Chatty Things - Making the Internet of Things Readily Usable for the Masses with XMPP

Ronny Klauck
IHP
Im Technologiepark 25, D-15236 Frankfurt (Oder), Germany
eMail: klauck@ihp-microelectronics.com

Michael Kirsche
Computer Networks and Communication Systems Group
Brandenburg University of Technology Cottbus, Germany
eMail: michael.kirsche@tu-cottbus.de

***Abstract*—An important challenge for the Internet of Things is the gap between scientific environments and real life deployments. Smart objects need to be accessible and usable by ordinary users through familiar software and access technologies to facilitate any interaction and to increase their acceptance rate. This work deals with a seamless integration, discovery, and employment of smart objects into the Internet infrastructure under Human-to-Machine (H2M) communication aspects. We introduce an XMPP-based service provisioning sublayer for the IoT to integrate resource constrained devices seamlessly into the Internet by showing how XMPP can empower the collaboration between humans and smart objects. To meet the requirements of constrained devices, we propose to extend XMPP's publish-subscribe capabilities with a topic-based filter mechanism to effectively reduce the number of exchanged XMPP messages. We further present standardized bootstrapping and handling processes for smart objects that adapt automatically to infrastructure and ad hoc network environments and do not require predefined parameters or user interaction. The applicability of XMPP for constrained devices is further demonstrated with an XMPP client and mDNS/DNS-SD service for the Contiki operating system.**

***Index Terms*—Internet of Things, XMPP, H2M, Contiki**

I. Introduction

The use of wireless-enabled mobile devices has grown exponentially during the last couple of years and created scenarios in which access to manifold services from ubiquitous resources is desired without requiring specific knowledge [1]. Smartphones represent a programmable and flexible platform with support for a wide range of applications while at the same time leveraging from the human element when carried as ubiquitous and pervasive commodity hardware [2]. Smart objects, in contrast, are deeply embedded in the physical world and therefore not always perceivable by users [3]. The integration of smart objects into the Internet enables an interaction with the physical world for humans through computer-based and mobile devices [4], facilitating the Internet of Things (IoT) vision [5]. This paradigm of interconnected (via ad hoc) as well as Internet-connected (via infrastructure) objects and devices (things) is fundamental for any pervasive and ubiquitous networking and computing vision. A seamless integration of smart objects into the current Internet infrastructure requires at least two integral parts: in-network localization (discovery, interoperability, addressability) through a standardized scheme that complies with the Internet's IP standard as well as self-configuration (“arrive and operate”, scalability) to handle large quantities of devices according to the IoT vision [6]. As IP is no longer restricted to ordinary computers due to the development of small IP stacks [7], embedded and resource constrained devices can nowadays be directly connected to IP-based networks for various application scenarios.

Integration should always focus on the user: instead of requiring users to learn new interaction schemes to access data from their environment, smart objects should be integrated seamlessly into the Internet infrastructure with known and standardized approaches. As users already use their smartphones for communication (e.g., messaging and chat) and staying in touch with their (human) environment, we prefer approaches that support the Internet's end-to-end principle and solutions that integrate into the software that a user is familiar with. We therefore chose the *Extensible Messaging and Presence Protocol* (XMPP) [8], which is standardized by the IETF and widely deployed for real-time data stream and Instant Messaging (IM) applications. XMPP itself is a set of flexible and open XML technologies that are expandable by protocol extensions (XEPs) to adopt to various environments and scenarios. From the network point of view, using XMPP as the default communication protocol allows us to realize pervasive networking without the need for a protocol gateway or a middleware. XMPP simplifies the interconnection of devices [6] and it is the basis for our Human-to-Machine (H2M) communication between various device classes. An important aspect for pervasive networking is that XMPP provides ad hoc (P2P) communication with *XEP-0174 Serverless Messaging* [9] via *Multicast DNS* (mDNS) [10] and *DNS Service Discovery* (DNS-SD) [11]. These technologies are based the IETFs Zeroconf working group work that facilitates standards in the field of service-oriented networking, also known as *Bonjour*. Both, XMPP and Bonjour, offer a rich variety of open source software for servers, clients and libraries, supporting several mobile and desktop operating systems.

The rest of this work is structured as follows. Section II discusses related work. Section III introduces *Chatty Things* while Sections IV, V, and VI present technical requirements and architectural solutions for H2M to discover and collaborate with smart objects in IP-based networks through XMPP. Initial performance evaluations are presented in Section VII while concluding remarks in Section VIII complete this work.

II. Related Work

Current application layer protocol solutions for resource constrained devices (e.g., the Constrained Application Protocol (CoAP) [12] or the Message Queuing Telemetry Transport (MQTT) protocol [13]) focus on Machine-to-Machine (M2M) communication to address and manage smart objects. Both example protocols require application protocol gateways to connect devices to the Internet since the protocols are not compatible with the currently established infrastructure. Application protocol gateways introduce additional complexity in terms of message translation and protocol version support [14]. Message translation is typically time-consuming and failure-prone [15] and involves a loss of flexibility and end-to-end functionality from a protocol [6] and security [16] point of view. Protocol gateways are a limiting factor we want to omit when integrating smart objects in IP-based infrastructures while providing “simple, yet powerful and generally usable abstractions” for the development of IoT applications [17].

Approaches to overcome this issue are diverse, web services and middlewares being two of them. The Devices Profile for Web Services (DPWS) for embedded devices (uDPWS) [18] resorts to standardized protocols like IP, UDP/TCP, and SOAP to realize web services for embedded microcontrollers to integrate them in existing infrastructure. Services like dynamic discovery, subscribing to services, and receiving events from web services are enabled with uDPWS. The main drawback at the moment is that DPWS is only available for systems running Microsoft’s Windows operating system and Microsoft’s Universal Plug’n’Play stack. An application- and network-independent announcement of services to couple different device types should be favored instead.

Sensor Web Enablement [19] is a standard developed by the Open Geospatial Consortium. It represents a generic framework for a platform- and protocol-independent interaction between sensors to realize complex scenarios while at the same time requiring large infrastructure support [19, Sec. 4.1.]. Its appliances depend on a complex middleware for the sensor network management, whereas different middleware versions can disturb the cooperation if needed updates fail.

Our focus lies on the use and adaptation of established protocols instead of introducing new protocols or complex software architectures. We adapt the established XMPP and mDNS / DNS-SD protocols to integrate smart objects seamlessly into the current infrastructure, to discover and locate their offered applications and services, and to ease their handling and employment by the user as well as to boost the development of collaborative IoT applications at higher layers. XMPP [8] is a widely deployed and standardized communication protocol that facilitates the publish-subscribe paradigm, which enables entities to interact efficiently with each other based on their own context. mDNS [10] and DNS-SD [11] are parts of the *Zeroconf* initiative to standardize network discovery mechanisms for devices ranging from desktop computers to smartphones. Both protocols together enable an application- and network-independent announcement of services and both are nowadays supported on nearly every operating system. Initial steps for running XMPP on resource constrained devices were *uXMPP* [20] and the *XMPPClient for mbed*[1], which provided rudimentary and lightweight implementations but no conclusive scientific results for the use of XMPP in the context of the Internet of Things.

