UPnP Compression Implementation for Building Automation Devices

Stefan Knauth, Rolf Kistler, Daniel Käslin and Alexander Klapproth

Abstract— **IP based field bus networks enable the usage of common IP protocols for example for security or automatic configuration, on the field level. IP based devices can take advantage of the well-specified UPnP protocol for commissioning and operation. A drawback of this protocol deployment is the high required network bandwidth due to enormous high level protocol overhead, especially on the XML- and SOAP based UPnP schemes. We investigate UPnP datagram size reduction on an experimental IP based field bus with Ethernet and wireless IEEE802.15.4 devices, for building automation and control applications. To be able to use standard unmodified UPnP stacks, we use transparent bidirectional XML/SOAP proxies for example on the Ethernet/IEE802.15.4 hub and on the wireless devices itself. The proxy uses cache-based tokenizing of the SOAP messages and feedback, typically deflating subsequent similar messages by orders of magnitude, thereby significantly reducing the number of data packets and increasing the network performance especially when using IEEE802.15.4 as wireless physical layer.**

I. Introduction

UPNP compression is an important aspect of our ongoing research on deploying IP as fieldbus protocol in building automation and control networks. IP as fieldbus protocol allows adopting common IT network technologies to the field devices provided that the devices offer sufficient computational resources. Using highly optimized stacks, it is possible to integrate TCP/IP and high level protocols like the chain HTTP-SOAP-UPnP on low resource devices [1], [2].

For current fieldbuses in building automation and control (LON, EIB/KNX, PROFIBUS [3], [4], [5]), connectivity to IP networks is typically realized with an OPC [6] server. The field devices themselves do not communicate via IP. There exist solutions, where devices of, for example, an EIB field bus system are presented on an IP network in a UPnP [7] facade [8]. This approach is somewhat superior to an OPC XML presentation, but still the BAU devices are operating on their native field bus protocols.

For building automation, ease of device and network commissioning is very important, in order to keep the network manageable and reliable and keep maintenance efforts and costs on a reasonable level [9]. The used protocols should be standardized or widely– and vendor–independently used. Therefore we chose UPnP for configuration and operation. UPnP (Universal Plug & Play, (C) Microsoft Corp.) [7] is a protocol for automatic device integration into IP networks. It covers addressing, device discovery, description, control, eventing, and presentation. The initial target of UPnP technology was, among others, home multimedia and office applications for PCs, but nowadays there are also templates defined for HVAC (Heating, Ventilation and Air-Conditioning), and yet proprietary device types can be modelled in the basic device template.

One drawback of using UPnP as operation protocol is its ”talkativeness”. Since it is based on SOAP [10], XML and HTTP, a basic command easily can have a length of several hundreds of bytes, which may cause congestion or command propagation delays when IP communication is carried out via low- or mid rate physical layers (PHY), in our example the wireless standard IEEE 802.15.4. In this paper we report on our investigations on XML/SOAP compression for UPnP applications in building automation networks.

Manuscript received January 15, 2007, revised April 12, 2007.

This work is supported by the Hasler foundation, Berne, Switzerland, in the frame of the SARBAU project (ManCom Call).

Performed at the CEESAR competence centre (www.ceesar.ch) at Lucerne University of Applied Sciences, Technikumstr. 21, CH 6048 Horw/Lucerne, Switzerland. EMail sknauth@hta.fhz.ch

II. UPnP compression

A. Overview

The UPnP command transfer is done via SOAP messages, transferred as XML data sets. A SOAP message for a typical command like ”switch light on” will have a length of some 100 Bytes. Besides addressing, the transferred information contains the desired state of the lamp, in the particular case 1 bit. Typically, in building automation such messages are transferred from one target to another with a rate well below 1 Hz. So in a 10 or 100 MBit Ethernet-based IP network, the traffic overhead due to the utilization of SOAP is not impacting the network in any way and the advantages of having a clearly defined command transfer mechanism and syntax justifies the overhead.

