LWESP:Light-Weight Exterior Sensornet Protocol*

Ángel Cuevas, Manuel Urueña
University Carlos III of Madrid
Avenida de la Universidad, 30
28911. Leganés -Madrid- (Spain)
{acrumin,muruenya}@it.uc3m.es

Annett Laube, Laurent Gomez
SAP Labs France, Sophia Antipolis
805, Avenue du Docteur Maurice Donat
BP 1216 - 06254 Mougins (France)
{annett.laube,laurent.gomez}@sap.com

Abstract

Access to Wireless Sensor Networks (WSNs) from external applications will become a key aspect in order to successfully deploy this technology. This paper presents the Light-Weight Exterior Sensornet Protocol (LWESP), to enable the communication between applications and WSNs. This protocol is a combination of the Sensor Communication Language (SENCOMLNG) and the eXtensible Binary Enconding (XBE32). SENCOMLNG is an XML-based language. XBE32 provides an efficient encoding of the SENCOMLNG in order to reduce the bandwidth utilization and the processing cost in the WSN gateways, that in many scenarios could also be resource-restricted devices. Finally, we present a test-bed based on JAVA as well as a comparison between our proposal and other solutions showing that LWESP outperforms all the previous proposals in terms of bandwidth utilization.

1. Introduction

Wireless Sensor Networks (WSNs) have been told to become one of the key technologies for the near future. They have focused the interest of both the academia and the industry due to the wide set of novel applications that can be developed using this technology. Some of the most interesting fields where WSN can be used are: remote health control, military, inter-vehicle communication, building automation, herd control, etc.

In the last five years much research work has explored the communication stack, from the physical to the application layer, as well as cross-layer approaches in order to provide more efficient solutions [5].

It is time to check whether WSN will be the successful commercial technology that has been promised. A key feature to achieve this goal is to enable external applications to easily access sensor data. In the literature different approaches are presented to allow external applications to access sensor data [1, 3, 4, 6, 8, 9]. In particular, WSNs will become a real success when business applications (BAs) do massively use sensor data to streamline business processes.

In this paper we propose a simple XML-based language called SENsor COMmunication LaNGuage (SENCOMLNG) to communicate applications and WSN Gateways (WSN GWs). This language defines 4 basic operations for the applications to access sensors data: *getData*, *setData*, *monitoring* and *trap*. The first and second ones are used to retrieve or set a single value from/to a sensor/actor. The *monitor* operation is used for obtaining periodical measurements from a particular sensor. Finally, the *trap* operation establishes a threshold value on a sensor, which propagates a notification to the application when the threshold is surpassed. With the combination of these four basic operations, an application is able to generate semantic information out of the raw sensor data retrieved.

Since there are many different WSN platforms, the WSN-GW is the responsible for translating the SENCOMLNG messages into the particular WSN platform messages.

Other solutions presented in the literature (using UPnP, WebServices, XML, etc.) follow a similar approach. However, they do not take into account the big amount of bandwidth and processing resources required for these technologies, and in many scenarios the WSN-GW will be a resource constrained device. For instance a PDA acting as a WSN-GW for a Body Sensor Network that monitors elder people health.

Having this in mind, we propose a further step in our solution in order to reduce bandwidth consumption and processing complexity. By codifying the SENCOMLNG messages using the eXtensible Binary Encoding (XBE32) [10] we are able to reduce the bandwidth utilization from a 30% up to 80% comparing to other solutions, while keeping its great flexibility, so that any modification in SENCOMLNG would imply little or no modifications to the XBE32 imple-

*This work was supported by the Spanish government through the Project T2C2 TIN2008-06739-C04-01.

mentation. Therefore, using XBE32 does not reduce XML flexibility and, at the same time, yields a dramatic reduction in the bandwidth utilization. Moreover, it is well suited for processing tasks in limited devices, since XBE32 carries binary data (short, int, long, etc.) to process the messages while XML requires a lot of text parsing.

The protocol obtained when SENCOMLNG messages are encoded using XBE32 is the main contribution of this paper. This protocol is called Light-Weight Exterior Sensornet Protocol (LWESP).

This paper is organized as follows: Section 2 describes SENCOMLNG in detail. XBE32 is introduced in Section 3. In Section 4, LWESP is defined as the combination of SENCOMLNG and XBE32. Next, related works are described in Section 5. The evaluation and comparison of LWESP in front of other approaches are presented in Section 6. Finally, Section 7 concludes the paper and indicates future works.

