This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE Secon 2010 proceedings.

LTP: An Efficient Web Service Transport Protocol for Resource Constrained Devices

Nils Glombitza, Dennis Pfisterer, and Stefan Fischer
Institute of Telematics, University of Lübeck
Ratzeburger Allee 160, 23538 Lübeck, Germany
Email: {glombitza, pfisterer, fischer}@itm.uni-luebeck.de

***Abstract*—Wireless Sensor Networks (WSNs) are envisioned to become an integral part of the Future Internet. Together with countless other embedded appliances, such resource-constraint devices will form an Internet of Things (IoT) where all kinds of devices extend the Internet to the physical world. In this vision, the seamless and flexible integration of IoT devices ranging from simple sensor nodes to large scale Enterprise IT servers are the basis for novel applications and business processes not possible before. A major challenge is to master the arising challenges of scale, low resources, and heterogeneity. In the Internet and especially in Enterprise IT, heterogeneity is addressed using Service-Oriented Architectures (SOA). However, today's technologies used to realize SOAs are too heavyweight to be used in resource-constraint networks (RCNs). In this paper, we introduce a novel, versatile, and light-weight Web Service transport protocol (called Lean Transport Protocol, LTP) that allows the transparent exchange of Web Service messages between all kinds of resource-constrained devices and server or PC class systems. We describe LTP in detail and show by real-world measurements that LTP has the potential to serve as standard Web Service transport protocol in the Internet of Things.**

I. Introduction

Currently, the Internet is a means for people to request information from servers (Web 1.0) or to contribute information (Web 2.0). Beyond these, more and more research groups work on technologies to build the *Internet of Things* (IoT) where all kinds of devices extend the Internet to the physical world. In this vision, the seamless and flexible integration of IoT devices ranging from simple sensor nodes to large-scale Enterprise IT servers are the basis for novel applications and business processes not possible before. Wireless Sensor Networks (WSNs) are no longer limited to simple "sense and send" applications. Now, they can trigger business processes or actions can be triggered on them by a business process being executed on the Internet. Currently, despite the amount of research targeting middleware solutions for sensor networks and resource constrained devices in general, WSNs are not widely used in industry and are not well integrated into Enterprise IT systems. This is mainly caused by the different characteristics and demands of Enterprise IT systems and WSNs.

In Enterprise IT systems, it is very important to adapt applications and business processes quickly and flexibly in order to react to changing market requirements. Furthermore, the inter-system compatibility of the used middleware technologies is more important than their performance. Today, virtually all Enterprise IT systems are realized as Service-Oriented Architectures (SOAs), which are implemented using standard Internet technologies, such as XML, Web Services, and the Business Process Execution Language (BPEL, [1]). WSNs on the other hand are optimized in terms of low energy consumption as well as low cost and hence CPU power, memory, and bandwidth are severely limited. Hence, these technologies are not applicable in WSNs out-of-the-box. Still, we strongly believe that a successful integration of WSNs is only possible if WSNs are becoming compatible with these technologies rather than Enterprise IT being adapted to special WSN technologies.

The contribution of this paper is a novel transport protocol optimized for resource-constrained devices which we call *Lean Transport Protocol* (LTP). *LTP* allows the transparent exchange of messages between resource-constrained devices and server or PC class systems. It consumes only little memory, CPU power, and features small message sizes. We show how LTP can be combined with existing work on XML compression (or serialization) such that it is possible to fully integrate WSNs into SOAs based on standard Web Service technology.

The remainder of this paper is structured as follows. In Section II, we present fundamentals of Web Services and discuss the gap that prevents its use on extremely resource constrained devices (RCDs). Section III describes our approach's architecture to realize Web Service-based communication between RCDs using *LTP* which is introduced in-depth in Section IV. In Section VI, we present evaluation results of *LTP* which were generated using our prove of concept implementations described in Section V. Section VII concludes this paper.

II. Fundamentals and Related Work

In this section, we first summarize the important fundamentals of Web Services, which are relevant for understanding *LTP*. This includes SOAP [2], WSDL (Web Service Description Language, [3]), and WS-Addressing (Web Service Addressing, [4]). Second, we summarize related work on realizing Web Services in resource-constrained environments for i) XML representation, ii) Web Service transport, and iii) Web Services in resource-constraint environments. Figure 1 depicts the standard Web Service technology stack (contributions of this paper have a black background).

978-1-4244-7151-5/10/$26.00 ©2010 IEEE

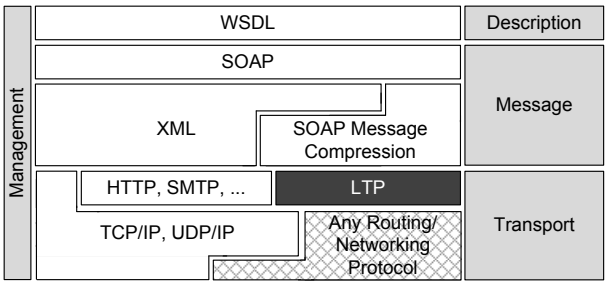

Fig. 1. Web Service Technology Stack (elements in black are contributions of this paper)

SOAP describes the message format of the Web Service communications. Messages are usually serialized using XML and exchanged via HTTP. Apart from mentioning these defaults, the SOAP standard explicitly allows different serialization technologies and transport protocols. Specifications for SMTP, JMS, TCP, and UDP exist.

WSDL is an XML-based language describing the method signatures, transport binding, and (service) endpoint address of a Web Service. The signature describes sequence and layout of the messages while the transport binding specifies how a message is serialized and which transport protocol is used to exchange it. For a specification of the message flow between a consumer and a provider of a Web Service, WSDL (Version 1.1) defines four abstract message exchange patterns: "one-way", "request/response", "notification" as well as "solicit/response".

