Efficient sharing of dynamic WSNs

Dennis J.A. Bijwaard[2] and Paul J.M. Havinga[1,2]

[1] Pervasive Systems, University of Twente, P.O. Box 217, 7500 AE Enschede
D.Bijwaard@utwente.nl and P.J.M.Havinga@utwente.nl
[2] Ambient Systems, Colosseum 15d, 7521 PV Enschede

Abstract. The Ambient middleware supports real-time monitoring and remote maintenance across the Internet via wired and mobile wireless network access technologies. Additionally, the middleware offers easy integration with third-party applications. Ambient Studio utilizes the middleware for remote WSN configuration and monitoring. The ConnectBox utilizes it to monitor and maintain WSNs remotely. This paper describes the Ambient middleware and compares its efficiency with the existing messaging protocols used for instant messaging and web services.

1 Introduction

The Ambient middleware enables remote monitoring and maintenance of WSNs, and makes it easy to use sensor readings in customer applications. The GPRS-enabled ConnectBox allows deployment of sensor networks in moving vehicles like trucks. This enables real-time monitoring while goods are in transit. When there are temporary connection outages, the ConnectBox buffers the sensor messages and flushes them when the connection is re-established

The Ambient network [2] is self-organizing and consists of two main layers: an infrastructure layer with a Gateway and MicroRouters that relay messages across multiple hops, and a layer of SmartPoints that move through the network and in/out of networks.

This paper describes the Ambient middleware and compares its messaging efficiency.

2 Ambient middleware

The Ambient middleware enables customers to easily integrate their applications and to enable remote monitoring and maintenance. The interaction between the different components is depicted in Figure 1. Note that AmbientStudio and the ConnectBox share the same Ambient middleware (AmbientMW).

One or more Gateways can be connected via RS232 using the AmbientMW in a ConnectBox device or AmbientStudio on a PC. The ConnectBox device is an embedded Linux device that offers Ethernet connectivity towards the wireless nodes from XML applications, AmbientStudio, or other AmbientMW instantiations.

The AmbientMW offers the ConnectAPI to ease integration with third-party applications using asynchronous XML messages over a TCP/IP connection (optionally encrypted with SSL). The XML messages are the same Device Driver Interface (DDI) that are used between the nodes in the WSN, but fully parsed so they can be easily used in an application utilizing its XML schema (which enables code generation in for instance Java and C#). When required, a pass filter can be configured to reduce the type of DDI messages that are forwarded over ConnectAPI.

To offer flexibility, the ConnectAPI can be started as client and server: The server allows multiple local or remote applications to connect. The client allows connecting to a remote host, automatic re-connects, and automatic logging of messages while disconnected and flushing when the connection is re-established.

The AmbientMW also offers AmbiLink to ease remote monitoring and maintenance of sensor networks using asynchronous binary messages over optionally SSL encrypted TCP/IP connections. Similar to the ConnectAPI, both AmbiLink client and server can be started with the AmbientMW. This offers the flexibility to monitor and maintain multiple ConnectBoxes with one or more AmbientStudio instances without loosing messages when client connections are disrupted. Additionally to DDI messages, also management messages can be sent over both ConnectAPI and AmbiLink for configuring, opening and closing, serial ports and remote connections. New message types can easily be added, for instance, for fetching historical data or changing DDI message filters. Another message type could be introduced for file exchanging (for instance firmware) with the WSN, such that the WSN can use its own pace and protocol for exchanging it with the involved node(s). AmbiLink also supports merging sensor information from all connected nodes via multiple ConnectBox or AmbientStudio instances. It can then provide the merged data to multiple applications using the ConnectAPI. In both AmbiLink and ConnectAPI, message destinations can be unicast, multicast, and broadcast using wildcards in the destination of messages.

For both AmbiLink and ConnectAPI, conversion between DDI and respectively their binary and XML counterpart was automated. Logging and flushing is implemented in the middelware for both protocols in order to cache messages that cannot be sent by the client during connection outage. Server logging and flushing is not implemented, since sensor messages are usually towards a server and there is no guarantee that a client will ever reconnect to the server. To reduce message loss, the TCP connections were set up such that small messages were sent without delay and the last sent message is logged until it is possible to sent the next message. This removed the need for a special acknowledgement scheme on top of TCP (which already has its own acknowledgements), since a new message cannot be sent unless the previous one was successfully sent.

3 Messaging efficiency

In this section the efficiency of ConnectAPI and AmbiLink is compared with existing messaging protocols.