2. Sensor Communication Language (SENCOMLNG)

This XML language has been designed to define the basic set of operations that can be performed by a sensor, including the format for the request, reply and data messages. By combining some of these basic operations, an application can build more complex ones and then extract semantic information. There are 4 main operations that can be issued a sensor:

- **getData**: This operation is used to request a measurement to a sensor (e.g. get a temperature).
- **setData**: This operation is used to set a register to activate some device which could be located in a sensor or actor (e.g. turn on a led).
- **monitoring**: This operation is useful when an application wants to obtain periodical measurements of the environment (e.g. obtain the temperature measured by a sensor every second).
- **trap**: This operation is related to the notification of events (similar to the SNMP [7] trap functionality), for example report an event when the temperature surpasses 40 degrees Celsius.

Following, a SENCOMLNG header element, a query message and a reply operation[1] are shown for a better understanding of the protocol.

[1]The header element is not included in the reply operation due to space limitation.

```
<LWESPv1>
  <header>
    <msgID> 0x12340000 </msgID>
    <srcID> 0xEEEEEEEEEEEEEEEE </srcID>
    <dstID> 0x1111111111111111 </dstID>
     <tstamp> 7-Jan-2009 12:52:45 (1231329159) </tstamp>
  </header>
  <monitorStart>
    <sensor32ID> 0x11111111 </sensor32ID>
    <data>
      <dataCat> temp_anyunit (0x00000100)</dataCat>
    </data>
    <measurePeriod> 1000 </measurePeriod>
    <reportPeriod> 8000 </reportPeriod>
  </monitorStart>
<LWESPv1>

<getDataReply>
   <sensorID> 0x22222222 </sensorID>
   <data>
     <timeDelta> -10 </timeDelta>
     <dataCat> temp_farenheit (0x00000102) </dataCat>
     <int32Values> 1352 </int32Values>
     <dataScale> -2 </dataScale>
   </data>
</getDataReply>
```

The root of every SENCOMLNG message is the $<LWESPv1>$ element that specifies the protocol version and contains the whole structure. Following, all SENCOMLNG messages contain a header structure which is defined with $<header>$. This element always has 4 different fields represented with the labels: $<msgID>$, $<srcID>$, $<dstID>$ and $<tstamp>$.

After the header structure, three different types of operations can be found: (i) **Requests**, which are messages sent from the applications to the WSN-GWs; (ii) **Replies** are the answer to the requests, and are sent from the WSN-GWs to the application and (iii) **Data** messages that are used in monitor and trap operations when the WSN-GW sends the monitor data or the event to the application. Request and reply are synchronous operations, so each request has a reply. On the other hand data messages are asynchronous and can be generated at any time.

A request could be composed of more than one operation. Thus, some overhead is saved because only one header is used for multiple operations.

The labels that identify each request operation are: $<getData>$, $<setData>$, $<monitorStart>$ and $<setTrap>$. All of them are XML elements that contain different fields depending on the particular operation.

The first field that is found inside any SENCOMLNG operation is the sensor identifier. The label used for this purpose is $<sensorXID>$, where the X indicates the number of bits used for the sensor address, being X = 16, 32 or 64.

Following, in all SENCOMLNG message requests, a data element appears using the label $<data>$ (except in the trap operations where it is labeled as $<thresholdData>$ but it contains the same fields). Although this element has different fields depending on the particular request, all of them include, the data category field which uses the $<dataCat>$ label and defines the sensor data measurement

type that is solicited: temperature, humidity, noise level, etc. The value inside this field is a number of 32 bits. The first 24 bits identify the data category itself, whereas the last 8 bits indicate the unit used in that category. For instance, in a get operation requesting a temperature value, the lower 8 bits would specify whether the requested value is in Celsius, Kelvin or Farenheit degrees (0x01, 0x02 and 0x03 respectively). However, if an application can manage any of the available units then these 8 bits could be left to 0x00 and the sensor can provide the value in any unit.

In the *getData* and *monitorStart* requests only the data category field is required within the data element. However, in a *setData* operation one value must also be provided in order to ask the sensor to perform the action requested by the application. The values in SENCOMLNG can be defined using all the data types provided by XBE32, which are listed in Section 3. In our testbed, we have used mainly numerical data using the labels $<opaqueXvalues>$ where X indicates the number of bytes of the data, and $<intXvalues>$ where X is the number of bits.
In addition, when an $<intXValues>$ field appears, the label $<dataScale>$ could also be used. Then, if in the dataScale field appears a -1 and the provided value in the previous attribute is 35, the value understood is 3.5. With this simple label, we avoid the necessity of managing floating point numbers which could be not available in low-power WSN-GWs and most sensors.