WS-Addressing specifies addressing mechanisms for Web Service messages going beyond using simple URLs. Using WS-Addressing, the addressing information is encoded within the SOAP message rather than as part of the transport protocol's URL. Thus, it provides a mechanism independent of a particular transport binding (such as HTTP) and provides additional features (such as the use of gateways). If a transport protocol is used that cannot address Web Service endpoints itself (e.g., TCP), the use of WS-Addressing is mandatory. WS-Addressing has four elements representing the addressing information: i) *To* contains the address of the intended receiver. This field is required in every message while the remaining three are optional. ii) *From* provides a reference to the endpoint from which the message originated. iii) *ReplyTo* is the endpoint reference of the intended receiver for replies to this message while iv) *FaultTo* provides the endpoint reference of the receiver for faults related to this message. In addition to the endpoints, the action within this service to be executed must be addressed as well. The *Action* field contains the name of the contained message (i.e., the name of an input, output or fault message as defined in the WSDL). Furthermore, WS-Addressing provides a mechanism to assign single messages to a conversational context (i.e., identify the reply to a request message). The optional *MessageID* field uniquely identifies a SOAP message and the optional *RelatesTo* field defines how two messages relate to each other. It is used to indicate that a message is the answer or the fault message to a previous message.

Efficient XML Representation: Usually, Web Services are exchanged using XML messages resulting in too large messages and therefore a too high demand for storage and processing capabilities to be applicable in resource-constraint environments. There are approaches solving these problems with XML compression/serialization techniques. Werner et al. [5] give an in-depth overview of compact encodings (ITU's standard X.891 "Fast Infoset" [6], XGrind [7], etc.) and introduce a more efficient technique called *Xenia*. *microFibre* [8] is another technique for schema-based XML compression. Other than Xenia, microFibre does not only provide XML compression but additionally provides a direct in-memory data structure representation of the compressed XML documents. Thus, no processing of decompressed XML is required. Compared with plain XML, both approaches achieve compression ratios of up to 98%.

Web Service Transport: Werner et al. analyze the overhead of bindings that use standard Internet protocols. For a simple Web Service operation, they measure an overhead for application layer protocols of 1,096 byte for HTTP, 4,487 byte for SMTP and 3,177 byte for FTP. If one directly uses transport layer protocols, there is still an overhead of 858 byte for TCP and 56 byte for UDP. PURE [9], a UDP-based protocol for resource constrained devices, has an overhead of 66 byte. Since the use of WS-Addressing is mandatory for TCP, UDP, and PURE, additional overhead is generated. Because of their overhead, none of the aforementioned transport protocols are applicable in sensor networks or comparable device classes.

Web Services: Due to the numerous problematic issues, Web Services in WSNs have not been researched very much so far. Amundson et al. [10] and Trifa et al. [11] present a gateway-based approach enabling sensor nodes to call Web Services. Inside the WSN, a proprietary, not Web Service-based middleware is used in both approaches. Based on IPv6 [12], 6LowPAN [13], and HTTP, Priyantha et al. [14] present an approach for Web Services for resource constrained devices. As far as possible, HTTP without SOAP is used for the message exchange. Thus, this approach is limited in terms of the complexity of data structures. Furthermore, the use of HTTP (even without SOAP XML) is too resource demanding for WSNs. For the same reasons, RESTful Web Services as presented in [15] are not applicable in all WSNs.

III. Architecture

As mentioned above, a key challenge is the efficient message exchange between WSN nodes and Internet nodes. In this section, we propose a new Web Service transport binding and a novel transport protocol for resource constrained devices called *LTP* (Lean Transport Protocol). The protocol is not limited to exchanging messages between different network types such between the Internet and RCNs (Network of Resource Constrained Devices) but can also be used within a single network.

Figure 1 depicts the standard Web Service technology stack as presented in Section II with our contribution (*LTP*, Lean

Transport Protocol) highlighted with a black background. *LTP*'s components are: i) a network agnostic addressing scheme, ii) a message transport for UDP/IP, TCP/IP, and resource-constraint environments such as WSNs, iii) support for arbitrary XML serialization/compression schemes, and iv) an efficient packet serialization and fragmentation layer.

LTP is providing a comprehensive addressing schema (*Platform Independent Addressing Schema*) allowing to transparently address Web Service endpoints in RCNs as well as in Enterprise IT systems. *LTP* endpoint addresses are following the regular URL (Uniform Resource Locator, [16]) notation (cf. Figure 4).

The packet format defines the structure of messages exchanged by *LTP*. A packet contains header information including information about the addressed Web Service endpoint, operation, and message as well as about the packet itself to support the same features as WS-Addressing in terms of addressing and message handling. In heterogeneous environments using primarily asynchronous networking protocols, these features are essential to realize all message exchange patterns defined in WSDL. In addition to the header information, an *LTP* packet carries a payload, which is usually a SOAP message (cf. Section IV-A).

A lot of Web Service related data (which are encoded in the header of an *LTP* packet) are long string values. To be applicable in RCNs, *LTP* uses optional header compression techniques (cf. Subsection IV-B) to further reduce the packet sizes apart from using efficient XML serialization. In addition to header compression, *LTP* optionally supports packet fragmentation which is discussed in Subsection IV-D).

The actual serialization of an *LTP* packet uses an arbitrary serialization technique. This can range from plain text-based XML to a highly-efficient compression scheme such as microFibre. Please note that this is not related to the payload of a Web Service message but the serialization of the packet structure itself.

In conjunction with the *Platform Independent Addressing Schema*, the *Platform Independent Messaging* layer realizes a transparent Web Service interaction between i) RCDs within a single RCN, ii) hosts on the Internet (e.g., Enterprise IT servers), iii) RCDs and hosts in the Internet, and iv) RCDs in different (remote) RCNs.

Underneath the *Platform Independent Messaging* layer; *LTP* packets are exchanged using different protocols (e.g., UDP, TCP, E-Mail, HTTP, or a proprietary ones in RCNs). We call the adaptation of a protocol to *LTP* a *connector*. The transition from one network to the other is accomplished using gateways (i.e., message routers) connected to one or more different networks. We currently assume that each gateway is connected to an IP-based network (e.g., the Internet or a private network) to ensure connectivity. Such a gateway has several connectors that perform the routing process using specific routing or networking protocols of the destination network and vice versa. Additionally, features such as header compression and packet fragmentation allow for a further adaption to the characteristics of a specific RCN.