III. Introducing Chatty Things

An advantage of XMPP is that it will significantly improve H2M communication, because users will be able to interact directly with their environment through standard chat clients, a software all users are familiar with today, in a way they already know from interacting with friends in social networks or chat rooms. Users can communicate with and gain access to information from smart objects using standard technologies, which are already available for every operating system nowadays. Example tasks are: organizing things in a contact list (XMPP roster), subscribing to *thing provided* topics that users are interested in (what happens in the user’s neighborhood) or automatic interconnecting of things to control a monitored environment (M2M). This introduced style of communication via chat clients extends plain smart objects in the Internet of Things to *Chatty Things* at the application layer. Chatty Things use XMPP as the underlying communication protocol to interact seamlessly with different device classes from the established Internet. XMPP provides an intuitive information handling and a central notification service for users on their mobile devices and computers. As mobility becomes the most important criteria in our daily lives, people are interested in gaining information on their environment, which could affect them at the moment (e.g., traffic jam, weather) or in the near future (e.g., forecasts, earthquakes), as well as gaining control over environmental conditions (e.g., room temperature, light). The publish-subscribe paradigm of XMPP will help users to filter information according to their specific interests or to important events they have subscribed to, because there will be a huge set of data which can be collected from environment-embedded devices. For that reason, important data changes can be announced with XMPP as presence updates in real-time (e.g., a dedicated threshold is reached, a critical amount of water is detected in the basement) to realize a target-oriented communication between humans and their IoT-driven environment. Nearby XMPP resources can appear dynamically in a user’s roster and will advertise information that is of interest for the user. Bookmarking Chatty Things in the XMPP roster will allow users to permanently receive relevant updates of things similar to presence updates of their friends, instead of requesting links (polling) to get updates. This is a huge benefit in contrast to the Web-of-Things approach, which only presents information on a website and ensures the browseability of resources “by clicking on links” [21, Sec. V.A]. Identifying required resources via links is inefficient because a link is defined by a long string of characters, which is hard to remember when accessing smart objects spontaneously [22].

[1]mbed XMPPClient [Online] http://mbed.org/cookbook/XMPPClient

A (not ideal) solution for this are short links [23], but they require an additional shortening service, acting as a translator from long to short links and vice versa, causing performance and security issues (e.g., phishing [24], spam [25]).

A. H2M Collaboration Challenges

Based on our vision of Chatty Things, various challenges for the H2M collaboration between humans (using ordinary computational devices) and smart objects are exposed [6, IV]:

- **Interoperability and Discovery:** since the tasks of smart objects in the IoT vision can be manifold, these devices need a standardized communication schema to announce their availability and their advertised services for being identified automatically or found by nearby users via discovery, look up or name services.
- **Self-Configuration:** users do not want to struggle with a complicated setup or manual programming for large numbers of devices. This requires support for the diversity of smart object technologies as well as the integration of a flexible bootstrapping process for any subsequent interaction between devices in hybrid networks [4].
- **Information Filtering**: monitoring of real-world scenarios produces huge data amounts. Users need a mechanism to address information or events target-oriented, which supports a low message overhead and a low data-rate.

B. Our XMPP-based Solution Approaches

As the integration of smart objects does not stop at the network layer [17], we aim to provide a XMPP-based integration at the application layer to advertise advanced services to enable consumer-friendly IoT applications with:

- **Service Provisioning Sublayer** (Section IV),
- **Parameter-less Bootstrapping** (Section V),
- **Temporary Subscription for Presence** (Section VI).

By using XMPP as the underlying communication protocol, resource constrained devices can offer environmental information for humans through standard chat clients and thus boost the easy collaboration with smart objects in our daily lives.

IV. Service Provisioning Sublayer for IoT

The main focus of our service provisioning sublayer (refer to Fig. 1) is the ubiquitous collaboration of Chatty Things with devices and standard applications to seamlessly integrate smart objects in IP-based networks. Usability and user autonomy shall be increased by abstracting specific devices as services according to the concept of Service-oriented Architectures (SOA) [26]. Such an abstraction of functionality into services provided through a network is what non-technical users need to get in touch with smart objects. The users' interaction with their environment will hence be simplified by browsing through the network for published services and subscribing to them for new information. Service provisioning here means providing common services (discovery, authentication, identity management, security), a bandwidth-efficient communication (via publish-subscribe) and a high flexibility (i.e. expansion

Application Layer	Application 1	Application 2	XEP-based Application
	XMPP Core + XEP-0174 (as common service provisioning sublayer)		
Transport Layer	TCP		UDP
Internet Layer	IPv6		uIPv6 & 6LoWPAN
Link Layer	Ethernet	802.11	802.15.4

Fig. 1. XMPP-based Service Provisioning Sublayer for the IoT

with new protocol features) while omitting intermediate systems (gateway, proxy). Without a standardized application protocol, each vendor would develop its own solution leading to fragmentation and non-interoperability, hindering and lowering the acceptance rate of the IoT [27]. XMPP and mDNS / DNS-SD ensure a network- and device-independent interaction via unique *Jabber IDs* (JIDs) and automatic entity detection. Protocol gateways are hence not necessary to offer a transparent access. Moreover, the publish-subscribe paradigm of XMPP, a highly scalable, bandwidth and energy efficient event distribution system [28], announces only changes in sensed data to interested entities. XMPP, as a standardized protocol, will increase the acceptance rate of smart objects for IoT vendors, network administrators and application programmers, because XMPP-empowered Chatty Things can be seamlessly integrated at the application layer allowing programmers and administrators to use existing tools and handling expertise while reducing integration costs and compatibility tests with currently used software. This boosts the development of a consumer-friendly interaction via a standardized communication scheme with the Internet of Things.

A. Technical Requirements and Architectural Solutions

Embedded microcontrollers have limited memory and computing resources as well as low bandwidth (e.g., 127 Bytes max. packet size for IEEE 802.15.4). The XMPP-based service provisioning sublayer for resource constrained devices must be memory-efficient and extremely lightweight while including the most essential functions (using existing or possibly new XEPs) for typical IoT appliances and H2M / M2M communication, such as presence and message exchange, grouping of devices, information filtering, and support for hybrid smart object networks. For this reason we chose a building blocks concept for replaceable XMPP features to ensure a predictable memory consumption and to support different use cases: the service provisioning sublayer consist of a modular XMPP stack that implements XMPP Core / IM and *XEP-0174 Serverless Messaging* as common services for smart objects. Additionally required XEPs can be implemented on-top on demand (refer to Section IV-C). The XMPP software stack consists of different components and modules that can be activated during compile time. Figure 2 depicts the appropriate software stack. The displayed components (e.g., XMPP client, *XEP-0174* client) can either be combined or used separately as stand-alone modules. Both clients implement the API of the available XMPP function set.