After the sensorID, in the trap operations, a threshold type for the event is established. For that we use the label $< op >$, that contains a numerical value which is linked to one of the following operations: $>$, $<$, $>=$, $<=$, $==$ or $!=$. Then, using the field $<op>$ together with the element $<thresholdData>$ the solution provides enough flexibility to define a big amount of trap operations. In addition, if several $<thresholdData>$ are included, complex *traps* could be generated.

The monitoring operation needs two more elements: $<measurePeriod>$ and $<reportPeriod>$. The first one indicates the interval when the sensor has to get the measurements and is specified in milliseconds. The latter represents the interval when the collected measurements must be forwarded to the application and it is also defined in milliseconds.

Finally any of the requests defined so far can use an optional field labeled as $<lifetime>$ which defines the milliseconds while the request is still valid.

The four main request operations have been already defined, however some other are required for providing a full functionality. Due to the dynamism of the *monitor* and *trap* operations, a *refresh* and *stop* operations are also needed. The first operation is periodically issued by the application to let the WSN know that the established monitor or trap operation still is needed, while the stop operation finalizes a given trap or monitoring operation. The labels for these two operations are: $<monitorRefresh>$, $<trapRefresh>$, $<monitorStop>$ and $<trapStop>$.

There is one reply operation for each of the four main request operations, labeled as $<getDataReply>$, $<setDataAck>$, $<monitorStartAck>$ and $<trapAck>$. Every reply includes the same fields than its respective request. This provides some negotiation capability in the interaction between the WSN and the external application. For instance, an application could request a sensor to report an event when the temperature goes over 30.5 degrees Celsius. However, maybe the sensor resolution is just 1 degree. Therefore, SENCOMLNG allows the possibility of including the value 30 degrees in the *trapAck*, which indicates to the application that the sensor is going to report events when the temperature goes over 30. In the case that the reply does not satisfy the application requirements, the application can just stop the process.
In addition, to what was explained for the trap and monitor replies, they include a field labeled as $<processID>$ that together with the $<sensorID>$ represents a particular process. This field eases the syntax of the refresh and stop operations because they only need these two fields, $<sensorID>$ and $<processID>$, apart from the header.
Also, an error message for all SENCOMLNG operations is defined. This error message includes the operation that has failed and the reason why it failed.
Finally, the last kind of SENCOMLNG messages are the asynchronous data messages periodically generated due to a monitor operation, or forwarded when an event related to a *trap* operation happens. The label for the last one is $<eventData>$, and it includes: $<sensorID>$, $<processID>$, $<data>$ structure including $<timeDelta>$, $<dataCat>$, XBE32 data type labels and $<dataScale>$. All fields have been already described except the timeDelta label that indicates the difference of time between the $<tstamp>$ value of the header and when the sensed data was obtained. This design decision is to replace the 64-bit long timestamps by 32-bit delta field.
The label for the data from a monitor operation is $<monitorData>$ and it includes $<sensorID>$, $<processID>$ and a set of $<data>$ elements, as many as measurements are reported. All these data structures contains a $<timeDelta>$ related to the header's timestamp and a value field. The first one, in addition to these two fields, includes $<dataCat>$ and $<dataScale>$, that are also applied to the rest of $<data>$ elements.
In a nutshell, SENCOMLNG allows the applications to obtain any kind of raw data from a WSN. However, it is a task of the application, that is supposed to run on a powerful device, to fuse, combine, etc., the sensor data in order to obtain the desired information.

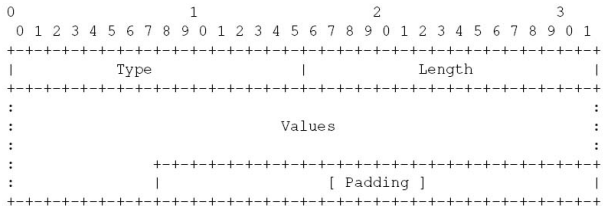


Figure 1. XBE32 TLV FORMAT

3 XBE32 Overview

We defined a binary encoding of the SENCOMLNG with two main goals in mind:

- We shall codify SENCOMLNG messages being aware of the bandwidth consumption due to the potential big amount of data that can be transmitted from WSNs to the applications. Furthermore, a given WSN could be connected to many applications and an application could be connected to many WSN-GWs. Therefore, bandwidth could be an important limitation of the system.

