership of SWITCH, the Swiss National Research and Education Network, and has been supported by funds from the State Secretariat for Education and Research (SER). The work has been also partially carried out within the framework of the Internet of Things Program of TIVIT Oy nominated to organize and manage the programs of the Strategic Center for Science, Technology and Innovation in the field of ICT funded by the Finnish Funding Agency for Technology and Innovation (TEKES).

References

1. TMN management functions. ITU-T Recommendation M.3400, February 2000.
2. More Than 50 Billion Connected Devices. Ericsson White Paper, February 2011.
3. D3.1 Architectures of BUTLER Platforms and Initial Proofs of Concept, October 2012.
4. Wasai Shadab Ansari, Atif M. AlAmri, Mohammed Mehedi Hassan, and Muhammad Shoaib. A survey on cloud-sensor: Architecture, applications and approaches. *International Journal of Distributed Sensor Networks*, 2012.
5. Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Comput. Netw.*, 54(15), October 2010.
6. A. Bierman, D. Romascanu, and K.C. Norseth. Entity Sensor Management Information Base. RFC 3433 (Proposed Standard), December 2002.
7. C. Bormann and M. Ersue. Terminology for Constrained Node Networks. draft-bormann-lwig-terms-00, November 2012.
8. Carsten Bormann, Angelo Paolo Castellani, and Zach Shelby. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012.
9. Mike Botts and Alexandre Robin. OpenGIS Sensor Model Language (SensorML) Implementation Specification. OpenGIS Implementation Specification, July 2007.
10. D. Caputo, L. Mainetti, L. Patrono, and A. Vilei. 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*, pages 770 –774, july 2012.
11. Shafique Ahmad Chaudhry, George Boyle, Weiping Song, and Cormac J. Sreenan. Emp: A network management protocol for ip-based wireless sensor networks. In *ICWUS*, pages 1–6. IEEE, 2010.
12. Haksoo Choi, Nakyoung Kim, and Hojung Cha. 6lowpan-snmp: Simple network management protocol for 6lowpan. In *HPCC*, pages 305–313. IEEE, 2009.
13. B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), January 2008.
14. W. Colitti, K. Steenhaut, N. De Caro, B. Buta, and V. Dobrota. Evaluation of constrained application protocol for wireless sensor networks. In *Local Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*, pages 1 –6, oct. 2011.
15. Simon Cox. Observations and Measurements - XML Implementation. OGC Implementation, March 2011.
16. Adam Dunkels. The ContikiMAC Radio Duty Cycling Protocol. Technical Report T2011:13, Swedish Institute of Computer Science, December 2011.
17. Adam Dunkels, Fredrik Österlind, Nicolas Tsiftes, and Zhitao He. Software-based sensor node energy estimation. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 409–410, New York, NY, USA, 2007. ACM.
18. R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241 (Proposed Standard), June 2011.
19. M. Ersue, D. Romascanu, and J. Schoenwaelder. Management of Networks with Constrained Devices: Use Cases and Requirements. Internet Draft 02, IETF, October 2012.
20. R. Gerhards. The Syslog Protocol. RFC 5424 (Proposed Standard), March 2009.
21. Christophe Huygens and Wouter Joosen. Federated and shared use of sensor networks through security middleware. In *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, ITNG '09, pages 1005–1011, Washington, DC, USA, 2009. IEEE Computer Society.
22. B. Jennings, R. Brennan, W. Donnelly, S.N. Foley, D. Lewis, D. O'Sullivan, J. Strassner, and S. van der Meer. Challenges for federated, autonomic network management in the future internet. In *Integrated Network Management-Workshops, 2009. IM '09. IFIP/IEEE International Symposium on*, pages 87 –92, june 2009.
23. C. Jennings, Z. Shelby, and J. Arkko. Media Types for Sensor Markup Language (SENML). Internet draft, IETF, October 2012.
24. M. Karsten, J. Schmitt, B. Stiller, and L. Wolf. Charging for packet-switched network communication-motivation and overview. *Comput. Commun.*, 23(3):290–302, February 2000.
25. Matthias Kovatsch, Simon Duquennoy, and Adam Dunkels. A low-power coap for contiki. In *Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2011)*, Valencia, Spain, October 2011.
26. Matthias Kovatsch, Simon Mayer, and Benedikt Ostermaier. Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things. In *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012)*, Palermo, Italy, July 2012.
27. Siarhei Kuryla and Jürgen Schönwälder. Evaluation of the resource requirements of snmp agents on constrained devices. In *Proceedings of the 5th international conference on Autonomous infrastructure, management, and security: managing the dynamics of networks and services*, AIMS'11, pages 100–111, Berlin, Heidelberg, 2011. Springer-Verlag.
28. Tobias Kurze, Markus Klems, David Bermbach, Alexander Lenk, Stefan Tai, and Marcel Kunze. Cloud Federation. In *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*. IARIA, September 2011.

