Evaluation of hardware requirements for device management of constrained nodes based on the LWM2M standard

Paul Schmelzer[1,2] and Jens-Peter Akelbein[1,3]

[1] University of Applied Sciences, Darmstadt, Germany
[2] Research Assistant, paul.schmelzer@h-da.de
[3] Professor for Computer and Software Engineering, jens-peter.akelbein@h-da.de

Abstract. Nowadays, resource constrained devices are seen as one of the main components for building the Internet of Things (IoT). Typical constraints of such devices are given by the size of the built-in memory, a non-continuous power supply, and limited computing power. A categorization for such constraints into classes is given in RFC7228. The exponential growth of the number of end devices demand for a centralized, automated, and secure device management to handle the complexity of the upcoming IoT. The emerging open standard Lightweight Machine to Machine (LWM2M) is one of the first approaches to meet the requirements for managing constrained devices. A growing variety of implementations of the LWM2M protocol stack became available in the past years. This paper aims to define useful metrics for measuring device management capabilities on constrained nodes. Two relevant open source implementations are compared by their memory usage. The results show that LWM2M is feasible on Class 2 devices.

Keywords: IoT · Device Management · LWM2M

1 Introduction

According to Gartner and other analysts the amount of IoT devices in consumer and business environments doubles each two years. Such an increase of interconnected devices forming complex systems with broader attack surfaces demands for an appropriate security level of communication protocols and firmware for IoT devices. These new requirements ask for a new infrastructure for managing devices in an automated and centralized manner. This approach is well known for interconnected devices like personal computers, mobile phones, printers, etc. In the field of constrained devices such as small sensor nodes for measuring temperature, humidity, pressure etc. or actuators like thermostats or light switches, device management is not yet widespread [1]. Microcontroller for such applications are available since a few years providing hardware accelerations for cryptography as well. Furthermore, wireless connectivity for sensor nodes is developing towards a standardized interoperability like given by the IEEE 802.15.4 standard. In a wireless sensor network (WSN) devices often interact directly

with their environment. This leads to new risks especially in safety-critical applications like smoke detectors. So with introducing interconnectivity such small devices must become maintainable with Firmware Over-the-Air (FOTA). In addition, for detecting misbehaviour a WSN requires monitoring capabilities and should provide remote parameter configuration. This paper focuses on functionality for security purposes in particular. The LWM2M protocol developed by the Open Mobile Alliance (OMA) is an example for how to standardize open device management [2]. With its focus on constrained devices LWM2M provides a variety of services besides device management and is build on top of IP. In the last years several open implementations for LWM2M clients came up. They differ in a large variety in their memory consumption and implemented functionality. So for selecting an appropriate implementation an upfront analysis is necessary whether functional requirements are met and hardware requirements can be fulfilled by a given hardware device.

2 Device management and its implementation

Before personal computers got interconnected through a network, management functionality was rarely required. Each user had to manage its personal device manually and physical access to the device was required. By interconnecting such devices, large enterprises introduced a centralized device management enabling remote control over a continuously growing number of computers. In general, such management consists of configuration, monitoring and administration of managed entities. Managed entities could be network elements, applications, system resources or services [3]. In the 1990s, management was divided into three categories: system management, network management and application management. With the emergence of smart devices the new domain "device management" appeared. Device management consists of functions from all three categories. Furthermore, it is not only used in a business context, but for private purposes by individual users as well [3].

Comparing common management protocols, the way of managing entities is very similar [4]. Managed entities are named resources or objects. They consist of a name and a corresponding value representing a unique ID. Resources and objects are often sorted into logical groups (e.g. a group named "temperature sensor" with two objects: battery level and temperature). Similar types of operations exist that can be performed on objects. Primarily those are: *GET* the value of the object or resource, *SET* the value of the object or resource, *EXECUTE* a specific function provided by the object or resource, *NOTIFY* a central management point about events being observed by a specific object or resource.