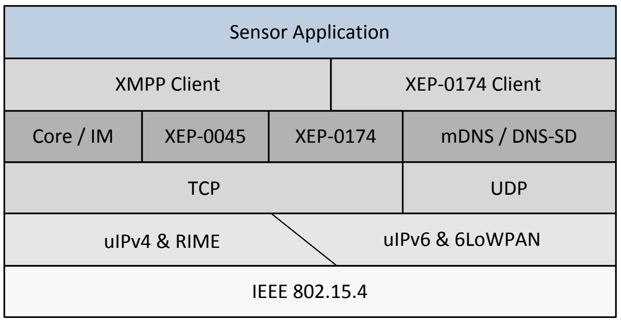

Fig. 2. Sensor Application with XMPP Software Stack

We use the *uXMPP* project [20] as a starting point for the development of our modular XMPP software stack. The initial version of uXMPP (v0.1) was released in 2009 for the Contiki operating system as an early proof-of-concept, implementing only certain XMPP Core functions. XMPP Core [8] specifies the use of TCP connections for the exchange of XML elements between XMPP entities over XML streams. As long as the XML stream is established, any number of XML elements (e.g., XMPP message, presence) can be transmitted in an efficient manner and in near-real-time. As the original code of uXMPP v0.1 is in an early developing state, no further controlling options, memory optimizations or low data-rate enhancements for smart objects are implemented while all implemented XMPP Core functions are fixed and hard-coded. To fulfill our goal of a highly memory-efficient and flexible service provisioning sublayer for the IoT we optimized the uXMPP prototype in several ways to realize a modular and lightweight XMPP software stack enriched with a useful feature set as the basis for the development of Chatty Things:

- **Message Flow Optimization:** Version 0.1 of uXMPP sends each XML element of a specified XMPP message in a separate and independent TCP packet. This leads to a strong increase of sent packets which we avoid by using the full available TCP payload length (refer to Table I);
- **Reduced Definition of Functions:** As the code footprint of an image increases with additional function definitions and external calls [29], we restructured the original code by using only one single file to avoid external calls and minimize function definitions as much as possible;
- **External API Calls:** Connecting both clients in the sensor application can be done through activating the external API calls which will consume more memory than the standalone versions of each client. But this offers a dynamic interface of selectable XMPP features which is accessible from outside of the modules and highly extensible (new XEPs can be easily integrated);
- **Compiler Flags:** We use several compiler flags[2] to reduce the firmware size of our improved uXMPP.

With these code optimizations, the improved uXMPP stack works very memory-efficient (using only a third of its original memory size) and leaves memory for additional XEPs and intuitive sensor applications on resource constrained devices while offering a dynamic API for IoT application developers.

B. New Features of our Modular XMPP Software Stack

Our improved uXMPP stack follows the building blocks concept that enables an appliance and sensor-specific protocol support. The XMPP client component uses the *Core / IM* module, which is an integral part of the client, as a basis. The publish-subscribe paradigm is used here and a simple XMPP (chat) client on notebooks or smartphones can be used to directly interact with smart objects. The *XEP-0045* (used for device grouping) and *XEP-0174* modules (includes the mDNS / DNS-SD module) are optional and can be selected on demand. Each feature is therefore realized as an independent module and can simply be enabled or disabled during compile time, depending on the actual duties and scenarios:

- **IPv6 Support:** Offers the possibility to manage and connect up to billions of devices via uIP over XMPP;
- ***XEP-0045 Multi-User Chat (MUC)*:** Implements entering and leaving of chat rooms as well as sending of group chat messages;
- ***ANONYMOUS*[3] JID Cutting for MUC:** Uses only a specified length of a JID as an alias for a chat room. This shortens the displayed name in a MUC of a smart object and reduces the size of a group message;
- ***XEP-0174 Serverless Messaging*:** Enables endpoint discovery via zero-configuration networking and implements the negotiation of a serverless communication via XML streams for the XMPP message exchange. This specific component is introduced in Section IV-C;
- **Temporary Subscription for Presence (TSP):** Enables a topic-based publish-subscribe mechanism for smart objects in a XMPP network to reduce network traffic and to offer more filter options for the user. This component is explained in detail in Section VI.

Our improved uXMPP implementation can easily be extended with additional features and XEPs, one of the main advantages of XMPP. Normal XMPP connections are covered by today’s security standards (TLS/SSL) to provide an adequate solution for end-to-end security. At the moment, secure connections via SSL/TLS are not implemented in uXMPP. In future work, we want to integrate embedded SSL and TLS implementations to provide end-to-end security for IoT scenarios by using solutions like MatrixSSL[4], which is designed for small footprint applications and devices. No secure login is hence available for uXMPP yet, we instead prefer *ANONYMOUS* as the login method for our uXMPP implementation because connecting nodes get a randomized and unique JIDs from the server. This ensures that no hard-coded or double-assigned JIDs for smart objects have to be programmed and that JIDs do not need to be pre-configured on a XMPP server.

[2]Reducing Contiki OS’ Firmware Size [Online] http://www.sics.se/contiki/wiki/index.php/Reducing_Contiki_OS’_Firmware_Size

[3]XEP-0175: Best Practices for Use of SASL ANONYMOUS [Online] http://xmpp.org/extensions/xep-0175.html

[4]MatrixSSL OpenSource Embedded SSL [Online] http://www.matrixssl.org

C. *Sensor Application and Feasible XEPs*

The sensor application addresses the built-in sensors of smart objects and uses the XMPP stack to send presence data to the network if a threshold value of a sensor is exceeded. A status icon in the XMPP roster is used to represent the condition of a sensor's measured value to the user. Thus, the status icons can be depicted (exemplary) like traffic lights and indicate changes of a monitored environment:

- **Green:** The measured sensor data is below the threshold value, no presence notification is necessary.
- **Yellow:** The sensor data exceeds the threshold value, a presence message will inform all registered users.
- **Red:** The sensor data overloads the threshold value, a presence or a chat message with the actual sensor value(s) is announced to all interested users.

Interacting with smart objects is realized with XMPP messages from an ordinary XMPP client: threshold values for the sensors can be adjusted, current values of each available sensor can be received, and the battery state can be monitored. The sensor application announces a command list when an XML stream is opened (*XEP-0174 Serverless Messaging*) or when a XMPP chat message arrives (XMPP IM). The concept is inspired by *XEP-0050 Ad-Hoc Commands*, which allows users to initiate a command session and to interact with an automated process through the XMPP client. A control and access mechanism could be provided with user interaction via remote commands while advertising and executing application-specific commands as defined in *XEP-0050* to realize a simple way to adjust thresholds (e.g., for alarming), to view historical measured data (e.g., exceeded thresholds, max. reached values), and to access the currently measured sensor data. Interacting with a set of smart objects can be realized with *XEP-0045 Multi-User Chat* through an automatic grouping of devices in a chat room during bootstrapping according to their integrated sensors. Sending a command message to a sensor-specific chat room will execute the command on all related devices. This eases the management of a set of smart objects by relying on a simple XMPP chat client.