- SENCOMLNG is a very flexible tool that allows to extend the communication with new sensors in an easy and quick way. Therefore, the codification used must allow extending SENCOMLNG in an easy way as well.

The chosen codification is the eXtensible Binary Encoding (XBE32) [10] which allows implementing a binary protocol based on SENCOMLNG, being at the same time much more efficient in terms of bandwidth and processing, and fully extensible for including future SENCOMLNG extensions.

XBE32 is a simple binary encoding that carries hierarchical data, thus it is similar to XML. XBE32 elements are serialized inside TLV structures (Figure 1) which are 32-bit aligned to ease the parsing process. As data is clearly delimited by length, XBE32 does not require to escape characters as XML does, thus it also eases message creation.

XBE32 defines two kinds of XBE32 Elements: *Attribute Elements* which carry primitive data values, and *Complex Elements* which do not carry data by themselves but contain other XBE32 Attributes and/or other Complex Elements themselves.

In order to be employed by modern programming languages, XBE32 uses common primitive data types for its Attribute Elements, such as Strings, Booleans, Integers, Floats, as well as Arrays. Other data types can be encoded using the different binary Opaque value types defined by XBE32.

The **Type** field is 16 bits long and describes the processing rules, TLV structure and what kind of data is carried inside the Values field. Following, we describe the subfields inside Type field.

C and E bits: Specify what to do if the type is not known: continue processing and/or send an error message.

Meta (6 bits): This subfield describes the internal structure of the TLV's Values field, as well as the type of the primitive data it contains. Following the list of Meta Types is shown:

Type.Meta	TLV Values structure
0x00-0x1F	Multiple variable-length TLVs
0x20	Single variable-length opaque Value
0x21	Single variable-length string Value
0x24	Multiple opaque1 Values
0x25	Multiple int8 Values
0x26	Multiple boolean Values
0x28	Multiple opaque2 Values
0x29	Multiple int16 Values
0x2C	Multiple opaque4 Values
0x2D	Multiple int32 Values
0x2E	Multiple float32 Values
0x30	Multiple opaque8 Values
0x31	Multiple int64 Values
0x32	Multiple float64 Values
0x34	Multiple opaque12 Values
0x38	Multiple opaque16 Values

All Meta types have a self-defined structure except of those of opaque type. An opaque X value contains any kind of data having a length of X bytes. This provides high flexibility for protocols or applications that has to use XBE32, because the opaqueX Meta type offers a container for anything.

Subtype (8 bits): This subfield, together with the Meta field, identifies the semantic meaning of this TLV and/or the data carried inside its Values field. Therefore, Subtype values should be defined by the upper application/protocol that is employing a XBE32 encoding.

The **Length (16 bits)** field specifies the size in bytes of the whole TLV structure, excluding padding.

Finally, the **Values** field may contain a single variable-length value, multiple fixed-length values, or other TLVs, as defined by the Type and Length fields. To properly align a non-empty Values field to 4-octet words, up to 3 octets of padding space could be added and filled with zeros.

4 Light-Weight Exterior Sensornet Protocol (LWESP).

LWESP is a protocol that enables the communication between a Wireless Sensor Network Gateway (WSN-GW) and an external application.

After the description of both, SENCOMLNG and XBE32, it is easy to understand that XBE32 is a suitable codification schema for reducing the bandwidth in the communications that are based on SENCOMLNG. It is only necessary

a dictionary to map the SENCOMLNG elements to XBE32 types and vice versa. The dictionary has one entry per SENCOMLNG label, and follows the notation *SENCOMLNG label = XBE32 Type*.
All SENCOMLNG labels that define complex elements containing other labels will be defined as XBE32 Complex Types, whereas the rest of the labels that only contain a value but and not other labels are mapped into Attribute TLVs. These labels have a type depending on the length and value type of the contained information. Later, the subtype defines the meaning of a given TLV in the context of LWESP.