LWM2M implementations: As of today, the LWM2M standard has a very low market penetration. For now there are a couple of open implementations already available. Some of them are part of IoT operating systems like MBED, RIOT, Contiki or Zephyr. Others are maintained by companies [5, 6]. LWM2M uses a client-server architecture while this paper investigates clients only. An

LWM2M server requires a more powerful machine where resource constraints should not be an obstacle.

Related Work: The literature provides already some work on evaluating device management on constrained nodes. Z. Sheng et al. presented an efficient way minimizing packet size with Constrained Application Protocol (CoAP) [7]. Others implemented their own LWM2M client for evaluation patterns of memory usage and network load [8, 9]. D. Tracey et al. used Contiki for expanding LWM2M by new objects for a comparison with the Common Information Model (CIM) as a standard being used in enterprise-wide management solutions [10]. The results show that LWM2M manages data in a more efficient way. Furthermore there had been attempts to connect Bluetooth Low Energy (BLE) networks to LWM2M via a Gateway mapping the BLE service onto LWM2M objects [11]. According to the authors, this approach was taken because IP isn't still suitable on all constrained devices. A very similar approach was taken by [12] who integrated LWM2M into the Continua architecture for medical remote patient services. This was also realised by a Gateway mapping medical data (e.g. blood pressure, smart watch) to LWM2M Objects. To run LWM2M on more constrained devices directly without the needs for a gateway in between A. Karaagac et al. presented several optimizations in the LWM2M communication flow [13]. Improving security and power supply on constrained devices is an active field of research [14–17]. In the future IoT a centralized interoperable device management is key [18].

3 Evaluation criteria and methodology

Devices for large IoT environments bring their own non-functional requirements. When the number of devices grows, manual workload for changing batteries after one or a few years drive operational cost becoming a strong inhibitor for new business models based on such large IoT environments. Hence their energy consumption should allow periods of 5, 10, or even more years of running time. Another criteria is on limiting the required size of memory to a minimum for using the cheapest hardware configuration of microcontroller variants. Both energy consumption and memory size depend on the hardware architecture and a modest resource footprint of the used software. The following list of non-functional requirements should be used for assessing the eligibility of network protocols and their implementations in resource constrained environments.

Memory usage: Several software components like an operating system, IP stack, etc. build the software stack running on constrained devices. Device management adds further components to the stack. Its code is stored in ROM/Flash where the size can be determined after compilation time. The size of the data segments define the size of required RAM. It is used for variables, buffers etc. where only the size of the static part of memory can be determined after compilation time exactly. During runtime the size of dynamically allocated memory like a heap needs to be added. For constrained devices, an implementation should achieve that only a minimal part of RAM is being used dynamically.

Energy usage: For a very low energy consumption constrained devices should be in a power saving or sleeping mode most of the time. Both optimizing the software stack behaviour for maximizing the deep sleep time and measuring the energy consumption per software function is an active research area [19].

Protocol overhead: Reducing the overhead of communication protocols for constrained node networks (CNN) can be achieved by minimizing the size of messages and the number of messages to be sent. Readable formats like XML or JSON use much more bytes as a consequence compared to binary formats. Corresponding metrics allow measuring the efficiency of the complete network stack or layers of it.

Encryption: Public key infrastructures (PKI) are hardly implementable even for class 2 devices. Current open approaches are based on DTLS with PSK or raw public keys. If weaker encryption is sufficient there are a few lightweight cipher suites choosable [20]. Furthermore, hardware accelerated cryptography primitives can be used. However, for FOTA, secure multicasting would be desirable [21].