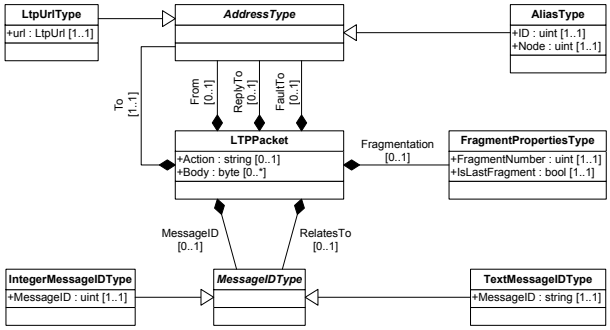

Fig. 2. Structure of an *LTP* Packet

IV. Definition of LTP

In this section we describe *LTP* in detail and discuss the packet format, how XML is serialized, discuss optimizations using header compression, present how packets are fragmented, and finally show how Web Services are described in WSDL such that they use *LTP* and a XML serialization other than plain text.

A. Packet Format and Serialization

Many Internet protocols standardized, e.g., in RFCs define a fixed packet structure. *LTP* avoids such a fixed packet structure and only specifies the information an *LTP* packet has to carry by providing a formal (XML Schema [17]) definition without fixing the type of serialization to use. In this paper, we use the microFibre framework (cf. Section II) in our implementation to serialize *LTP* packets, but *LTP* allows using arbitrary other serializations including plain text. Thus, the serialization can be flexibly adapted to the use case demands.

Figure 2 describes the structure of an *LTP* packet and its elements. To identify, which serialization method is used, every serialized *LTP* packet has a preamble (1 Byte) specifying the actual serialization method.

For maximum compatibility, *LTP*'s structure resembles that of WS-Adressing and therefore defines the necessary subset of its elements. *LTP* defines fields containing addressing information (*To*, *From*, *ReplyTo*, *FaultTo*) and fields describing the semantics of a packet (*Action*, *MessageID* and *RelatesTo*). These fields have the same semantics as in WS-Addressing. Optionally, *LTP* contains *Fragmentation* information as well as the payload itself in the element *Body*.

All fields are optional except for *To*, which identifies the receiver of a message. *LTP*'s addressing fields are of the data type *AddressType* which encodes an endpoint address either as a standard string-based URL or as a more compact *Alias* identifier. Aliases are only used when the optional header compression is enabled. This is discussed later in Subsection IV-B. The optional string-based *Action* field identifies the semantics of the *LTP* message. Usually, it identifies input, output or fault messages within a WSDL port type or an Web Service operation according to its SOAP action as shown in line 6 of Figure 5 (cf. Section IV-E).

MessageID and *RelatesTo* have the same semantics as in WS-Addressing. In addition to WS-Addressing, *LTP* uses the *MessageID* to realize fragmentation (see below). In WS-Addressing, their data type is *string* while in *LTP* they are of data type *MessageIDType*. *MessageIDType* allows encoding the identifier either as a string or as unsigned integers. Again, this is used for optimizations discussed later.

To allow for message fragmentation in networks without this feature (e.g., RCNs such as WSNs), the optional *Fragmentation* is used. It is of data type *FragmentPropertiesType* with the fields *FragmentNumber* and *IsLastFragment*. For details, please see Subsection IV-D.

The last field of an *LTP* packet is the optional *Body* field holding the payload. As *LTP* is intended for exchanging SOAP Web Service messages, the *Body* field carries the serialized SOAP message. The format of the payload (e.g., plain-text XML or an arbitrary compression technique) is defined in the WSDL of the Web Service (cf. Section IV-E).

B. Header Compression

With *Action*, *MessageID*, and *RelatesTo* as well as URL compression, *LTP* provides several (optional) header compression mechanisms to minimize resource consumption and required bandwidth share. All compression mechanisms can be used in all subnets, but are recommended especially in RCNs.

To compress addressing information, URLs are replaced (e.g., at a gateway) with a shorter value stored in the *Alias* field. As shown in Figure 2, *Alias* contains the fields *ID* and *Node*. *ID* is a local unique identifier representing the endpoint. *Node* carries the networking address of the endpoint which is addressed by the URL. If the replaced URL addresses an endpoint outside the local RCN, the network address of the Connector gateway's link to the local RCN is stored in *Node*. The tuple of ID and Node together uniquely identify a URL. However, every URL can be mapped to different Alias entries. The mapping between Node and the networking address enables a direct routing of *LTP* packets in RCNs with no need to resolve mappings between an Alias and the related URL. Every RCD sending an *LTP* packet can replace the contained URLs with the corresponding Aliases. Since Aliases are only valid inside single subnets, the gateway has to convert Aliases of incoming packets to the correlating URLs and optionally recompress these URLs using the mapping of the next-hop's subnet.

Since the compression of URLs is realized by replacing it with an Alias, the mapping between Alias and URL has to be resolved. A straightforward, easy solution is a static configuration where mapping tables are simply stored on each RCD. Since such an approach easily overburdens the memory resources of most RCDs, alternative ways are required. The approach presented in this paper uses a service offered by a gateway that performs this mapping. This so-called *Infrastructure Web Service* (IWS) is a standard Web Service running on a gateway, which can be addressed and invoked using *LTP*. As this service is offered within the same network, no URLs are required for addressing and only short aliases may be used. How the service is discovered inside a network is beyond the scope of this paper.

Figure 3 describes the interaction of *LTP* nodes and an IWS instance. Each node can register mappings at the IWS. To resolve mappings, a node requests it from the IWS via a Web Service call. To reduce the communication overhead, a certain number of mappings are cached at each node. Thus, the IWS is only invoked if the mapping was not found in the node's cache. Optionally, in addition to this resolution strategy following the "pull"-principle, the IWS provide to"push" mappings periodically into a RCN as well. This strategy is recommended if a lot of nodes have to resolve the same mappings, since the according Web Service messages of the IWS are sent to a broadcast/multicast address.