The situation changes when regarding two-wire or wireless fieldbus systems with rates down to 9600 bps. To apply UPnP in such scenarios, it is under many circumstances necessary to reduce the size of a command datagram. In a recent paper [11] we investigated the applicability of available compressors and known approaches in the field of XML compression: Possible methods include for example ASN.1 conversion ([12], [13]) or binary XML ([14]). Stream compressors are typically based on the Lempel-Zev algorithms ([15], [16], [17]), where a dictionary is continuously updated and reconstructed from the decoded message on the receiver side. They are used for example in the serial IP protocol PPP (Peer to Peer Protocol), as MPPC (Microsoft Point to Point Compression) [18] on CCP (RFC1962: Compression Control Protocol) [19]. These methods are very good in lossless compression of streams with local correlation, and do not use or need any prior knowledge of the structure of the data to be processed. On the other hand their horizon is defined by a sliding window so that no quasi–static parts in the dictionary are foreseen.

Compression meant as suppression of repeated transmission, is for example used in the TCP header compression [20] used in the SLIP (Serial Line IP) protocol. More sophisticated methods like the XMill [21] and XGRIND [22] projects use a variety of technologies for decomposing and tokenization, but do not especially encounter for repeatedly sent datagrams, and are not suitable for low resource devices. The WAP Binary XML Content Format (WBXML) [23], as for example used in mobile radio WAP XML compression, is tailored to lower resource systems and transmission on low-rate wireless channels. It uses predefined tokens for commonly used strings and for syntax elements, and uses a table for referencing repeatedly used document strings. The scope of the compression is one "document", in our case this would map to a single SOAP XML datagram.

None of the investigated methods did ideally fit to the problem of UPnP compression for fieldbus devices in building automation and control. On one hand, a suitable method must take into account the limited computational resources of the devices, which can be as low as 60 KBytes of ROM and 4 KBytes of RAM [1]. On the other hand typically there is only a few configurable attributes on a device, which leads to a high redundancy in subsequently filed UPnP datagrams. Therefore for compression we chose a tokenization mechanism, were message parts which occur frequently are replaced by tokens. This approach is related to the WBXML format. Additionally our method implements dynamic string dictionaries (in this paper also termed "caches") which have a lifetime beyond the scope of a single XML datagram. The algorithm is able to consistently handle connections to multiple endpoints.

B. Bidirectional Proxy

Since we want to use devices which implement standard UPnP, the tokenizing and the recomposition shall occur in proxies, which intercept the connection between the sender and the receiver of an UPnP message. "Sender" is here meant in the scope of the message direction, i.e. when a control point sets an attribute in an UPnP device, at first the control point acts as "sender" by sending the message to the device, which acts as "receiver". When the device sends its response, it acts as "sender" and the control point acts as receiver. Messages are compressed at the sender side and decompressed at the receiver side, or at the corresponding proxies.

The dictionaries for substitution of strings by tokens are dynamic and the proxies learn by performing command transfers. Principally, in contrast to Lempel-Ziv [15], [16] methods, the transmitted data does not contain the dictionary information within a defined window. Also, by requirement, the sender shall not rely on the dictionary state at the receiver. Therefore there exist a feedback mechanism enabling the receiver to request missing information from the sender. The lifetime of the dictionary is beyond the lifetime of a single datagram transmission or the corresponding TCP connection.

In figure 1 a part of a typical network layout is sketched. IP capable devices are connected either directly to the backbone or communicate via a gateway. Gateways are especially used to connect to wireless devices. As it can be seen in the figure,

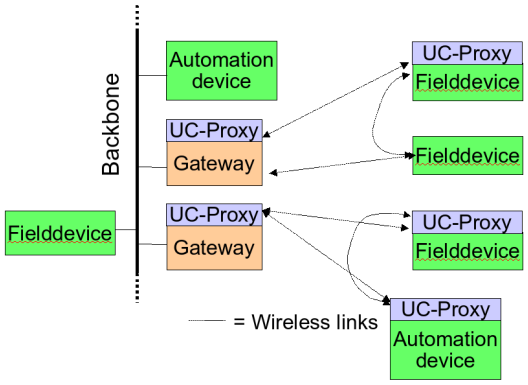

Fig. 1. View into a typical network constellation. Wireless devices are connected to the backbone via gateways, which also act as UPnP compression proxy.