XEP-0174 allows two entities to establish an XML stream without the need of a XMPP server while using mDNS [10] and DNS-SD [11] to discover entities that support XMPP and to identify their IP addresses and preferred ports, using the `_presence._tcp` DNS SRV service type. We developed our own tiny *XEP-0174*-based communication stack working with Contiki's integrated IP stack and consisting of a mDNS / DNS-SD service, called uBonjour, and a *XEP-0174* client for Contiki. uBonjour supports the resolving of hostnames and services as well as the registering, removing, and updating of services. It consumes (with optimizations enabled - *OWT*) 3.89 kBytes of ROM / 0.3 kBytes of RAM while the *XEP-0174* client handles incoming session requests and XMPP messages. Refer to [30] for a detailed introduction of uBonjour and its mode of operation.

Our implementation of the *XEP-0174* client for Contiki is also based on uXMPP with an added TCP listener process, because users should be able to initiate a chat session. The TCP handler will manage incoming TCP connection requests via a connection state and will accept opening XML streams from other entities in the network. The implementation of our *XEP-0174* client can be used in combination with uBonjour to easily establish *XEP-0174 Serverless Messaging* between smart objects and ordinary computers without the need for application protocol gateways. This gives users the possibility to interact spontaneously with nearby smart objects (e.g., if they are entering a room) by initiating an ad hoc chat session with a simple XMPP chat client to one of these devices. *XEP-0174* is mainly used in situations where no XMPP server exist (ad hoc network) or the connection to the XMPP server fails (i.e. a fall-back mechanism for XMPP Core/IM).

V. Parameter-less Bootstrapping

Self-configuration is essential for Chatty Things to realize a network-independent localization of smart objects and to integrate them seamlessly as described in Section III-A. This allows users to place Chatty Things everywhere – if no infrastructure network (e.g., no fixed access point) is available or devices are mobile, smart objects should automatically adapt to their environment by forming ad hoc networks and route information towards the infrastructure or to a dedicated smart object that is accessible by users [27]. Automated bootstrapping of Chatty Things is hence mandatory, independent from: user interaction, the used network environment (infrastructure or ad hoc), hard-coded start-up addresses (e.g., IP address of a XMPP server), and other pre-configurations (e.g., JIDs to log in on XMPP servers). If infrastructure services are either failing (e.g., router crashes, no connection to XMPP server) or not available during bootstrap (e.g., no XMPP server in the domain), smart objects should still be accessible for users via ad hoc communication (*XEP-0174*).

Failure-resistant network bootstrapping can be enabled by SOA [31]. SOA provides transparency with service abstraction for specific device functions as well as seamless interaction with various device types while devices browse their network domain for neighbors and newly published services [32]. With uBonjour (refer to [30]) we have implemented such a powerful tool as a lightweight service for resource constrained devices to discover and address devices and their provided services in different network environments. In combination with our *XEP-0174* client a parameter-less bootstrapping for hybrid networks can simply be realized for Chatty Things while communication between smart objects and humans can be established in ad hoc and infrastructure networks. Connecting a Chatty Thing to a server requires the IP address and port of the XMPP server. Since smart objects should be usable in different network environments and thus adapt themselves automatically, no pre-configured IP addresses can be used. Adding a new Chatty Thing to a network is possible by requesting services (e.g., XMPP server) or by receiving information from surrounding devices. A corresponding device responds with its list of DNS resource records that contains host / domain name(s), IP address, port, and assigned service(s).

Due to the scarce resources of constrained devices, not all parts of our XMPP software stack (introduced in Section IV-A) can be activated at bootstrap and run-time, because the dynamic memory use is a limiting factor. An intelligent handler process for both clients and uBonjour is therefore very important to combine memory-efficiency with flexibility. We use multi-level bootstrapping to reduce the memory consumption of uXMPP by enabling its components stepwise:

I: uBonjour will be activated to discover a XMPP server in the current network environment. If a XMPP server is found follow step II, else follow step III;

II: *Infrastructure mode:* deactivate the uBonjour client and connect to the XMPP server as *ANONYMOUS*;

III: *Ad hoc mode:* activate the *XEP-0174* client.

During run-time an automatic switching between infrastructure and ad hoc mode is triggered depending on incoming mDNS messages. The switching process follows these rules:

IV: In infrastructure mode: if the connection to the server gets lost and a predefined number of reconnect attempts failed then deactivate the XMPP client, follow step I;

V: In ad hoc mode: if a XMPP server joins the network then deactivate the *XEP-0174* client, follow step II.

These states allow us to realize a hybrid environment detection on resource constrained devices. The incremental activation of components is a resource-efficient way for the discovery, the self-configuration, and the seamless integration of Chatty Things in IP-based networks without the need for any user interaction or configuration. The devices will automatically discover their network environment and react autonomously on topology changes and un/available services. Compared to CoAP and MQTT, which always rely on a connection to a gateway (single point of failure), Chatty Things can be placed everywhere and can collaborate in every situation with users.

VI. Temporary Subscription for Presence (TSP)

As the flexibility and the extensibility of XMPP is based on XML, the heavy use of XML in low data-rate networks can cause a high network traffic through a large message overhead. Minimizing the overhead of XML-based messages can either be realized through XML compression techniques or through a simple reduction of the number of exchanged messages. XML compression techniques, expensive in terms of memory usage and CPU load, are not urgently necessary for the XMPP interaction of IoT components, because Chatty Things and users can actively reduce the number of messages by subscribing only to events and information they are really interested in (i.e. publish-subscribe [33]). This *Temporary Subscription for Presence* (TSP) approach implies a reduction of message overhead in terms of fewer retransmissions and refrains from using fragmentation while being compatible to the XMPP Core / IM standard. The *TSP* extension supports a real topic-based publish-subscribe for XMPP to reduce the network traffic of constrained Chatty Things by using small presence messages to signalize status updates (traffic lights). Extended information can be requested via remote commands. A topic-based filter to search for specific sensors (e.g., accelerometer, temperature) and a simple access control are provided on top of *XEP-0045*, which ensures the backward-compatibility of *TSP* to all existing XMPP chat clients. *TSP* allows users to retrieve updates about their local environment (temporarily) or events of specific sensors they have subscribed to (XMPP roster) without getting bombarded by information they are not interested in, while reducing the number of sent messages.

A. Presence Subscription Dilemma

The starting point of our *TSP* approach is the fact that *presence* messages (max. 38 Bytes) fit in a single TCP/IP packet (max. 48 Bytes – measured for IPv6 packets) within IEEE 802.15.4 radio frames (max. 127 Bytes). The reason is that presence information (not to confuse with directed presence [34, Sec. 4.6] or presence for entering a chat) is sent from a client without a 'from' or 'to' attribute. Table I shows the sizes of typical XMPP messages (e.g., presence [34, Sec. 4.4.1], one-to-one chat session [34, Sec. 5.2.1], presence to join a chat room [35, Example 18], and group chat message [35, Example 44]) and the number of used IP packets in Contiki for both uXMPP versions. The dilemma is that receiving this kind of presence message requires a manual subscription by the user to each sending entity for each joined network. As the network can change (e.g., smart objects leave, join, are dis/enabled) and a user can get in touch with different network environments (e.g., by just walking by), the subscriptions to objects are neither fixed nor stable and the number of publishing objects can increase considerably (according to the IoT vision). The XMPP roster of the user's chat client can thus become outdated very fast. Our solution to this problem is a dynamic and up-to-date roster that holds topic-related Chatty Things of the current network (XMPP domain) only temporarily to display their presence. If the user leaves the network, the temporary Chatty Things are removed automatically from its roster to keep it clean and up-to-date.