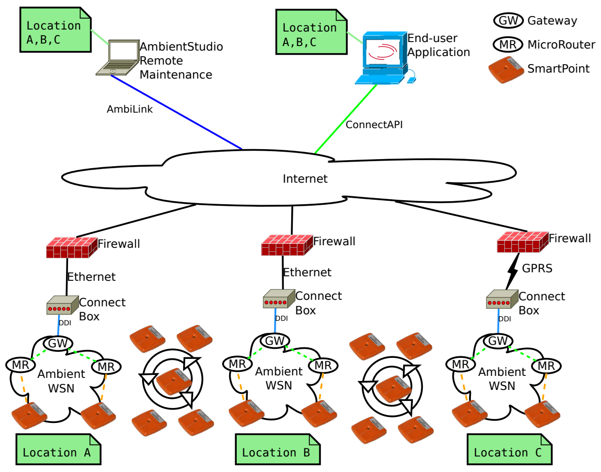

Fig. 1. Ambient connect framework

3.1 Comparing existing methods

Existing methods for messaging over the Internet are the email protocol Simple Message Transfer Protocol [12] (SMTP) for sending/receiving email, and Instant Messaging (IM) protocols like Internet Relay Chat [10] (IRC), Protocol for SYnchronous Conferencing [1] (PSYC), Session Initiation Protocol (SIP)/SIP for IM and Presence Leveraging Extensions [7] (SIMPLE) and Extensible Messaging and Presence Protocol [14] (XMPP). Also web-services like Simple Object Access Protocol [13] (SOAP) and Representational State Transfer [16] (REST), and peer to peer (P2P) messaging like P2P SIP can be used for message exchange over the internet. These messaging protocols can be used over a variety of transport protocols like TCP and UDP, and can use security protocols like Internet Protocol Security [11] (IPsec), Secure Socket Layer [5] (SSL) and Transport Layer Security [4] (TLS). Most of the protocols can also provide nomadicity (i.e. reconnection after connection loss), Mobile IP (MIP) can be used to provide seamless connectivity when switching networks. Unfortunately, MIP is not deployed in current networks and would therefore at least require a home agent and driver software on each involved computer to function.

Criteria for comparing the existing methods:

- Availability: are the required elements widely deployed, or are can they be easily deployed? Availability is positive when the protocol is generally supported in the endpoints and intermediate routers, negative when is is hardly supported on the endpoints and routers. For instance MIP and multicast are not widely deployed, application-level protocols can often be easily deployed.
- Impact: The impact is high when the routers along the path must be equipped to support the protocol (denoted as ”dr” for dedicated router), or when the

firewall must be updated to support incoming traffic (denoted as "df"). The impact is also high when dedicated clients (denoted as "dc") or a dedicated server (denoted as "ds") is required. The impact is less when a library can be used for clients (denoted as "lc"), and servers (denoted as "ls"). Using for instance XML messages, usually requires a library for parsing it.

– Latency, i.e. are messages forwarded in real-time, or are there inherent delays? For instance request-based mechanisms like web-services require higher bandwidth and processing time and double that with the required return messages.
– Reliability: is message loss prevented, or is there a mechanism to prevent losing messages?
– Reachability: can the Wireless Sensor Networks (WSN) be reached remotely when there is an Internet connection? For instance (company) firewalls often block all incoming ports and are not keen on clear-text protocols, a default Network Address Translation [15] (NAT) router blocks all incoming connections unless configured with specific forwarding rules.
– Bandwidth: can the protocol work across a limited bandwidth link such as General packet radio service (GPRS)? For instance verbose messaging like SOAP could add much overhead and other associated costs across a wireless link such as GPRS.
– Security: can others inject or obtain messages, or disrupt the service? Can the protocol easily be encrypted?

The web-service protocols eXtensible Markup Language (XML)-RPC and its successor SOAP use XML documents for messaging. REST can use both text, XML and other representations (for a request an URL could suffice). These web-services all use the request/response model of HTTP. JSON-RPC uses a compact representation and is one of the few web-service protocols that can also be used bi-directionally over a socket, i.e. it allows requests, responses and notifications to be sent asynchronously in each direction over the same connection. When behind a firewall the other protocols require either opening a firewall port, tunnelling or polling on a reachable server to receive messages (SOAP could also be used over SMTP with associated high latency, but then it would not act as a web-service). Using HTTP Secure (HTTPS) for security increases the latency of the first message, since the connection needs not only to be set up for each request but also the security association. The reliability of web-services is generally ok. Multiple libraries are available for all protocols, however there is no cross-platform C++ library available for JSON-RPC (JsonRpc-Cpp is GPLv3 licensed which requires opening all linked source when releasing). Table 2 compares the popular web-service protocols.