Holding the destination node's id, the *Alias* already contains routing information and the URL itself is typically not required. URLs must only be resolved in three cases: i) Packet forwarding at a gateway. This is no problem since both, the gateway and the IWS are typically running on PCs or even the same PC. ii) If the message to be sent is not an answer to a previously received message. Then, a node must resolve the alias for a URL. iii) Each node has to know the mapping of the service endpoints which it is providing.

In addition to the URL compression, *MessageID* and *RelatesTo* can be compressed as well. In the Web Service context a message ID is typically an arbitrary string. Thus, strings can be used as default with *LTP* as well. But if using an integer value instead, the message ID can be serialized much more efficiently.

Since a lot of serialization methods of SOAP messages also encode the *Action* field, a Web Service can read that information from the body of an *LTP* message. If *LTP* is used with such a serialization method, such as microFibre, the *Action* field can just be omitted.

C. Message Exchange

LTP provides a transparent end-to-end message exchange between Web Service endpoints in different types of networks. This is realized by a *Platform Independent Messaging* scheme that uses a *Platform Independent Addressing Scheme*.

So-called connectors at gateways adapt *LTP* to a specific underlying network. These gateways build self-contained subnets realizing an internal end-to-end communication. The Platform Independent Messaging layer defines a transparent end-to-end message exchange between endpoints in arbitrary subnets. Messages inside a single subnet are exchanged using its particular networking protocol. Packets crossing subnet boundaries are routed via Connector gateways of the according subnets. As mentioned above, we require that a gateway is connected to an IP-based network such as the Internet. As a result, messages are only passed between nodes inside a single subnet, between nodes inside a subnet and the gateway, or between gateways.

To address arbitrary endpoints, *LTP* uses the Platform Independent Addressing Scheme. As shown in Figure 4, it follows the URL notation with "ltp" as protocol prefix. The host and

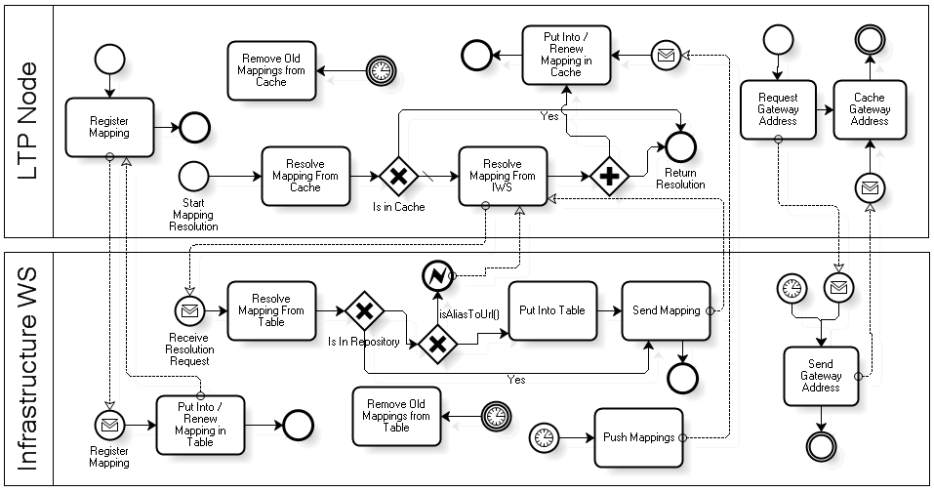

Fig. 3. Interactions between the Infrastructure Web Service and *LTP* nodes

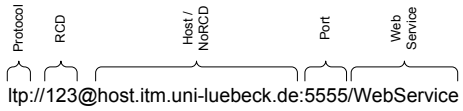

Fig. 4. URL of an *LTP* Web Service Endpoint

port part of an *LTP* URL describe the message routing in the IP-based network. Adding the optional "@" to the endpoint URL, an endpoint inside a RCN is addressed. In that case, the host and port tuple is addressing the Connector gateway of the addressed endpoint, while an arbitrary string in front of the "@" is addressing the endpoint inside the RCN. As in any other URL based Web Service endpoint, the path addresses a specific Web Service on that host since multiple services may be deployed in one application.

Algorithm 1 describes the routing rules at a gateway. Packets coming from the Internet, which are flowing to a RCN, are forwarded to the RCD by the gateway. The destination address is either forwarded to the node encoded in the endpoint URL or (if URL compression is used) to the node encoded in the Alias field. Packets leaving a RCN are forwarded using UDP or TCP. If the packet to forward has a compressed destination address, the Alias has to be replaced by the corresponding URL which is resolved from a cache or the Infrastructure Web Service.

Algorithm 1 Routing of an LTP Packet at the Gateway

Input: Let p be the LTP packet to route and t_{Alias} respectively t_{URL} the Alias and URL value of the *To* field of p.

1: **if** p was received from the Internet **then**
2: let d be a MAC or network address used in the connected RCN n
3: $d := \begin{cases} node(t_{\text{Alias}}) & isCompressed(t) \\ node(t_{\text{URL}}) & \neg isCompressed(t) \end{cases}$
4: send p to d using the Routing/Networking Protocol of n
5: **else if** p was received from RCN **then**
6: **if** $isCompressed(t)$ **then**
7: $t_{\text{URL}} := \begin{cases} getUrlFromCache(t_{\text{Alias}}) & isUrlInCache(t_{\text{Alias}}) \\ getUrlFromIWS(t_{\text{Alias}}) & else \end{cases}$
8: **end if**
9: send p to $(host(t_{\text{URL}}), port(t_{\text{URL}}))$ using TCP or UDP
10: **end if**

Algorithm 2 describes the routing of an *LTP* packet on a RCD. Packets for destinations inside the local subnet are routed directly to the destination node while packets with a destination outside the local subnet are routed to the gateway. To perform this forwarding, the node must obtain the gateway's host h_{G} and port p_{G} to decide if a node inside the local subnet is addressed. In a static scenario, this information can be statically configured on the RCDs. Furthermore, for dynamic scenarios, this information can be requested from the InfrastructureWS (cf. Figure 3).