TABLE I
Size of Typical XMPP Messages and Sent IP Packets

Message Type	Size (in Byte)	uXMPP v0.1	Improved uXMPP
Presence	(max) 38	5 Packets	1 Packet
One-to-one Chat	164	12 Packets	4 Packets
Chat Join	101	-	3 Packets
Group Chat	164	-	4 Packets

B. Topic-based Publish-Subscribe

For the realization of the dynamic XMPP roster (*TSP*), we have implemented a topic-based publish-subscribe for XMPP to achieve a subscription of topic-based interests for users, because subscriptions of XMPP are bound directly to a node at the moment (*XEP-0060 Publish-Subscribe*). Therefore, we use *XEP-0045* as a basis for our access control: users can search for available chat rooms (the room name represents the topic of interest) and join them. By entering a chat room the user actively subscribes to a topic of interest. All Chatty Things in the topic-related chat room will be automatically added to the user’s roster, so that the user will only receive published information and events (sensor values) that he is currently interested in. This has the advantage that all *XEP-0045*-compliant XMPP chat clients can be used without modifications, because *TSP* has to be implemented only by the Chatty Thing and the XMPP server (refer to the following Subsections). The XMPP chat client Pidgin[5] remembers the last used chat rooms and rejoins them automatically if the user reconnects to a XMPP domain, so that the user will always be informed about events of nearby Chatty Things. Pidgin is available for a wide range of operating systems and can be used for free when no XMPP chat client is pre-installed.

The behavior of *XEP-0060* could cause bottlenecks in the message flow of a smart object network because such a network might consist of a large number of devices while utilizing only a small part of the bandwidth: a node always takes on the role of a subscriber when publishing data and will hence get information about every update just like a normal subscriber, which produces a large number of exchanged messages. With *TSP* enabled, Chatty Things can only take the role of a publisher: uninterested objects do not need to be informed about value changes of sensors, they just collect data and provide it to interested objects and users while lowering the network traffic. Users and Chatty Things (without activated *TSP*) can still act as subscribers and publishers, as defined in *XEP-0060*, with the enhancement of filtering information by topics. Our *TSP* extension enables Chatty Things to act only as publishers without getting updates from the XMPP network. Chatty Things with enabled *TSP* will receive no group chat or presence messages from the XMPP server.

C. Announcing Enabled TSP

The availability of *TSP* will be announced from a smart object by adding the `type` attribute to a presence message, which is used to join a chat room. The `type` attribute is already defined for this message type [35, Example 23] and will be recognized but not handled by unmodified XMPP servers. Adding the `type` attribute is automatically done by activating the *TSP* module in uXMPP. The following listing depicts an example (according to [35, Example 20]) of a *TSP*-enabled presence message:

```
<presence
      from='hag66@shakespeare.lit/pda'
      id='n13mt31'
      to='coven@chat.shakespeare.lit/thirdwitch'
      type='tsp'>
   <x xmlns='http://jabber.org/protocol/muc'/>
</presence>
```

The *TSP* flag is only a small addition to the original presence message and extends the message size with just a small amount of Bytes, as depicted by `type='tsp'` in the listing.

D. XMPP Server Modifications

We implemented a prototypical handling of *TSP* presence messages for the XMPP server *Prosody* (version 0.8.2) with modifications of its roster manager and its *MUC* plug-in. When a *TSP*-enabled presence message to enter a chat room is received by the XMPP server, it is going to check the `type` attribute for *TSP*. If the attribute is set, then the server will save the joining smart object as a new group member flagged with *TSP* to the room list. The server will also add the smart object to the roster and forward its presence message to every non-*TSP*-enabled room member. Afterwards, a roster update will be send to these XMPP clients and the smart object appears remotely in the user’s roster. Incoming group chat or presence messages will not be forwarded to room members with enabled *TSP*. Removing the smart object from the list is done automatically by the XMPP server when the node or the user is leaving the chat room while sending a presence of type “unavailable”, so no obsoleted smart objects will be held in the user’s roster. Overall, *TSP* only requires modifications of the XMPP server and the XMPP client of the smart object. The advantage is that existing XMPP clients do not need to be adapted while users can benefit from the topic-based filtering and smart objects from the reduced network traffic.

VII. Evaluation

We evaluated our XMPP software stack for Contiki in terms of memory footprint and TSP in terms of achievable message optimization. Therefore, our real world test setup seamlessly interconnects various stationary computers and workstations, mobile devices (e.g., netbooks, notebooks, smartphones), as well as embedded devices, sensors, and actuators over IP links. Figure 3 depicts this exemplary use case.

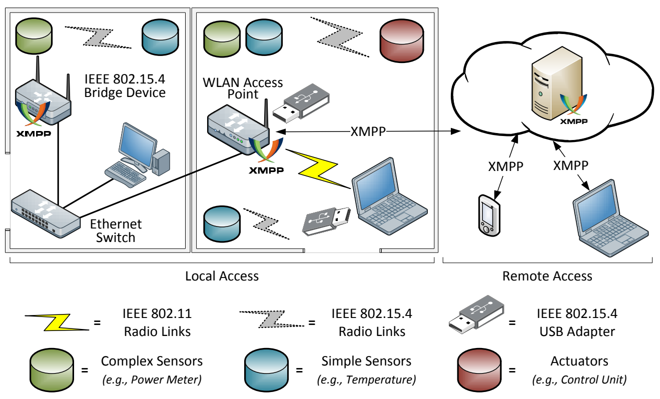
IEEE 802.15.4 Bridge Device
XMPP
WLAN Access Point
XMPP
XMPP
Ethernet Switch
XMPP
XMPP
XMPP
Local Access
Remote Access
= IEEE 802.11 Radio Links
= IEEE 802.15.4 Radio Links
= IEEE 802.15.4 USB Adapter
= Complex Sensors (e.g., Power Meter)
= Simple Sensors (e.g., Temperature)
= Actuators (e.g., Control Unit)

Fig. 3. Test Setup: Smart Home Use Case

A basic anchor point in our testbed is the general purpose access point [36] that provides physical links between different network access technologies of smart objects, ordinary computers, and smart phones. We use commodity router hardware that runs with embedded Linux systems like *OpenWrt*[6] to realize the physical interconnection, because OpenWrt WLAN router are low-priced and widely applicable while also supporting additional software packages. The router runs a *Prosody*[7]

[5]Pidgin - Universal Chat Client [Online] http://www.pidgin.im/

[6]OpenWrt - Wireless Freedom [Online] https://openwrt.org/

[7]Prosody IM [Online] http://prosody.im/

XMPP server and a *Avahi* [37] daemon. Prosody servers are Internet-connected as well as interconnected with each other. All devices can communicate locally and be accessed remotely at the same time. This strengthens the scalability and reliability of the system: it can be extended with local or remotely available XMPP servers while traffic bottlenecks and multi-hop scenarios can be bypassed by a direct integration of smart objects in IP-based networks. The Avahi daemon is responsible for the forwarding of mDNS / DNS-SD messages to the different network interfaces that are interconnected by the router, which enables the discovery of smart objects.