5 Related Work

In [9], Vassileios Tsetsos et al, discuss how a Sensor-Based Services (SBS) model, which combines mobile telecommunications technologies and WSN, can be realized. In this work the authors defines the Unified Sensor Language (USL), an XML-based language similar to the SENCOMLNG. Both languages present similarities since both highlight the monitor and event operations as crucial when an application wants to communicate with a WSN. However, the USL is more complex than SENCOMLNG. In USL, sensors have to understand requests that include filter compositions and they claim that in a future the USL might support complex time-based events like "if the temperature change rate in a computer room is +3 ^{o}C/h, inform the building caretaker so as to check the air conditioner". We believe that this complexity is not feasible nowadays and not even in a near future, specially because vendors would need to develop specific sensors for supporting this complex language. Furthermore, USL does not pay attention to the possibility of interact with actors, that can be requested to perform some actions. SENCOMLNG has defined the $<setData>$ operation having this in mind. In addition, USL does not consider the data category concept, where it is included or how it is defined, so they assume that applications and sensors have some previous common knowledge. Finally, USL does not encode the XML messages.
On the other hand, the WASP project [3] propose employing Web Services for communicating business applications with WSNs via a middleware. Web Services are quite popular, therefore it facilitates the implementation task. However, they are costly in terms of processing and bandwidth consumption, and they can only be considered in the case when a WSN-GW is a powerful device. Moreover, depending on the scenario, even with a powerful WSN-GW, the use of Web Services could create at bottleneck in the communications due to its high bandwidth requirements.
In the CoBiS project [1], UPnP [2] is proposed to interconnect applications and WSN. It must be highlighted that the UPnP protocol uses SOAP (Service Oriented Architecture Protocol) and XML over HTTP. Therefore, the same drawbacks introduced for the solutions based on Web Services appear in UPnP solutions.
Finally, there are other solutions [4, 6, 8] that propose different approaches to interconnect applications and WSN. However a direct comparison between them and our proposal cannot be established since they define an architecture with intermediate active nodes involved in the path between applications and WSNs.

6 Performance Evaluation

An XBE32 parser has been implemented in JAVA [2]. This implementation offers an API that includes functions to convert XML documents to XBE32 using a dictionary, and the opposite operation, transforming XBE32 messages to XML documents.
Using this XBE32 implementation, we have deployed a test-bed to validate and evaluate the LWESP protocol. With this goal in mind we have implemented the WSN-GW functionality as well as emulated sensor behavior using JAVA.
First of all, we have validated that the implementation of a new defined protocol as LWESP does not need much development time since it is based on XML and codified using the XBE32 library. But what is even more important, the time required for updating the protocol to add new label definitions is really short. Any new label only requires appending a dictionary entry and a small piece of code to process it.

In this section LWESP is compared with other solutions in order to demonstrate that our proposal has a dramatic impact in the reduction of the bandwidth requirements.
First of all, we want to show the bandwidth saving when XBE32 is used instead of using directly XML SENCOMLNG messages in the communication between the application and the WSN-GW. For that we evaluate the number of bytes needed for requesting a temperature value and the number of bytes in the reply assuming that a single temperature measurement is digitalized to 32 bits. We used our JAVA test-bed for evaluating it. The number of bytes of the SENCOMLNG query is 329 bytes and the SENCOMLNG reply size is 445 bytes, whereas the same results using LWESP were 76 and 100 bytes respectively. This leads to a saving of 77%.
It must be noted that the longer the XML labels are, the higher the bandwidth consumption using XML is. Therefore, in order to demonstrate the power of the XBE32 codification, the size of the SENCOMLNG request/reply was evaluated using labels as short as possible, that is just 1 character per label name in the best case. In addition, we evaluate a single request message including a *getData*, and

[2]The source code is available at url http://www.it.uc3m.es/muruenya/xsdf/

Solution	*1 Req.*	*1 Reply*	*8 Req.*	*8 Replies*
SENCOMLNG	329	445	1521	2195
1 char label	149	177	565	680
LWESP	76	100	312	440

Table 1. Number of Bytes required for a single getData Request/Reply and a multiple (8 getData) Request/Reply

when 8 *getData* operations are included in the same message. Table 1 summarizes the results.