The Alias to URL mapping shown in line 7 of Algorithm 1 is actually not part of the routing, but of the header compression. However, we included it in Algorithm 1 to show that in this particular case, no routing decision can be made based on the Alias without knowing the URL. For all other routing decisions (cf. Algorithms 1 and 2), packet routing inside a RCN is achieved without knowing the URL, since an Alias always contains the networking address of the addressed endpoint.

So far, we described how *LTP* exchanges asynchronous packets between endpoints in different subnets. Additionally, *LTP* provides synchronous communication as well. By setting the *From*, *ReplyTo* or *FaultTo* fields in the request packet, the receivers of regular and error responses can be specified. With the *MessageID* field of the request as well as *RelatesTo* field of the response packet, *LTP* builds a relation between request and response packet.

D. Fragmentation

LTP packets which exceed the maximum packet size s are fragmented. The *Body* of an *LTP* packet p is split up and put into several single *LTP* packets p_i which are exchanged separately. To indicate that p_i are fragments of p, the values of the *Fragmentation* field are set in each p_i. *FragmentNumber* indicates the index of a fragment while *IsLastFragment*

Algorithm 2 Routing of an LTP Packet on a RCD

Input: Let t be the *To* field of the LTP packet to route. Let t_{Alias} and t_{URL} be the respective Alias and URL values of t. Let h_{G} be the host, p_{G} the UDP or TCP port and n_{G} the MAC or networking address of the gateway's link to the local RCN.
Output: d as next hop destination of p in the local RCN.
1: $\langle isCompressed(t)\rangle \longrightarrow \langle d := t_{\text{Alias}}\rangle$
2: $\langle \neg isCompressed(t) \wedge host(t_{\text{URL}}) = h_{\text{G}} \wedge port(t_{\text{URL}}) = p_{\text{G}} \wedge node(t_{\text{URL}}) = N/A\rangle \longrightarrow \langle d := N/A\rangle$
3: $\langle \neg isCompressed(t) \wedge host(t_{\text{URL}}) = h_{\text{G}} \wedge port(t_{\text{URL}}) = p_{\text{G}} \wedge node(t_{\text{URL}}) \neq N/A\rangle \longrightarrow \langle d := node(t_{\text{URL}})\rangle$
4: $\langle \neg isCompressed(t) \wedge (host(t_{\text{URL}}) \neq h_{\text{G}} \vee port(t_{\text{URL}}) \neq p_{\text{G}})\rangle \longrightarrow \langle d := n_{\text{G}}\rangle$

indicates the fragment with the highest *FragmentNumber*. *MessageID* relates each p_i to p. While *LTP* handles unsorted fragments, it does not request for resending fragments if they got lost. All received fragments are rejected if fragments are still missing after a timer expires.

LTP realizes packet fragmentation individually in local subnets. Every RCN defines its own local maximum packet size s_{local}. Thus, fragmented packets, which are crossing subnet borders, are defragmented at the corresponding gateway and reassembled according to the maximum packet size of next-hop's subnet s_{next}. Since reassembling packets at intermediary hops is only necessary if $s_{\text{local}} > s_{\text{next}}$ this process happens more often than necessary. However, it is not expensive since it is done at gateways which are hosted on PCs. The advantage is that different fragmentation strategies can be defined in each subnet resulting in optimal performance.

E. Web Service Description

Our transport binding for resource constrained devices using *LTP* and compressed SOAP is 100% compliant to the SOAP standard. Thus, WSDL can be used without any adaptations to describe Web Services using our transport binding. Figure 5 shows an extract of an example WSDL describing a microFibre compressed SOAP over *LTP* Web Service. For the concrete binding to our transport binding, one has to define that the Web Service uses *LTP* as transport protocol and that the SOAP message is compressed using microFibre. This is done using the *binding* tag of the SOAP 1.2 namespace (cf. lines 3-4). Furthermore, if the Web Service should be bind to a concrete endpoint as well, its location can be set with the *address* tag of the SOAP 1.2 namespace (cf. lines 11-12).

Since *LTP* supports asynchronous as well as synchronous communication, it supports all four messages exchange patterns defined in WSDL 1.1 out of the box. No use of additional Web Service standards such as WS-Addressing are required. microFibre compressed SOAP over *LTP* indentifies Web Service operations by their *Action* field. Therefore, for each operation, the *soapAction* attribute of the SOAP 1.2 namespace has to be defined (cf. line 6). Alternatively, every input, output or fault message can be attributed with a WS-Addressing action.

```
1   [...]
2   <wsdl:binding [...]>
3     <soap12:binding transport="http://www.itm.uni-luebeck.de/users/ ⬅
4           glombitza/ltp" style="document" compression="microFibre"/>
5    <wsdl:operation name="exchTemp">
6      <soap12:operation soapAction="urn:exchTemp" style="document"/> [...]
7    </wsdl:operation>
8   </wsdl:binding>
9   <wsdl:service [...]>
10   <wsdl:port [...]>
11     <soap12:address
12           location="ltp://localhost:8080/services/TemperatureService/"/>
13   </wsdl:port>
14  </wsdl:service> [...]
```

Fig. 5. Excerpt from the WSDL defining the "Temperature" Web Service using the microFibre-compressed SOAP over *LTP* binding

V. Implementation

To prove our concept, we implemented *LTP* in four different flavors. First, we implemented a standalone Java-based version of *LTP* intended for PC and server class systems. This implementation is only capable of handling *LTP*-based Web Services but provides a high performance compared to full-blown Web Service frameworks.

Second, we have extended the Web Service framework Apache Axis2 [18] to include *LTP* as an additional transport protocol/binding. This allows invoking the same Web Service using standard transport protocols (such as HTTP, SMTP, or FTP) and additionally using *LTP*.