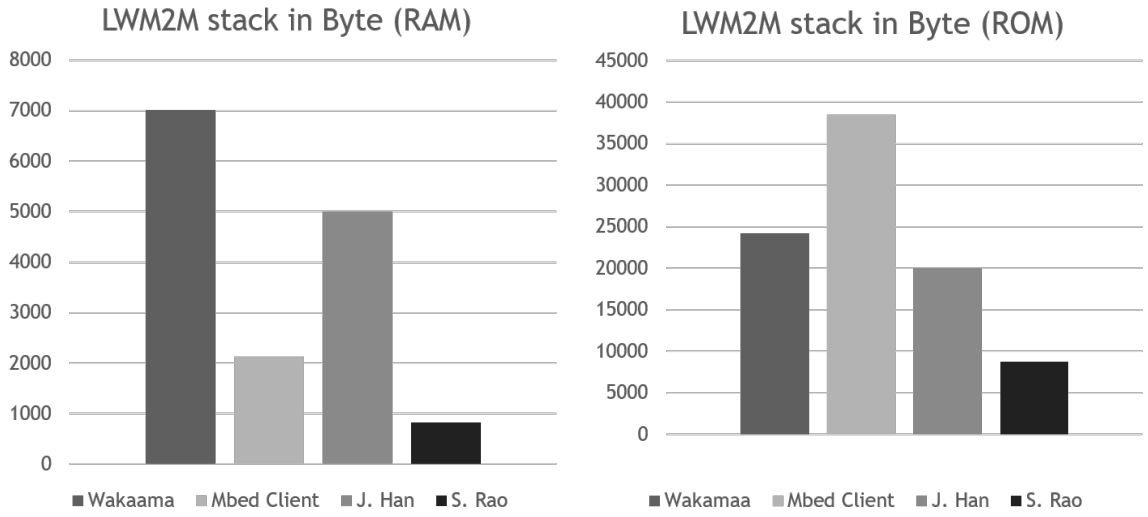
Modularity: For maintaining a software stack over a long lifetime, the implementation should be modularized. Configurability allows providing only the required code parts within the shipped firmware.

As the following results were gained in an ongoing project this paper presents first results investigating memory usage. For determining the memory footprint a tool from G. Mukundan [22] was used. It analyses firmware in executable format from the *Map* and *ELF* file. For the investigation, sample firmware applications were created where a Nordic nRF52-DK board was chosen for the evaluation as a typical hardware used for constrained nodes.

4 Results

In total, four LWM2M implementations were selected for this examination. Data for two of them were already provided by Han et al. [8] and Rao et al. [9]. For investigating the other two, according components were extracted from the Mbed stack (named *mbed client*) and RIOT OS (named *wakaama*). They were ported and integrated as a sample firmware for chosen hardware board. Figure 1 presents a comparison of the memory usage for both RAM and ROM. S. Rao's implementation results in very low consumption of 820 Byte RAM and 8764 Byte ROM without the mandatory LWM2M objects and DTLS. The mbed client results in an exceptionally high ROM usage of 38450 Byte. One reason seems to be the overhead generated by using a highly object oriented C++ design with a large code base while the other three are implemented in C. The mbed client provides all LWM2M capabilities. For porting it to another software stack easily a platform abstraction layer is provided also resulting in ROM overhead. The mbed client is designed to connect to the ARM's native cloud platform only. A few changes in the code were necessary for allowing connections to an open source LWM2M server like *Leshan*. The RAM consumption of the mbed client is modest with only about 2130 Byte. Wakaama was developed by the Eclipse Foundation.

The implementation requires a lot of dynamic memory. Such RAM usage is not recommended for embedded programming. Nevertheless, the code was ported to RIOT OS. The ROM usage results in 24250 Byte which is closer to the size cited by Han. Wakaama provides DTLS security with PSK and raw public key, certificates are not implemented. Depending on the used DTLS library and its modularity, security adds around 4-8 KB in RAM and 25-60 KB in ROM. The RAM and ROM size also depends on which LWM2M capabilities are already provided by an implementation. This may differ to a small extend between the presented implementations. The RAM usage of Wakaama with 7010 Byte is still beyond design objectives for class 2 devices.



(a) RAM (b) ROM

Fig. 1. LWM2M implementations in Byte

As a sort of hacky workaround for the port, around 5 KB of memory are allocated statically for simulating the previous dynamic memory design on the heap. This results in memory leaks by assigning static blocks bigger than the containing data. The memory actually been used by the wakaama code is less than the reserved memory. To fix this issue it would be advisable to allocate only the utilized memory for the used objects and identify the parts where dynamic memory is needed. Furthermore, on RIOT the occupied stack size on the nRF52-DK could be measured. It was about 1,2 KB.