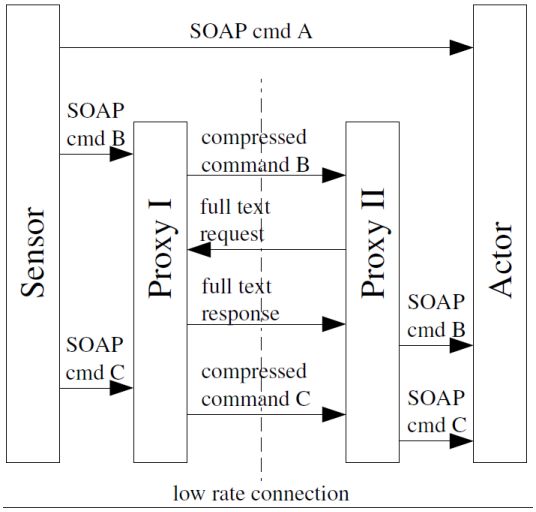

Fig. 2. Compression scenarios. (A) direct transmission (B) learning scenario (C) compressed command transfer.

the gateways and most of the wireless devices are equipped with a UPnP bidirectional proxy such that the communication among the devices is compressed, reducing the datagram size and thus the number of needed IEEE802.15.4 data packets typically to one packet. When communicating with devices which are connected directly to the backbone and which are not compression-aware, the UPnP data sent over the wireless link is still compressed since the gateway is equipped with a transparent compression/decompression proxy.

Figure 2 shows three scenarios for command transfer. Command A is transferred directly from the sensor to the actor. This is the normal UPnP scenario. Command B is transferred through Proxy I - Proxy II. Proxy I replaces some parts of the message with tokens, and stores this compression information in his cache. Proxy II does not yet know about these tokens and requests their full text. After that, the command can be reconstructed and transmitted to the actor. In the case of command B, the overall amount of transmitted data is higher as it would be in the uncompressed scenario A, and also the command delivery will be delayed by the duration of the additional packet transfers. Depending on available memory

UPnP Action Datagram (Switch on Light)

```
 1: POST /HTA_wireless_Switch/V10 HTTP/1.1
 2: HOST: 147.88.104.23:53194
 3: CONTENT-LENGTH: 327
 4: CONTENT-TYPE: text/xml; charset="utf-8"
 5: SOAPACTION: "urn:schemas-upnp-
org:service:SwitchPower:1#SetTarget"
 6:
 7: <?xml version="1.0"?>
 8: <s:Envelope
 9: xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
10: s:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
11: <s:Body>
12: <u:SetTarget xmlns:u="urn:schemas-upnp-org:service:
                 SwitchPower:1">
13:     <newTargetValue>1</newTargetValue>
14: </u:SetTarget>
15: </s:Body>
16: </s:Envelope>
```

UPnP Action response

```
 1: HTTP/1.1 200 OK
 2: CONTENT-LENGTH: 290
 3: CONTENT-TYPE: text/xml; charset="utf-8"
 4: EXT:
 5: SERVER: HTA-UCOS/1 UPnP/1.0 WLESS_SW/3.141
 6:
 7: <?xml version="1.0"?>
 8: <s:Envelope
 9: xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
10: s:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
11: <s:Body>
12: <u:SetTargetResponse xmlns:u="urn:schemas-upnp-
org:service:SwitchPower:1">
13:     <newTargetValue>1</newTargetValue>
14: </u:SetTargetResponse>
15: </s:Body>
16: </s:Envelope>
```

Fig. 3. Sample datagrams with HTTP, SOAP and XML content

and entropy of the transferred data, this will only happen very occasional. If the sending proxy knows that a text part is not available in the receivers dictionary, he directly transmits the string and a token, and the receiver does not need to request the full text. Command C is also transferred via the proxy chain. In the case C, all tokens which have been used in the command during transfer from Proxy I to Proxy II, are known to proxy II, and the UPnP command can be reconstructed without further communication.