Third, we have implemented *LTP* for different types of sensor node and embedded hardware platforms. Our implementation is using C++ as programming language. It can be compiled for every platform that provides a decent C++ compiler such as the GCC [19] compiler. We have successfully tested this implementation on the *iSense/Jennic* [20], *TelosB* [21], and *Pacemate* sensor network platforms as well as on PDAs running *Windows Mobile*. Since every feature of *LTP* induces costs in terms of additional code size and slower run-time performance, our implementation is very modular. Developers can freely enable or disable debug information or the *InfrastructureWS* as well as one of the following four configurations:

- *Minimal:* provides only the minimal required functionality to realize *LTP* communication.
- *Callback Handler:* puts request and response messages automatically into relation and invoke callback handlers that register for certain messages.
- *Fragmentation:* enables the automatic (de)fragmentation of *LTP* packets.
- *Callback Handler & Fragmentation:* enables both, *Callback Handler* and *Fragmentation*.

Fourth, we have implemented a gateway connecting the above-mentioned RCNs with the Internet.

VI. Evaluation

In this section we present the results of our evaluation. Many benefits of using *LTP* to enable Web Services in RCNs are non-measurable, qualitative characteristics such as increased

TABLE I
CODE SIZES OF *LTP* (IN BYTE)

Platform	Mini-mal	CBH	Frag.	CBH& Frag.
Jennic	2,476	4,300	6,864	8,688
Pacemate	2,888	5,100	7,992	10,204
Win. Mobile (ARM)	2,664	5,364	8,352	11,052
TelosB	680	1,362	2,524	3,206

Platform	microFibre fraction	IWS (incl. microFibre)
Jennic	5,172	7,077
Pacemate	4,164	5,848
Win. Mobile (ARM)	4,172	6,360
TelosB	4,226	5,724

development speed, a fast, easy and flexible adaptation of applications and business processes as well as a seamless integration of Enterprise IT systems and networks of resource constrained devices. In this paper, we focus on measurable properties and present a quantitative evaluation of our implementations. In the following, we evaluate code and message sizes as well as processing performance and memory footprint.

A. Static Code Size

Table I shows the code sizes of our *LTP* implementation for RCDs. We compiled the C++ code for the Jennic (32 bit RISC controller), Pacemate (32 bit ARM controller), and TelosB (16 bit MSP430 controller) WSN hardware platforms as well as for the Windows Mobile (32 bit ARM controller) target. For every platform we measured the code size using the minimal configuration, the configuration with call-back handlers (called CBH in the following), and the configuration with fragmentation support (called Frag. in the following).

With a minimum code size of $2,476$ KB (Jennic), $2,888$ KB (Pacemate), 680 KB (TelosB), and $2,664$ KB (Win. Mobile) and maximum code size of $8,688$ (Jennic), $10,204$ KB (Pacemate), $3,206$ KB (TelosB), and $11,052$ KB (Win. Mobile), *LTP* itself uses only few flash memory. In addition to *LTP*, the serialization/compression code requires some resources. As discussed above, we use microFibre [8], a highly efficient serializer. In our case, this adds between 4.2 KB (Win.Mobile) and 5.1 KB (Jennic) of code size.

The Infrastructure Web Service (including its serialization code) requires between 5.7 KB (TelosB) and 7.1 KB (Jennic) flash memory. The results show that even with all functionality turned on, *LTP* can be used even on extremely resource constrained devices such as sensor nodes without problems in terms of the flash memory usage.

B. RAM Requirements

Next to the static flash memory usage, we evaluated the RAM memory during the execution of *LTP* on the iSense (32 bit) platform. Table II summarizes these results.

In the minimal setup, we measured the memory while processing a sequence of one incoming and one outgoing *LTP* packet. In the fragmentation scenario, the incoming and outgoing packets were split into three fragments each. In the callback handler scenario, we sent one packet and received the answer packet. We did all measurements with and without *LTP* header compression.

TABLE II
MEMORY USAGE OF PROCESSING LTP PACKETS (IN BYTE)

Scenario		Min	$\mathbf{Q}_{.25}$	Median	$\mathbf{Q}_{.75}$	Max
Minimal	uncomp.	40	1203	1215	1263	1283
	comp.	40	443	455	503	523
Fragmen-tation	uncomp.	52	1363	1427	1481	1510
	comp.	52	547	607	717	741
Callback handler	uncomp.	56	1295	1307	1333	1353
	comp.	56	475	487	513	533

The minimum memory usage after initialization is between $40\,\mathrm{B}$ and $56\,\mathrm{B}$ with or without *LTP* header enabled. The fragmentation mode's max memory usage of $1,510\,\mathrm{B}$ in the uncompressed and $741\,\mathrm{B}$ in the compressed mode is higher than in the other scenarios. This is also true for the 25%, 50% and the 75% quartile. This fact was not unexpected since information has to be cached to put fragments together.

However, the small difference compared with the minimal setup ($212\,\mathrm{B}$ for the median in uncompressed mode) is not obvious. If comparing the uncompressed and compressed mode of every scenario, the compressed mode generates a significantly lower memory usage (fragmentation median: $1,427$/607). This fact is caused by implementation characteristics of microFibre which is in the current version not using dynamic memory allocation. Thus, for the in-memory representation of the *LTP* packet, memory is allocated even for not used fields. With compression enabled, no memory consuming strings are used but even the memory consumption values of the compression enabled scenarios contain a static $255\,\mathrm{B}$ buffer for the SOAP body. Thus, the memory consumption could extremely be reduced by using dynamic memory allocation.

$$s = 2 \cdot |e - 8| + |m - 4| + a + b + 2 \cdot e + m + 47 \qquad (1)$$

Equation 1 shows the large influence of the static memory allocation of microFibre in our evaluation and describes the theoretic (without alignment) size s of an in-memory *LTP* packet representation. s depends on the maximum endpoint reference length e, the maximum message/relates to ID length m, the maximum action length a, and the maximum body length b.