29. Laurent Lefort, Cory Henson, and Kerry Taylor. Semantic Sensor Network XG Final Report. W3c incubator group report, W3C, June 2011.
30. Ignacio M. Llorente. Key Challenges in Cloud Computing to Enable Future Internet of Things, January 2012.
31. Yi-Wei Ma, Jiann-Liang Chen, Yueh-Min Huang, and Mei-Yu Lee. An efficient management system for wireless sensor networks. *Sensors*, 10(12):11400–11413, 2010.
32. Neha Mehrotra and Nitin Dangwal. Interoperate in Cloud with Federation. View point, August 2011.
33. Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. Technical report, July 2009.
34. R. L. Morgan, Scott Cantor, Steven Carmody, Walter Hoehn, and Ken Klingenstein. Federated Security: The Shibboleth Approach. *EDUCAUSE Quarterly*, 27(4):12–17, 2004.
35. Hamid Mukhtar, Kang-Myo Kim, Shafique Ahmad Chaudhry, Ali Hammad Akbar, Ki-Hyung Kim, and Seung-Wha Yoo. Lnmp- management architecture for ipv6 based low-power wireless personal area networks (6lowpan). In *NOMS*, pages 417–424. IEEE, 2008.
36. A. Okmianski. Transmission of Syslog Messages over UDP. RFC 5426 (Proposed Standard), March 2009.
37. F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641 –648, nov. 2006.
38. Linnyer B. Ruiz, José Marcos S. Nogueira, and Antonio A. F. Loureiro. Functional and information models for the MANNA architecture. In *Proceedings of the 5th French Conference on Networks and Services Management (GRES'03)*, pages 455–470, Fortaleza, CE, Brazil, February 2003.
39. Linnyer B. Ruiz, José Marcos S. Nogueira, and Antonio A. F. Loureiro. MANNA: a management architecture for wireless sensor networks. *IEEE Communications Magazine*, 41(2):116–125, February 2003. ISSN 0163-6804.
40. Corinna Schmitt, Lothar Braun, Thomas Kothmayr, and Georg Carle. Collecting sensor data using compressed ipfix. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 390–391, New York, NY, USA, 2010. ACM.
41. John Schneider and Takuki Kamiya. Efficient XML Interchange (EXI) Format 1.0. Recommendation, W3C, March 2011.
42. J. Schnwlder, T. Tsou, and B. Sarikaya. Protocol profiles for constrained devices. In *Proceedings of the IAB Workshop on Interconnecting Smart Objects with the Internet*, February 2011.
43. A. Sehgal, V. Perelman, S. Kuryla, and J. Schnwlder. Management of resource constrained devices in the internet of things. *IEEE Communications Magazine*, 50(12), December 2012.
44. Martín Serrano, Steven Davy, Martin Johnsson, Willie Donnelly, and Alex Galis. The future internet. chapter Review and designs of federated management in future internet architectures, pages 51–66. Springer-Verlag, Berlin, Heidelberg, 2011.
45. Z. Shelby. Embedded web services. *Wireless Communications, IEEE*, 17(6):52 –57, december 2010.
46. Ian G. Smith, Ken Sakamura, Anthony Furness, Ricky Ma, Yong-Woon Kim, Eldor Walk, Craig Harmon, Paul Chartier, Patrick Guillemin, and David Armstrong. Casagras final report. Technical report, October 2009.
47. Thomas Staub, Benjamin Nyffenegger, Desislava C. Dimitrova, and Torsten Braun. Operational support of wireless mesh networks deployed for extending network connectivity. In Javier Ser, Eduard Axel Jorswieck, Joaquin Miguez, Marja Matinmikko, Daniel P. Palomar, Sancho Salcedo-Sanz, Sergio Gil-Lopez, Ozgur Akan, Paolo Bellavista, Jiannong Cao, Falko Dressler, Domenico Ferrari, Mario Gerla, Hisashi Kobayashi, Sergio Palazzo, Sartaj Sahni, Xuemin (Sherman) Shen, Mircea Stan, Jia Xiaohua, Albert Zomaya, and Geoffrey Coulson, editors, *Mobile Lightweight Wireless Systems*, volume 81 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 322–329. Springer Berlin Heidelberg, 2012.
48. Burkhard Stiller. Accounting and monitoring of AAI services. *SWITCH Journal*, 2010(2):12–13, October 2010.
49. K. Terplan. *OSS essentials: support system solutions for service providers.* Wiley computer publishing. John Wiley, 2001.
50. Craig Walls and Ryan Breidenbach. *Spring in action.* Manning Publications Co., Greenwich, CT, USA, 2007.