C. Compression Algortithm

The actual compression is performed by replacing text with tokens. The compressor has some implicit knowledge of the SOAP format which, in our implementation, was coded as rules into the algorithm. These rules are best described by looking at a typical UPnP XML datagram, as shown in figure 3 in the “UPnP Action datagram”: With respect to the receiver device, besides the body length, lines 1..3 do not change between subsequent commands. Therefore the first 3 lines are replaced by a token called the ”address“ token and the length information. Lines 4 and 5 are again replaced by one token, the ”action” token. The third token, “body start” consists of the whole body up to the attribute value, i.e. lines 7..13. The

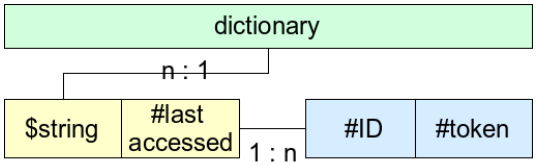

Fig. 4. Dictionary data structure.

compressor recognizes the attribute values, by simple semantic analysis i.e. by finding text not encapsulated by the ”<“ and ”>“ characters. The attribute value is not compressed. If there are more attributes, the part between the attribute values is put into different ”separator“ tokens. The remaining part of the message is again replaced by one token, called the ”body end“ token. In the whole the compressed message consist of the following items: An address token, the length information, the start element, a header token, an action token, the ”body start“ token, the attribute value, and the ”body end“ token.

While being quite simple, this compression algorithm has the advantage of being easily implementable on low resource devices. The text blocks are actually not really compressed but stored in the dictionaries. This of course works only if the set of applicable UPnP commands and attribute names on the device is small and the number of the to–be–stored text fragments is low such that the strings fit into the device memory. The compressed messages may be composed out of the following elements:

- command: begin of UPnP message
- command: end of UPnP message
- command: receiver cache reset
- command: receiver cache reset OK
- command: request full text for token
- command: full text answer for token request
- text
- token
- text with token

Besides the ”request full text“ and the ”cache reset OK“ command, the commands are issued by the sender (compressor) and consumed by the receiver (decoder).

There is no 1:1 relationship between the sender and the receiver of UPnP messages. Instead, principally any device may send messages to any other device. As mentioned earlier, each proxy keeps a dictionary of strings and tokens. The layout of the dictionary is sketched in figure 4. For each string, there is a list of ID/token pairs which refer to that string. The tokenisation algorithm implements the following rules:

- Each device has a unique device ID, for example the IP address or a subset of it.
- A token is a number with a limited length, for example 1 byte or 2 byte. It references a string in a string dictionary. Several ID/token pairs may reference the same string.
- A token is defined between two particular devices. A token with the same number may have a different meaning when used between the same sender but another receiver
- A device uses the same dictionary for sending and receiving. When sending, the ID in the dictionary structure refers to the receiver ID and when receiving it refers to

the sender ID.

- The sender looks whether a string to be tokenized is already in his dictionary. If not the sender tries to to put the string in his dictionary, and transfer tokens to the receiver. If there is no more memory for dictionary entries, the sender may remove a string from the dictionary to make place for a new one, or may send the string without tokenization.
- If a receiver has no entry for a token, for example because he removed it to make place for other tokens, he requests the full string at the sender. For storing and string replacement, the same rules as for a sender apply.

The devices in the network may loose dictionary alignment, for example due to power loss, radio link loss, etc. The following rules ensure consistent dictionary states in the system, i. e. the receiver of a message does not misinterpret tokens which have already changed their meaning:

- Each sender keeps a record, the ”clean ID list“ of IDs which have consistent cache content with respect to the sender.
- If the receiver of a message is not in the Clean ID list a ”receiver cache reset“ command is issued, on which the receiver clears all references to the sender ID in his cache, and the sender puts the ID in the list.
- If a device is powered up or otherwise feels disturbed, it completely clears its cache and its ”Clean ID“ list.
- A sender enumerates tokens on a per receiver base. When there is an overflow in the token counter, the corresponding receiver gets a cache reset command.

On low resource devices, the memory available for the dictionary is limited such than not all appearing strings may be available in the dictionary simultaneously. Therefore the algorithm employs a simple dictionary string storage and release strategy based on ”least recently used“ (LRU), but extended with a configurable relaxation mechanism to delay string drops.