Our smart object prototype is based on the Zolertia Z1[8], which represents a typical resource constrained device with limited memory (92 kB of ROM / 8 kB of RAM), built-in sensors (temperature, accelerometer, battery level) and communication capabilities with an IEEE 802.15.4-compliant RF transceiver and a microUSB connector. For our embedded and memory-constrained hardware, we favor the lightweight and highly portable open source operating system Contiki [38] that runs on small networked sensor boards and provides IP connectivity with its integrated uIP stack [7]. The uIP stack is an embedded IPv4 / IPv6-compatible stack that enables TCP and UDP connections. An event-driven kernel with on-demand preemptive multithreading facilitates communication and memory sharing between all processes while inter-process communication is performed by posting events. All implementations and tests were performed with Contiki 2.5 and corresponding firmware images were built with msp430-gcc (GCC) 4.4.5 for the Zolertia Z1 platform. Each prototype runs our XMPP stack on top of uIP through which it can advertise its current presence, publish measured data, and receive control commands (Chatty Thing). For our XMPP software stack, we measured the memory footprint and successfully tested its protocol behavior against Pidgin and Empathy[9], both well known XMPP chat clients for Linux. The evaluation of *TSP* was performed with the COOJA network simulator (included in Contiki), which allows to execute a native image of real nodes in a simulation [39]. The simulated nodes can run the same native image that we used for our smart object prototype while test results can be easily reproduced with a wide range of network topologies and any number of nodes. COOJA was directly connected via SLIP to our real world testbed.

A. *uXMPP Memory Footprint*

The memory footprint is very important for a lightweight and memory-efficient implementation because smart objects only have limited resources. Additional measurements were made with the firmware images of a TelosB and a Tmote Sky, which use the same compiler and are both based on the MSP430 microcontroller family. The compiled images for each device differ only in the size of the Contiki device driver abstraction of their used hardware components. Binary sizes of the realized modules for our improved uXMPP implementation are listed in Table II.

[8]Zolertia Webpage [Online] http://www.zolertia.com/
[9]Empathy [Online] http://live.gnome.org/Empathy

TABLE II
MEMORY FOOTPRINT OF UXMPP FOR THE MSP430 (IN KBYTE)

Component / Module	ROM	RAM
uXMPP Core / IM	4.17	0.19
uXMPP *XEP-0045* module	1.19	0
uXMPP *TSP* module	0.01	0
uXMPP *XEP-0174*	2.96	0.14
uBonjour (*OWT* enabled)	3.89	0.3
uXMPP Total	12.21	0.63

As the results proof, we were able to reduce the memory consumption of uXMPP while adding new features at the same time. The uXMPP Core/IM is an optimized version of uXMPP v0.1 and provides the same feature set. Its memory consumption could be reduced to a third of its original size and uses only 4.17 kBytes of ROM / 0.19 kBytes of RAM, compared to 12.42 kBytes of ROM / 0.65 kBytes of RAM for version 0.1. The memory use of our *XEP-0174*-based communication stack (including uBonjour) is 6.84 kBytes of ROM / 0.44 kBytes of RAM. We extended uXMPP with a variety of features useful for H2M communication while keeping the memory consumption (12.21 kBytes of ROM / 0.63 kBytes of RAM) comparable to version 0.1 of uXMPP. The current uDPWS implementation[10] uses 10.03 kBytes of ROM / 3.07 kBytes of RAM on a TelosB, but this implementation misses some features (e.g., WS-Eventing capabilities, HTTP Chunked Mode, and HTTP keep alive are not supported). Our uXMPP implementation is therefore very competitive in terms of memory consumption and feature coverage.

B. *TSP Message Optimization*

For this test, we used the system image of the Tmote Sky (MSP430), because it can be used directly in COOJA and it is the only native image with hardware specifications comparable to the Zolertia Z1. The setup consisted of simulation nodes (programmed with uXMPP) and a simulation node programmed with a 6LoWPAN border router running in COOJA. The border router forwarded all packets from the COOJA simulation environment via the Serial Line Internet Protocol (SLIP) to the network of the host computer running our modified XMPP server and vice versa. This enabled a direct interconnection from COOJA to the real world while we were able to extend our testbed with virtual nodes and to use COOJA’s integrated data logger for a detailed analysis of our experiments. As a single border router has scarced resources and needs to share its low bandwitdh with all connected nodes, the test runs were limited to max. 10 nodes booting in parallel to prevent a denial of service and to overcome this network traffic bottleneck of a border router. During each test run every node connected to the XMPP server (logged in, joined the test room, and sent a group chat message). Three different test runs were done: *MUC* (no node ran with *TSP*), *TSP* (all nodes ran

[10]uDPWS - The Devices Profile for Web Services (DPWS) for deeply embedded devices [Online] http://code.google.com/p/udpws/wiki/Introduction

with *TSP*), and *Mixed* (50% of the nodes ran *TSP* while the rest switched *TSP* off).

1) Bootstrap Time: The bootstrap time for nodes with *TSP*-enabled are reduced as Figure 4 shows, because less packets need to be exchanged between the node and the XMPP server.

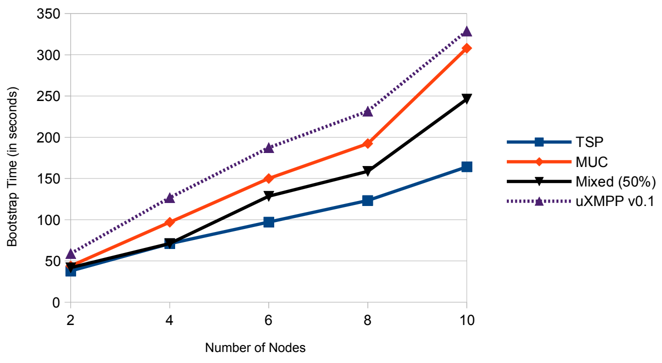

Fig. 4. Bootstrap Time – Comparison of *TSP*, MUC and Mixed

Figure 4 also contains the bootstrap time of uXMPP v0.1 as a reference. In a scenario of 10 nodes, which were connected over one 6LoWPAN border router to the testbed, *TSP* reduces the bootstrap time by 24.86% for a mixed setup and by 54.04% when running on all nodes. The effort for handling more than 10 nodes in parallel on one 6LoWPAN border router rises as our measurements show. In general, the bootstrap time increases with the number of joining nodes, because a lot of packets will be sent for parallel joining nodes over the border router, which needs to serialize these packets to process them. To overcome this bottleneck, joining nodes could perform delayed bootstrapping or the maximum number of assigned nodes for a 6LoWPAN border router could be restricted.