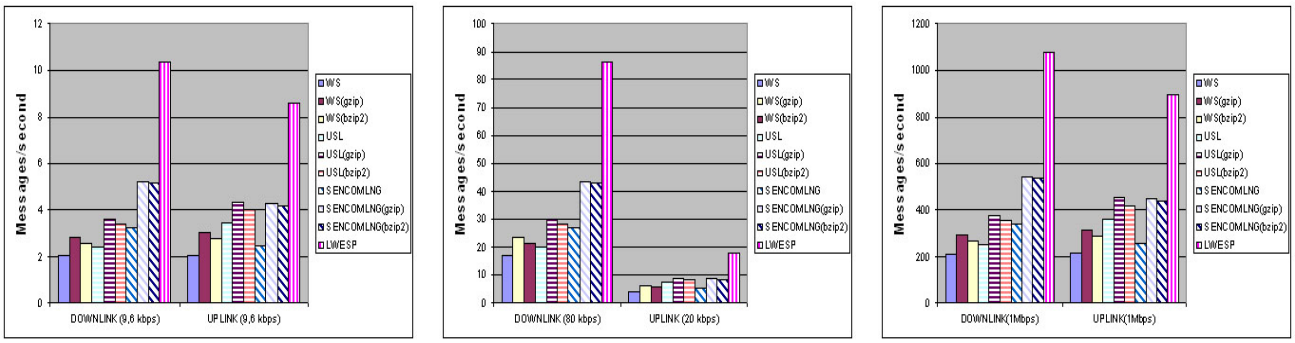
Table 1 shows that the use of XBE32 outperforms the use of XML, even if we assume the case when the SENCOMLNG avoids all the blank spaces and uses a single character per label (which is not a very realistic approach). Even in this case, the 49% of bytes are saved in the single request and 44% in the single reply.
In the results related to a multiple request, the first conclusion is that even in large messages, reducing the label size to the minimum (1 byte) does not surpass the XBE32 encoding, that saves a 45% of bandwidth in the request and more than a 35% in the reply.

Following, we evaluate the LWESP in front of different approaches. In particular we compare the two possible solutions introduced in this paper, SENCOMLNG and LWESP, in front of two other approaches: USL [9] and the protocol defined in the WASP project [3] based on Web Services (SOAP over HTTP) already presented in the related works. To obtain a clear overview of the bandwidth consumption, the maximum number of messages per second that can be sent/received using the different approaches has been compared. Moreover, different access technologies have been used in order to obtain more valuable results. The technologies utilized and their respective downlink/uplink rates are: GSM 9.6/9.6 Kbps, GPRS 80/20 Kbps and DSL 1/1 Mbps. Finally, it must be noted that it has been considered that all the communications runs over TCP/IP since the Web Services approach is using HTTP. Therefore, 40 extra bytes of the TCP/IPv4 header are added to each message in order to get more real values.
In order to be fair in the comparison we have used the real message format for all the solutions. Basically, it was compared the maximum number of request messages per second that can be received by the WSN-GW when an application request a temperature to a sensor, as well as the number of reply messages per second including the sensed temperature that the WSN-GW can send back to the application. Furthermore, we applied compression to XML and WS messages using gzip and bzip2, to see if even in such conditions our solution outperforms the other approaches. Figures 2(a), 2(b) and 2(c) shows the results for GSM, GPRS and DSL respectively.

Looking at the graphs, LWESP clearly outperforms any other solution, even if compression is applied to the other approaches.
In the GSM case, the most restrictive WSN-GW access technology, LWESP is able to send more than 8.5 reply messages/second, the double than the following best approaches, USL and SENCOMLNG compressed with gzip that send around 4.3 messages/second at the most. The same result is obtained when looking to the maximum number of request messages received (downlink), the best result is over 10 messages/second when LWESP is used, followed by the gzip SENCOMLNG solution that is able to receive 5.27 messages/second. In addition, it must be noted that compression algorithms have an important processing cost, specially when the WSN-GW is a low resource device. Then, if we compare all the approaches without compression, the use of LWESP has a dramatic impact on the bandwidth utilization, since our solution sends/receives between 3 and 5 times more messages/second than any other approach.

Following, Figure 2(b) shows some similar results when comparing the different solutions over GPRS. Our protocol sends, at least, the double of messages/second, than the best of the other solutions applying compression, in both the uplink and the downlink, but this value could reach up to 5 times more number of messages/second in the downlink when our solution is compared with the WS approach. In addition, what is interesting in the GPRS results is to appreciate the difference between the results for the uplink and downlink. Many deployed access technologies are asymmetric (GPRS, ADSL, etc.) and generally offer to the users higher downlink than uplink rate. In our scenarios this implies that a WSN-GW will be able to receive more requests than replies and data messages it is able to send. Therefore, when a WSN-GW is using these asymmetric access technologies, it is clearly limited by the uplink capacity in sending data to the applications.