- The dictionary carries a global ”dictionary access counter“. Each time the dictionary is accessed, this counter is incremented.
- Each string in the dictionary is accompanied by a ”last accessed“ index. The ”last accessed“ index of a string in the dictionary is is set to the current ”access counter“ value each time when the string is accessed. ”Access“ here means a lookup by token, as performed during token reception, a lookup by string, as performed during tokenisation, or the insertion of a new string.
- When there is no more space in the dictionary, the sender looks for the entry which has the lowest ”last accessed“ value. It then looks whether the difference between the current ”access counter“ value and the ”last accessed“ value is bigger than a constant Q, which is a configurable parameter, and is typically 2..3 times the string capacity of the cache. Capacity is meant as the cache size divided by the average cache entry size.

The meaning of the Q parameter may be discussed with help of figure 5. The scenario from which the values for the graph were taken is as follows: 10 UPnP devices are controlled by one control point. There are 5 groups of devices with 2 members each. In a group, the command strings are equal for either device, besides the ”address“ part. The control point sends data cyclic to all 10 devices. Providing enough dictionary space, after initial transmission of commands, all commands are stored in the control point dictionary and in the UPnP devices dictionaries. This is the scenario on the right side of figure 5, the data rate is at 7% of the uncompressed data rate. The actual dictionary requirement is about 5000 bytes. The average length of the complete uncompressed messages is about 450 bytes. If the dictionary size of the control point is reduced (the cache size of the UPnP devices is big enough in this experiment), not all command strings will fit in the dictionary. The result in average data size reduction is now dependent on the strategy of cache management.

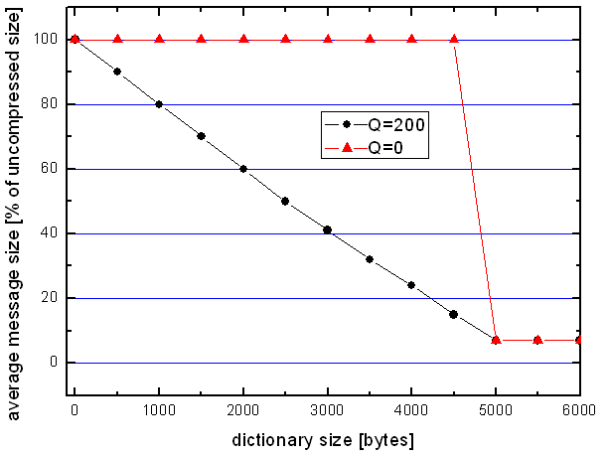

Fig. 5. Compression ratio vs. dictionary size: One controlpoint accesses 10 UPnP devices.

The line with the triangles represents the common ”least recently used“ strategy, where the string, which has not been used for the longest time, is removed from the dictionary and the new string is put into the dictionary. It is obvious that when cyclic addressing all devices one after each other, even when the cache holds strings for 9 devices, if used with 10 devices will not be useful at all because when data is sent to device 10, the oldest entry, which is for device 1, will be deleted always. Then, when sending data to device 1, the oldest entry will be the one for device 2, and so on.

Therefore a relaxation mechanism has been implemented. Using this relaxation mechanism, dictionary entries are only released if they have not been accessed for more than a definable number Q of dictionary-lookups. This leads to entries not be released immediately when memory is completely filled. In the given scenario, setting Q to a value of 200 will lead to usage of the cache for devices 1..9, and communication with device 10 will be uncompressed. Using this relaxation mechanism, compression will be useful even if the dictionary size is much smaller than the number of devices to be controlled.

III. Conclusion

We implemented a compression scheme for UPnP messages. This is motivated by the wish to use UPnP for configuration and operation of field level devices for building automation and control, on low transfer capacity physical layers. We use bidirectional tokenizing proxies for UPnP compression. Deployment of such proxies allows the use of existing standards compatible UPnP implementations on the devices and the management respective automation nodes, with highly reduced data rates on the IP network. Compression rates of more than 1:10 are achieved by intelligent splitting of UPnP messages in several parts and use of cache based text replacement. The developed schemes and implementations are tailored to low resource devices down to 8 bit microcontroller systems. The UPnP compression is especially useful when communicating over IEEE802.15.4 or other low rate wireless links. Here the packet rate can be reduced by a factor of about 5, which eases UPnP deployment in these environments and significantly decreases response times.