C. Message Size

A third important requirement of realizing Web Services in WSNs is the message size of the transport protocol as well as of the SOAP message. The message size of *LTP* packets are strongly influenced by the length of the string elements and its Huffman encoding. In this evaluation the request message carried 89 characters of data (To: 34, ReplyTo: 38, MessageID: 4, Action: 13) while the response message carried 63 characters (To: 38, RelatesTo: 4, Action: 21). The overall

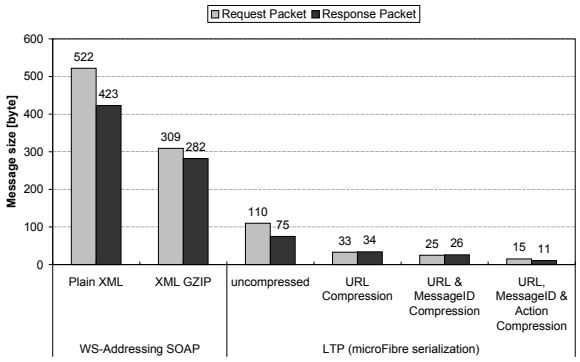

Fig. 6. Comparison of packet sizes

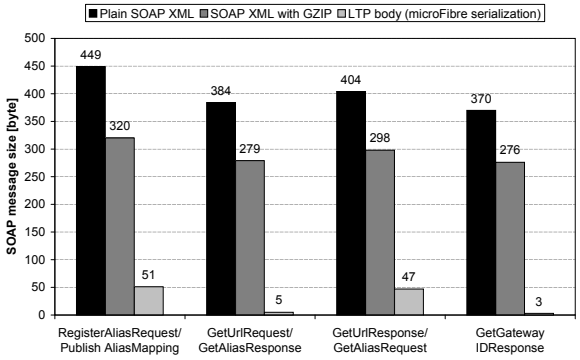

Fig. 7. SOAP message sizes of the of the *InfrastructureWS*

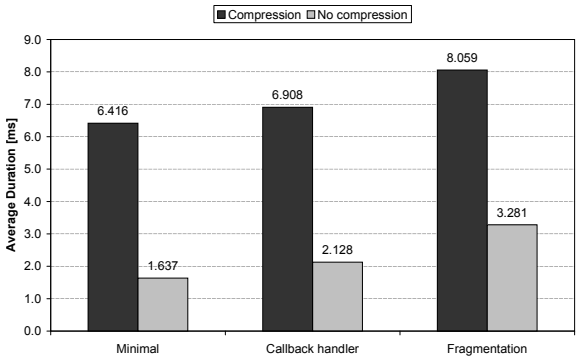

Fig. 8. Processing duration (roundtrip) of an *LTP* request- & response-packet

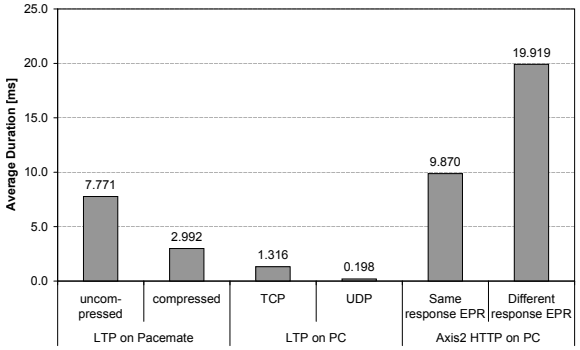

Fig. 9. Roundtrip times for a Web Service invocation

size s of *LTP* packets transferring a *Body* is defined by Equation 3 and 2. Let b be the size of the *Body* to transfer, h be the size of an *LTP* header without fragmentation information, and m the maximum size of one *LTP* packet (fragment) as well as n the number of fragments needed to transfer b. To encode *Fragmentation*, additional 9 bit are serialized which generates $f \in \{1, 2\}$ bytes in an *LTP* packet.

$$n = \begin{cases} 1 & h + b \leq m \\ \left\lceil -\frac{b}{f+h-m} \right\rceil & else \end{cases} \quad (2)$$

$$s = b + h \cdot n + \begin{cases} f \cdot n & n > 1 \\ 0 & else \end{cases} \quad (3)$$

Figure 6 shows the message size for the different types discussed above. Here, Plain XML represents the messages size of a normal Web Service using WS-Addressing. XML GZIP serves as a benchmark showing what is achievable with normal compression techniques. The packets are not fragmented and do not contain a *Body*.

With 423 B (plain XML) and 282 B (GZIP compressed), WS-Addressing is too resource demanding for WSNs. Next to the bad compression rate of 33%, GZIP compressed messages still have to be decompressed and processed on sensor nodes. With 75 B, *LTP* has a compression rate of 82% in its uncompressed mode. Using all compression modes, the compression rate of *LTP* is 97% compared to WS-Addressing and 85% compared with uncompressed *LTP*.

In addition to the message sizes of *LTP* itself, we measured the message sizes of the SOAP messages of the Infrastructure Web Service which uses a microFibre compressed SOAP message. The results are shown in Figure 7. It compares the message sizes of *LTP* with the ones generated by plain XML and GZIP compressed. A compression ratio between 88% and 99% is achievable.

D. Processing Performance and Roundtrip Duration

Finally, the processing performance of *LTP* is an important characteristic. We used the Pacemate platform with its 60 MHz CPU and measured the average processing duration for 10,000 iterations receiving and sending the same packets as for the corresponding RAM evaluation. Figure 8 shows the results. The average durations of processing the compressed packets are significantly lower than the corresponding uncompressed measurements (74% for the minimal scenario). This is due to the fact that no strings with the expensive Huffman encoding need to be processed. Thus, next to lower message sizes, the compression techniques also improve the processing performance. With processing times between 1.6 ms and 3.3 ms respectively 6.4 ms and 8.1 ms for the uncompressed scenario, the performance of *LTP* is very good on resource constrained sensor nodes.