For messaging over the Internet, a great number of protocols exist. Only a limited number of these protocols are suitable for integration in applications (i.e. are an open standard [8]). Most of these protocols are not designed for reliability, but reachability is good for all of them since they all provide one or more ways to traverse through firewalls. The messages in these protocols are quite large because they are text-based, especially SIMPLE and XMPP. Table 3 compares the popular open messaging protocols.

3.2 Comparing Ambient middleware

The AmbiLink users binary DDI and ConnectAPI uses DDI in XML format for messaging, for both messaging is asynchronous, meaning that no response is required like in web services. When an AmbiLink or ConnectAPI client is behind a firewall, it can still reach its related server on the Internet without having to reconfigure the firewall. Both AmbiLink and ConnectAPI can be secured with SSL with the added delay of setting up the security association. The reliability of the Ambient middleware is ok, it logs and flushes messages when the connection is temporarily unavailable. AmbiLink only works as part of the Ambient middleware, ConnectAPI can be used from any program that can send XML documents over a socket. Table 4 compares the Ambient middleware protocols.

Table 5 compares the number of messages and bandwidth for a number of protocols in more detail[3]. Typical HTTP header size is 256 bytes, the size of XML and JSON documents are comparable when XML attributes are used instead of tags (else XML is about 30% larger), a typical size of such a message is 1024 bytes. A typical SOAP envelope adds 172 bytes. Typical AmbiLink binary sensor messages are approximately 250 bytes long, typical ConnectAPI messages are approximately 900 bytes long. ConnectAPI messages make heavy use of XML attributes instead of tags, which make them comparable in size to JSON messages.

The table clearly shows that the asynchronous messaging of JSON-RPC, AmbiLink and ConnectAPI saves the return-trip messaging as well as the HTTP headers. Depending on the setup of server and client, the HTTP keep-alive can keep the TCP connection open for a long time. However, usually the keep-alive timeout is less than a minute, which means more connection setups (and associated higher latency) for low-frequency messaging over HTTP. Note that the typical SOAP messages are around 1500 bytes, so a slight increase would require an additional TCP packet.

3.3 Bandwidth optimizations

The aim is to use the Ambient middleware protocols across low bandwidth links like GPRS, in which the download bandwidth varies between 9 and 52 kbit/s, and upload is usually limited to 18 kbit/s. It is envisaged that also large sites may want to use GPRS to be independent of Ethernet infrastructure which could be owned or managed by another party or simply be unavailable in a storage area. For instance 1000 nodes with 3 sensors (e.g. temperature, humidity and

[3] TCP uses 3-way handshake for setup and teardown, the set-up ACK can already contain part of the message, HTTP1.1 can use keep-alive which reduces the number of required TCP connects, TCP message header is 24 bytes, The latency of messages doubles when there is an explicit response for each message. The table assumes that each TCP message is acknowledged, where it practice the acknowledgement can be for a number of them (depending on the rate of transmission). IP header is 24 bytes

Table 1. AmbiLink versus ConnectAPI features

Protocol	Usage	Transport	Security	Format	Filter	Destination	Merged WSNs
ConnectAPI	3^{rd}-party applications	TCP/IP	SSL option	XML	header fields	Broadcast to all applications	Using multiple clients
AmbiLink	Monitoring & maintenance	TCP/IP	SSL option	binary	per WSN	Routing to AmbiLink instance(s)	At client or server

Table 2. Comparison of web service protocols

Protocol	Availability	Impact	Latency	Reachability	Bandwidth	Security
XML-RPC	+	ls+lc	medium	issues	medium	HTTPS
SOAP	+	ls+lc	medium	issues	high[9]	HTTPS
REST	+	ls+lc	medium	issues	depends	HTTPS
JSON-RPC	+/-	ls+lc	low	two-way	low/medium	SSL/TLS

Table 3. Comparison of open messaging protocols

Protocol	Availability	Impact	Latency	Reliability	Bandwidth	Security
SMTP	+	ds+dc+lc	high	+/-	medium	-
IRC	+	ds+dc+lc	low	+/-	medium	SSL
PSYC	-	ds+dc	low	+/-	medium	TLS/SSL
SIMPLE	+/-	ds+dc+lc	medium	+/-	high	TLS
XMPP	+	ds+dc+lc	medium	+/-	high	TLS

Table 4. Comparison of Ambient middleware protocols

Protocol	Availability	Impact	Latency	Reliability	Bandwidth	Security
AmbiLink	+	ds+dc	low	+	low	SSL
ConnectAPI	+	ds+dc+lc	low	+	medium	SSL

Table 5. Comparison of Bandwidth (in bytes) & latency for N message exchanges, and bandwidth for N=10