References

[1] A. Klapproth, D. Käslin, and T. Bürkli. (2005, Apr.) Zigbee entwicklerforum 2005 munich: Lowcost wireless webserver. [Online]. Available: http://www.ceesar.ch/cms/upload/pdf/Paper%20Wireless %20Webserver%20HTA%20Luzern.pdf

[2] A. Klapproth and T. Bürkli. (2005, Apr.) Zigbee entwicklerforum 2006 munich: Tcp/ip über ieee 802.15.4. [Online]. Available: http://www.ceesar.ch/cms/upload/pdf/FH-Luzern_TCPIP-over-IEEE802%2015%204.pdf

[3] (1990) Lon: Local operating network. [Online]. Available: http://www.echelon.com

[4] (1995) En 50090 - eib/knx open standard for home and building control. [Online]. Available: http://www.konnex.org/

[5] (1995) Profibus international. [Online]. Available: http://www.profibus.com/pb/

[6] OPC Foundation. (1996) Openess, productivity, collaboration. [Online]. Available: http://www.opcfoundation.org/

[7] (2003, Dec.) Upnp forum,upnp^{TM} device architecture 1.0, version 1.0.1. [Online]. Available: http://www.w3.org/TR/wsdl20/

[8] W. Kastner and H. Scheichelbauer, *UPnP Connectivity for Home and Building Automation*. IASTED Anaheim, Calgary, Zürich: Proc IASTED (PDCN) 2004, Hamza M. H. (Ed.).

[9] M. R. Brambley and S. Katipamula, *Beyond Commissioning: The Role of Automation*. Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161: U.S. Department of Energy, 2005.

[10] (2003) Soap version 1.2 part 1: Messaging framework. [Online]. Available: http://www.w3.org/TR/soap12-part1/

[11] S. Knauth, D. Käslin, R. Kistler, and A. Klapproth, "Upnp compression for ip based field devices in building automation," in *Proc. 11th IEEE Conf. on Emerging Technologies and Factory Automation (ETFA06)*, 2006, pp. 445–448.

[12] (2002) Abstract syntax notation one (asn.1) specifications, itu-t rec. x.680 x.683 and x690 x.693 (2002) iso/iec 8824-1:2002 to 8824-4:2002 and iso/iec 8825-1:2002 to 8825-4:2002. [Online]. Available: http://asn1.elibel.tm.fr/xml/

[13] O. N. Inc, "Alternative binary representations of the xml information set based on asn.1," in *W3C Workshop on Binary Interchange of XML Information Item Sets, Santa Clara, California, USA*, Sept. 2003.

[14] (2004) Xml binary characterization working group public page. [Online]. Available: http://www.w3.org/XML/Binary/

[15] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE-Transactions-on-Information-Theory*, vol. IT-23, no. 3, pp. 337–343, May 1977.

[16] ——, "Compression of individual sequences via variable-rate coding," *IEEE-Transactions-on-Information-Theory*, vol. IT-24, no. 5, pp. 530–536, Sept. 1978.

[17] D. Sheinwald, A. Lempel, and J. Ziv, "On encoding and decoding with two-way head machines," *Information-and-Computation*, vol. 116, no. 1, pp. 128–133, Jan. 1995.

[18] Cisco Systems Inc. (2000, Jan.) Microsoft point-to-point compression (mppc). [Online]. Available: http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113t/113t_3/mppc.htm

[19] D. Rand. (1996, June) Rfc1962: The ppp compression control protocol (ccp). [Online]. Available: http://www.ietf.org/rfc/rfc1962.txt

[20] V. Jacobson. (1990, Feb.) Compressing tcp/ip headers for low-speed serial links. [Online]. Available: http://tools.ietf.org/html?rfc=1144

[21] H. Liefke and D. Suciu, "Xmill: An efficient compressor for xml data," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 153–164.

[22] P. Tolani and R. H. Jayant, "Xgrind: A query-friendly xml compressor," in *Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE)*, 2002.

[23] (1999, June) Wap tokenization. [Online]. Available: http://www.w3.org/TR/wbxml/