In addition, we evaluated the roundtrip durations of calling a simple Web Service with *LTP* in conjunction with a microFibre based SOAP serialization. With 1.3 ms using TCP and 0.2 ms using UDP as underlay network, compressed SOAP over *LTP* is performing way better on a PC than SOAP over HTTP with

an average of 9.9 ms using the same connection for request and response. Since *LTP* provides the ability to send responses to different endpoints than the request originated, we evaluated SOAP over HTTP using an different connection for the response as well (19.9 ms). With 7.8 ms for the uncompressed and 3.0 ms for the compressed case, *LTP* on the Pacemate platform performs even better than SOAP over HTTP on a PC. This is mainly caused by a more efficient and performing message serialization and partly caused by the fact that no TCP connection needs to be established on the sensor node.

VII. Conclusion & Future Work

In this paper, we introduced LTP, a novel, versatile and light-weight Web Service transport protocol that allows the transparent exchange of Web Service messages between all kinds of resource-constrained devices and server or PC class systems. Its main features are its platform-independence, low resource consumption and its implementation-agnostic definition of the protocol. This allows coming up with different implementations using different representations of LTP packets, e.g., one using plain XML or another one using sophisticated compressions techniques such as microFibre. Our evaluation shows that we achieve a compression ratio of 97.4% compared to a standard WS-Addressing packet in conjunction with an extremely low processing overhead compared to XML-based variants (only 0.95% on a PC and 14.6% on a sensor node). Our implementation of LTP runs on a variety of hardware platforms and only consumes 7kB of static code memory and a maximum of 1.5kB of RAM memory. Future work will include realizing self-describing Web Services on RCDs as well as improving the integration of RCDs into business processes. To realize both research goals, *LTP* is will be used as enabling technology.

References

[1] WS-BPEL Technical Committee, "Webservices – Business Process Execution Language Version 2.0," OASIS (Organization for the Advancement of Structured Information Standards), Tech. Rep., 2005. [Online]. Available: http://www.oasis-open.org/committees/wsbpel

[2] XML Protocol Working Group, "SOAP Specifications," World Wide Web Consortium (W3C), Tech. Rep., 2007. [Online]. Available: http://www.w3.org/TR/SOAP/

[3] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," World Wide Web Consortium W3C, Tech. Rep., 2001. [Online]. Available: http://www.w3.org/TR/wsdl.html

[4] Web Services Addressing Working Group, "Web Services Addressing 1.0," World Wide Web Consortium (W3C), Tech. Rep., 2006. [Online]. Available: http://www.w3.org/2002/ws/addr/

[5] C. Werner, C. Buschmann, and S. Fischer, "Advanced data compression techniques for SOAP web services," in *Modern Technologies in Web Services Research*, L.-J. Zhang, Ed. IGI Publishing, 2007, pp. 76–97.

[6] International Telecommunication Union (ITU), "Recommendation X.891: Generic applications of ASN.1 – FastInfoset," May 2005.

[7] P. M. Tolani and J. R. Haritsa, "XGRIND: A query-friendly XML compressor," in *Proceedings of the 18th International Conference on Data Engineering, 26 February – 1 March 2002, San Jose, CA*. IEEE Computer Society, 2002, pp. 225–234. [Online]. Available: http://csdl.computer.org/comp/proceedings/icde/2002/1531/00/15310225abs.htm

[8] D. Pfisterer, M. Wegner, H. Hellbrück, C. Werner, and S. Fischer, "Energy-optimized Data Serialization For Heterogeneous WSNs Using Middleware Synthesis," in *Proceedings of The Sixth Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net' 07)*, Jun. 2007, pp. 180–187.

[9] C. Werner, C. Buschmann, and T. Jäcker, "Enhanced Transport Bindings for Efficient SOAP Messaging," *IEEE International Conference on Web Services*, pp. 193–200, 2005.

[10] I. Amundson, M. Kushwaha, X. Koutsoukos, S. Neema, and J. Sztipanovits, "Efficient integration of web services in ambient-aware sensor network applications," in *3rd IEEE/CreateNet International Workshop on Broadband Advanced Sensor Networks (BaseNets 2006)*, 2006. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/284.html

[11] V. Trifa, S. Wieland, DominiqueGuinard, and T. M. Bohnert, "Design and Implementation of a Gateway for Web-based Interaction and Management of Embedded Devices," in *Proceedings of the 5th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS Workshop)*, 2009.

[12] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460 (Draft Standard), Dec. 1998, updated by RFC 5095. [Online]. Available: http://www.ietf.org/rfc/rfc2460.txt

[13] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944 (Proposed Standard), Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4944.txt

[14] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: Design and implementation on interoperable and evolvable sensor networks," in *SenSys'08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, 2008. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=73067

[15] D. Yazar and A. Dunkels, "Efficient Application Integration in IP-based Sensor Networks," in *Proceedings of ACM BuildSys 2009, the First ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings*, Berkeley, CA, USA, Nov. 2009. [Online]. Available: http://www.sics.se/~adam/yazar09efficient.pdf

[16] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," RFC 1738 (Proposed Standard), Dec. 1994, obsoleted by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986. [Online]. Available: http://www.ietf.org/rfc/rfc1738.txt

[17] World Wide Web Consortium (W3C), "Recommendation: XML Schema," Oct. 2004. [Online]. Available: http://www.w3.org/XML/Schema

[18] Apache Software Foundation, "Apache Axis2," 2009. [Online]. Available: http://ws.apache.org/axis2/

[19] Free Software Foundation, Inc., "Gnu Compiler Collection (GCC)," 1984. [Online]. Available: http://gcc.gnu.org/

[20] C. Buschmann and D. Pfisterer, "iSense: A Modular Hardware and Software Platform for Wireless Sensor Networks," 6. Fachgespräch Drahtlose Sensornetze der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme, Tech. Rep., 2007. [Online]. Available: http://ds.informatik.rwth-aachen.de/events/fgsn07/fgsn07proc.pdf

[21] Crossbow Technology Inc., "TELOSB - TelosB Mote Platform," 2009. [Online]. Available: http://www.xbow.com/