Protocol	TCP/IP headers 48 bytes	HTTP headers 256 bytes	request messages	response messages	typical message size	bandwidth N=10 and 1 TCP connect
XML-RPC	5+4N..9N	N*2	N*XML	N*XML	XML=1024	27760
SOAP	5+4N..9N	N*2	N*(envelope+XML)	N*(envelope+XML)	envelope=172	36790
REST	5+4N..9N	N*2	N*(URL|XML|other)	N*(XML|OK|other)	URL|OK=100	19000
JSON-RPC	5+2N..5+4N	0	N*JSON	N*(optional JSON)	JSON=900	10200..20160
AmbiLink	5+2N	0	N*AmbiLink	0	AmbiLink=250	3700
ConnectAPI	5+2N	0	N*ConnectAPI	0	ConnectAPI=900	10200

tilt) sending a message every 5 minutes yields an average rate of 10 messages per second[4].

For 10 AmbiLink messages per second[5] that would yield a bandwidth of 2500 bytes/s = 20 kbit/s. So, also AmbiLink could certainly use compression for bigger sensor networks over GPRS. A simple gzip[3] on a binary message gives a compression factor of 1.6 on AmbiLink messages. Compressing a group of messages, e.g. 4 at a time gives compression rate of 4, 25 at a time gives a compression rate of 8. So it would make sense to compress a group messages (e.g. all messages to be send in a second) when possible, this would also reduce the overhead on the TCP/IP level, but will increase the message latency. AmbiLink messages could also be reduced in size by shortening them or using a generic compressing on string values in these messages that are now sent as UTF-8. Huffman coding [6] would be a candidate for this, an alternative would be a look-up table for commonly used attribute names.

Sending 10 ConnectAPI messages per second would require a bandwidth of 70 kbit/s. Compression of these XML messages would thus be required for using ConnectAPI across GPRS with bigger networks. Compression with gzip of a temperature message achieves a compression factor of 1.8. Compressing a group of 4 messages yields a compression factor of 3, compressing a group of 25 messages yields a compression factor of 18. Some more can be saved by stripping redundant information from the ConnectAPI messages and shortening the XML tag and attribute names. A large part of these attribute and tag names come from the DDI descriptors, so shortening them in these descriptors will reduce the bandwidth.

4 Conclusion

The Ambient middleware utilizes the DDI framework that allows any resource in the system to be configured and accessed remotely. This paper compared the efficiency of the middleware messaging with existing methods and describes how it can be further improved.

References

1. Psyc instant messaging. `http://about.psyc.eu/`, Last visited March 2011.
2. Ambient systems. `http://ambient-systems.net`, Last visited August 2011.
3. P. Deutsch. GZIP file format specification version 4.3. RFC 1952, Internet Engineering Task Force, May 1996.

[4] note that these message rates are only required when full sensing history is required, else it is more practical to configure alarms in the SmartPoint on specific sensing conditions

[5] The amount of messaging depends on the number of SmartPoints, its reporting period or alarm thresholds, and scale of the network (current maximum is 64 infrastructure nodes)

4. T. Dierks. The transport layer security (tls) protocol version 1.2. RFC 5246, Internet Engineering Task Force, Aug. 2008.
5. A. O. Freier, P. Karlton, and P. C. Kocher. The ssl protocol version 3.0. `http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt`.
6. D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proc. IRE 40*, pages 1098–1101, 1952.
7. IETF. The simple working group charter. `http://datatracker.ietf.org/wg/simple/charter/`.
8. ITU-T. Open standard. `http://www.itu.int/en/ITU-T/ipr/Pages/open.aspx`.
9. M. B. Juric, I. Rozman, B. Brumen, M. Colnaric, and M. Hericko. Comparison of performance of web services, ws-security, rmi, and rmi-ssl. *Journal of Systems and Software*, 79(5):689 – 700, 2006. Quality Software.
10. W. Kantrowitz. Network questionnaires. RFC 459, Internet Engineering Task Force, Feb. 1973.
11. S. Kent and K. Seo. Security architecture for the internet protocol. RFC 4301, Internet Engineering Task Force, Dec. 2005.
12. J. Klensin. Simple mail transfer protocol. RFC 5321, Internet Engineering Task Force, Oct. 2008.
13. N. Mitra and Y. Lafon. Soap specificiations. `http://www.w3.org/TR/soap/`.
14. P. Saint-Andre. Extensible messaging and presence protocol (XMPP): core. RFC 3920, IETF, Oct. 2004.
15. P. Srisuresh and M. Holdrege. IP network address translator (NAT) terminology and considerations. RFC 2663, Internet Engineering Task Force, Aug. 1999.
16. S. Tilkov. Introduction to rest. `http://www.infoq.com/articles/rest-introduction`.