2) Number of Sent Packets during Bootstrap: TSP reduces the network traffic of Chatty Things significantly during bootstrap, as shown by Figure 5. Reference values of uXMPP v0.1 are given to show its higher use of sent messages.

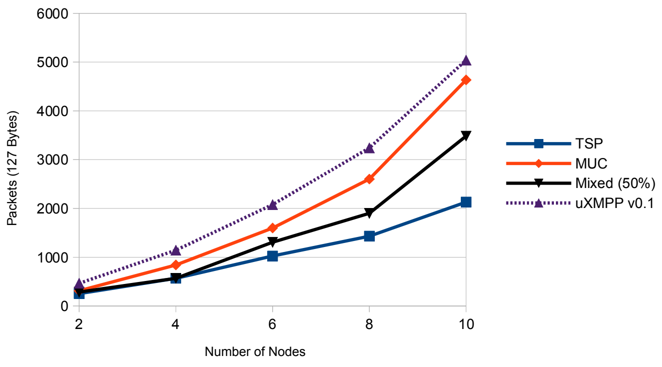

Fig. 5. Network Traffic Comparison of *TSP*, *MUC*, and *Mixed*

As each XMPP login requires a series of messages (at least 9 XMPP messages for *ANONYMOUS*), a large number of packets is exchanged during bootstrapping including ACK messages to the server and vice versa. It is assumed that the network traffic for smart objects with enabled *TSP* will not increase further after this point, because these nodes will not receive any messages related to the topic (joined chat room). In general, we demonstrated that *TSP* can lower the network traffic without the need for XML compression techniques while being standard-compliant to existing XMPP chat clients.

VIII. Final Remarks

Most sensor application protocols use UDP nowadays with additional reliability and sequence number support. But according to [40, Sec. 3.3.2], a TCP reinvention must be omitted since most application layer protocols are based on TCP and need to be redesigned to work with UDP. This would also counter our main idea: using already established protocols and standards to seamlessly integrate smart objects in IP-based networks at the application layer. We explicitly wanted to avoid extensive adaptations of XMPP clients, libraries, and servers. To overcome the performance issue of TCP, approaches like the TCP support for sensor nodes [41] were successfully tested to enable a energy-efficient usage and to increase the data throughput of TCP while caching TCP segments and executing local retransmissions, eventually making TCP applicable for low data-rate networks.

In conclusion, this paper presented a XMPP software stack for smart objects (Chatty Things) running on Contiki OS. We implemented an optimized uXMPP client supporting *XEP-0045 Multi-User Chat* for infrastructure networks and *XEP-0174 Serverless Messaging* with a mDNS / DNS-SD service for ad hoc networks. These implementations can be used to bootstrap smart objects without pre-configured parameters or user interventions for hybrid network environments. Chatty Things can be integrated seamlessly in current infrastructures while inexperienced users can access them through ordinary notebooks or computers with any familiar and available XMPP chat client at the application layer. Smart objects can thus be simply deployed and handled comparable to consumer electronics. We also developed and evaluated a solution to reduce the network traffic for XMPP-driven smart objects called *TSP*. *TSP* uses small presence messages, which fit into one single IP packet, to signalize topic-related status updates of Chatty Things to subscribed XMPP entities in the whole network through temporary presence subscription. It realizes a XMPP-compliant message reduction without using complex and costly techniques like XML compression.

Further work resides in the investigation of timing features for XMPP messages, as defined in [42], to reduce notification frequencies for idling Chatty Things.

References

[1] R.-C. Wang, Y.-C. Chang, and R.-S. Chang, "Design Issues of Semantic Service Discovery for Ubiquitous Computing," in *Proceedings of the International Conference on Multimedia and Ubiquitous Engineering*. IEEE Computer Society, 2007, pp. 880–885.

[2] S. Giordano and D. Puccinelli, "The Human Element as the Key Enabler of Pervasiveness," in *Proceedings of the 10th IFIP Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*. IEEE Computer Society, 2011, pp. 150–156.

[3] J. Bardin, P. Lalanda, and C. Escoffier, “Towards an Automatic Integration of Heterogeneous Services and Devices,” in *Proceedings of the IEEE Asia-Pacific Conference on Services Computing*. IEEE Computer Society, 2010, pp. 171–178.

[4] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas, “Service Oriented Middleware for the Internet of Things: A Perspective,” in *Towards a Service-Based Internet*, ser. Lecture Notes in Computer Science, W. Abramowicz, I. Llorente, M. Surridge, A. Zisman, and J. Vayssire, Eds. Springer Berlin / Heidelberg, 2011, vol. 6994, pp. 220–229.

[5] K. Ashton, “That ’Internet of Things’ Thing,” *RFID Journal*, July 2009, [Online]. Available: http://www.rfidjournal.com/article/view/4986.

[6] F. Mattern and C. Floerkemeier, “From the Internet of Computers to the Internet of Things,” in *From Active Data Management to Event-Based Systems and More*, ser. Lecture Notes in Computer Science, K. Sachs, I. Petrov, and P. Guerrero, Eds. Springer Berlin / Heidelberg, 2010, vol. 6462, pp. 242–259.

[7] M. Durvy, J. Abeille, P. Wetterwald, C. O’Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, “Making Sensor Networks IPv6 Ready,” in *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys)*, Nov. 2008.

[8] P. Saint-Andre, “Extensible Messaging and Presence Protocol (XMPP): Core,” IETF, Request for Comment 6120, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6120.txt

[9] ——, “XEP-0174: Serverless Messaging,” XMPP Standards Foundation, Standards Track, Nov. 2008. [Online]. Available: http://xmpp.org/extensions/xep-0174.html

[10] S. Cheshire and M. Krochmal, “Multicast DNS,” IETF, Internet-Draft, Dec. 2011. [Online]. Available: http://tools.ietf.org/html/draft-cheshire-dnsext-multicastdns-15

[11] ——, “DNS-Based Service Discovery,” IETF, Internet-Draft, Dec. 2011. [Online]. Available: http://tools.ietf.org/html/draft-cheshire-dnsext-dns-sd-11

[12] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, “Constrained Application Protocol (CoAP),” IETF, Internet-Draft, Mar. 2012. [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-coap-09

[13] IBM, “MQ Telemetry Transport,” [Online] http://mqtt.org/, 2012.

[14] M. Isomura, C. Decker, and M. Beigl, “Generic Communication Structure to Integrate Widely Distributed Wireless Sensor Nodes by P2P Technology,” in *Proceedings of the 7th International Conference on Ubiquitous Computing*, Sept. 2005.

[15] J. Schoenwaelder, T. Tsou, and B. Sarikaya, “Protocol Profiles for Constrained Devices,” [Online]. Available: http://www.iab.org/wp-content/IAB-uploads/2011/03/Schoenwaelder.pdf, 2012.