Finally, a high rate and symmetric access technology is shown in Figure 2(c). Again, the comparison among the different approaches clearly presents the LWESP like the best solution in terms of bandwidth utilization. However, using an 1 Mbps access link in the worst case, WS approach without compression, rates over 200 messages/second are achieved in both the uplink and the downlink. Then in many scenarios with few numbers of sensors any of the solutions could be adopted and the bandwidth utilization does not seem an important issue.
However, nowadays the interest about large WSNs is growing more and more. Therefore, we can expect in the near future WSNs with hundreds or even thousands of sensors deployed in the field. Talking about such number of sensors, the values shown in the Figure, from 200 to 1000 messages/second could not be even enough, thus the bandwidth



(a) GSM (b) GPRS (c) DSL

Figure 2. Performance Evaluation with GSM, GPRS and DSL

consumption of the WSN-GW must be taken into account. Furthermore, if we think in a popular WSN providing a hot-topic information, not only one but many applications could request data to that network. Then if there are hundreds of sensors and hundreds of external users requesting the data, again LWESP should be the chosen solution.

7. Conclusion and future works

This paper has presented a solution to access sensor data from external applications in a standard manner without depending on the particular WSN platform. A novel communication protocol, LWESP, between applications and WSN-GWs is introduced. This protocol is based on an XML language, SENCOMLNG, codified using XBE32. The combination provides a light-weight protocol retaining the advantages of XML regarding flexibility, extensibility and easy development. In addition, it reduces the processing complexity and leads to a dramatic reduction of the bandwidth utilization in the communications.

A test-bed has been deployed in JAVA in order to evaluate our proposal, and it must be highlighted that updating the protocol and adding new fields has required little effort.

Furthermore, our proposal outperforms any other proposal in the literature based on XML, Web Services and UPnP in terms of bandwidth utilization. LWESP is able to transmit/receive at least two times more messages/second than any of the other solutions, even if these ones are using compression. In addition, it has been also shown that reducing the size of the XML labels as much as possible (1 character per label) still offers worse performance than LWESP since it saves at least 35% of bandwidth for large messages and around a 45% for short ones.

On the other hand, SENCOMLNG, the base of this protocol, has been shown to be an easy and flexible language for defining WSN-GW messages. The SENCOMLNG messages define 4 different operations: *getData*, *setData*, *monitor* and *trap*, that should be easily mapped to most of the available WSN in the market.

For the future work, we are working on the sensor network side defining an application protocol that can be directly mapped from LWESP, so that the translation between LWESP messages and WSN messages requires low processing cost. Furthermore, we want to evaluate how LWESP itself would behave as the application layer protocol inside the WSN, because if a WSN works properly with the defined protocol, we could think in an end to end communication between applications and sensors without the necessity of translation in the WSN-GW, apart from the necessary MAC and routing layers adaptations.

References

[1] Cobis project. *CoBIs: Collaborative Business Items*, http://www.cobis-online.de/.

[2] Upnp forum. *UPnP: Universal Plug and Play*, http://www.upnp.org/.

[3] Wasp project. *WASP: Wirelessly Accessible Sensor Populations*, http://www.wasp-project.org/.

[4] K. Aberer et al. *Global Sensor Networks*. EPFL LSIR-REPORT-2006-001, 2006.

[5] I. F. Akyildiz et al. *Wireless sensor networks: a survey*. Computer Networks, 2002, 38 (4) pages: 393-422.

[6] P. B. Gibbons et al. *IrisNet: An Architecture for a World-Wide Sensor Web*. IEEE Pervasive Computing, Oct-Dec 2003.

[7] J. Case et al. *A Simple Network Management Protocol (SNMP)*. IETF, RFC 1157, May 1990.

[8] J. Shneidman et al. *Hourglass: An Infrastructure for Connectiong Sensor Networks and Applications*. Harvard Technical Report TR-21-04, 2004.

[9] V. Tsetos et al. *Towards Commercial Wireless Sensor Networks: Business and Technology Architecture*. Ad Hoc & Sensor Wireless Networks, 2006.

[10] M. Urueña and D. Larrabeiti. *eXtensible Binary Encoding (XBE32)*. IETF <draft-uruena-xbe32-02.txt> (Work in progress), September 2007.