[16] M. Brachmann, O. Garcia-Morchon, and M. Kirsche, “Security for Practical CoAP Applications: Issues and Solution Approaches,” in *Proceedings of the 10th GI/ITG KuVS Fachgespraech Sensornetze (FGSN11)*, Sept. 2011.

[17] M. Hauswirth, D. Pfisterer, and S. Decker, “Making Internet-Connected Objects readily useful,” in *25th Workshop of Interconnecting Smart Objects with Internet*. Prague, Czech Republic: The Internet Architecture Board, March 2011.

[18] E. Zeeb, G. Moritz, D. Timmermann, and F. Golatowski, “WS4D: Toolkits for Networked Embedded Systems Based on the Devices Profile for Web Services,” in *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW 2010)*. IEEE Computer Society, Sept. 2010, pp. 1–8.

[19] A. Broering, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, and R. Lemmens, “New Generation Sensor Web Enablement,” *Sensors*, vol. 11, no. 3, pp. 2652–2699, 2011.

[20] A. Hornsby and E. Bail, “uXMPP: Lightweight Implementation for Low Power Operating System Contiki,” in *Proceedings of the International Conference on Ultra Modern Telecommunications and Workshops (ICUMT 2009)*. IEEE, Oct. 2009, pp. 1–5.

[21] D. Guinard, V. Trifa, and E. Wilde, “Architecting a Mashable Open World Wide Web of Things,” Department of Computer Science, ETH Zurich, Technical Report 663, Feb. 2010.

[22] B. Ostermaier, M. Kovatsch, and S. Santini, “Connecting Things to the Web using Programmable Low-Power WiFi Modules,” in *Proceedings of the 2nd International Workshop on Web of Things (WoT 2011)*. ACM, 2011, pp. 2:1–2:6.

[23] D. Antoniades, I. Polakis, G. Kontaxis, E. Athanasopoulos, S. Ioannidis, E. P. Markatos, and T. Karagiannis, “we.b: The Web of Short URLs,” in *Proceedings of the 20th International Conference on World Wide Web (WWW 2011)*. ACM, 2011, pp. 715–724.

[24] S. Chhabra, A. Aggarwal, F. Benevenuto, and P. Kumaraguru, “Phi.sh/$oCiaL: The Phishing Landscape through Short URLs,” in *Proceedings of the 8th Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS 2011)*. ACM, 2011, pp. 92–101.

[25] F. Klien and M. Strohmaier, “Short Links under Attack: Geographical Analysis of Spam in a URL Shortener Network,” in *Proceedings of the 23rd ACM Conference on Hypertext and Social Media (HHT 2012)*. ACM, 2012, pp. 83–88.

[26] R. Zender, U. Lucke, and D. Tavangarian, “SOA Interoperability for Large-Scale Pervasive Environments,” in *Proceedings of the International Conference on Advanced Information Networking and Applications Workshops*. IEEE Computer Society, 2010, pp. 545–550.

[27] A. Bassi and G. Horn, “Internet of Things in 2020 - Roadmap for the Future,” in *Workshop on RFID / Internet-of-Things*. INFSO D.4 Networked Enterprise & RFID INFSO G.2 Micro & Nanosystems in co-operation with the Working Group RFID of ETP EPOSS, May 2008.

[28] T. Eugster, A. Felber, R. Guerraoui, and A.-M. Kermarrec, “The many Faces of Publish/Subscribe,” *ACM Comput. Surv. 35 (2)*, 2003.

[29] G. Oikonomou and I. Phillips, “Experiences from Porting the Contiki Operating System to a Popular Hardware Platform,” in *Proceedings of the International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*. IEEE Computer Society, Jun. 2011.

[30] R. Klauck and M. Kirsche, “Bonjour Contiki: A Case Study of a DNS-based Discovery Service for the Internet of Things,” in *Proceedings of the 11th International IEEE Conference on Ad-Hoc Networks and Wireless (ADHOC-NOW 2012)*, ser. Lecture Notes in Computer Science (LNCS), X. Li, S. Papavassiliou, and S. Ruehrup, Eds. Springer Berlin, July 2012, vol. 7363, pp. 317–330.

[31] M. Hammoudeh, S. Mount, O. Aldabbas, and M. Stanton, “Clinic: A Service Oriented Approach for Fault Tolerance in Wireless Sensor Networks,” in *Proceedings of the IEEE International Conference on Sensor Technologies and Applications (SENSORCOMM 2010)*. IEEE Computer Society, 2010, pp. 625–631.

[32] D.-K. Chen, “Systematic Review of Applying Service Oriented Architecture in Networking,” in *Proceedings of the International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. IEEE Computer Society, 2010, pp. 167–170.

[33] P. Costa, G. Picco, and S. Rossetto, “Publish-Subscribe on Sensor Networks: a Semi-Probabilistic Approach,” in *Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*. IEEE Computer Society, 2005, pp. 323–332.

[34] P. Saint-Andre, “Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence,” IETF, Request for Comment 6121, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6121.txt

[35] ——, “XEP-0045: Multi-User Chat,” XMPP Standards Foundation, Standards Track, Dec. 2008. [Online]. Available: http://xmpp.org/extensions/xep-0045.html

[36] E. Dressler and D. Tavangarian, “Heterogeneous Communication in Smart Ensembles,” in *Intelligent Interactive Assistance and Mobile Multimedia Computing*, ser. Communications in Computer and Information Science, D. Tavangarian, T. Kirste, D. Timmermann, U. Lucke, and D. Versick, Eds. Springer Berlin, 2009, vol. 53, pp. 155–166.

[37] The Avahi Team, “More About Avahi - Details about mDNS, DS-DNS and Zeroconf,” [Online] http://avahi.org/wiki/AboutAvahi, 2012.

[38] A. Dunkels, B. Gronvall, and T. Voigt, “Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors,” in *Proceedings of the 29th IEEE International Conference on Local Computer Networks (LCN 2004)*. IEEE Computer Society, 2004, pp. 455–462.

[39] T. Voigt, J. Eriksson, F. Österlind, R. Sauter, N. Aschenbruck, P. J. Marrón, V. Reynolds, L. Shu, O. Visser, A. Koubaa, and A. Köpke, “Towards Comparable Simulations of Cooperating Objects and Wireless Sensor Networks,” in *Proceedings of the 4th International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2009)*. ICST, 2009, pp. 77:1–77:10.

[40] G. Moritz, “DPWS for 6LoWPAN,” IETF, Internet-Draft, Jun. 2010. [Online]. Available: http://tools.ietf.org/html/draft-moritz-6lowapp-dpws-enhancements-01

[41] T. Braun, T. Voigt, and A. Dunkels, “TCP Support for Sensor Networks,” in *Proceedings of the 4th Conference on Wireless On demand Network Systems and Services (WONS 2007)*. IEEE/IFIP, Jan. 2007.

[42] G. Chen, “XMPP Pubsub Extension for Long-lived TCP Services,” IETF, Internet-Draft, Jul. 2011. [Online]. Available: http://tools.ietf.org/html/draft-chen-xmpp-pubsub-extension-00