Additional Results: LWM2M suggests the use of *TLS_PSK_WITH_AES_128_CCM_8* or *TLS_EC-DHE_PSK_WITH_AES_128_CCM_8* as cipher suites. These should suffice security needs of interconnected constraint devices according to RFC 7925 [20]. For LWM2M connections via the internet containing sensitive data an 8 Byte MAC is probably too weak. Comparing modularity, Wakaama provides only very limited configuration options for selecting code parts to be shipped. In contrast, the mbed client can be customized by a configuration file

for choosing modules to be contained within the firmware e.g. cipher suites, IPv4/IPv6, LWM2M bootstrapping etc.

5 Conclusion and Future Work

For using IoT networks including large numbers of constrained devices manageability of all components becomes a strong requirement. Therefore, device management on constrained nodes needs to be seen as essential for building secure and mature IoT solutions soon. As hardware resources on these nodes are limited, appropiate software components for implementing device management need to fulfil non-functional requirements on memory and CPU consumption as well. LWM2M is an emerging standard in the IoT for managing constrained devices. Because these devices are restricted, there are special requirements for software and hardware. The important metrics are memory usage, energy usage, protocol overhead, encryption and modularity. The presented implementations vary especially on there memory usage and functionality. Although it has a big ROM capacity, the mbed client contains all standard functionality and has a well rounded C++ API. Wakaama on RIOT can still be optimized in its RAM usage compared to other implementations. Both libraries are maintained actively. The amount of memory usage justifies the feasibility on Class 2 devices.

For future work it would be worth considering on how to validate the presented requirements for an application protocol. This paper presents first results of evaluating the memory footprint as part of a research project. Furthermore, questions need to be answered how to optimize functional coverage versus memory footprint, how RAM usage can be minimized, and energy consumption can be reduced during runtime. Another drawback of current implementations is that LWM2M clients have to wait for server requests actively. Constrained nodes are usually active for sending sensor data only while sleeping most of their time. Keeping the radio module on for receiving requests leads to unnecessary power consumption. So adding a timing schema with active slots for managing nodes will be another optimization.

References

1. Bormann, C., Ersue, M., Keranen, A.: Terminology for Constrained-Node Networks. IETF, RFC 7228 (2014)
2. Open Mobile Alliance: Lightweight Machine to Machine Technical Specification (2018)
3. Gürgen, L., Honiden, S.: Management of Networked Sensing Devices. In: Tenth International Conference on Mobile Data Management: Systems, Services and Middleware. pp. 502–507 (2009)
4. Ghetie, I.G.: Networks and Systems Management: Platforms Analysis and Evaluation. Springer US (2012)
5. AVSystem: Anja. https://github.com/AVSystem/Anjay (2018), last accessed 9 Aug 2018

6. Foundry, C.: Awa LWM2M. https://github.com/ConnectivityFoundry/AwaLWM2M, last accessed 9 Aug 2018
7. Recent Advances in Industrial Wireless Sensor Networks Toward Efficient Management in IoT. IEEE Access **3**, 622–637 (2015)
8. Han, J., Ha, M., Kim, D.: Practical security analysis for the constrained node networks: Focusing on the DTLS protocol. In: 5th International Conference on the IOT. pp. 22–29 (2015)
9. Rao, S., Chendanda, D., Deshpande, C., Lakkundi, V.: Implementing LWM2M in constrained IoT devices. In: ICWiSe. pp. 52–57 (2015)
10. Tracey, D., Sreenan, C.: OMA LWM2M in a holistic architecture for the Internet of Things. In: IEEE 14th ICNSC. pp. 198–203 (2017)
11. Ha, M., Lindh, T.: Enabling Dynamic and Lightweight Management of Distributed Bluetooth Low Energy Devices. In: ICNC. pp. 620–624 (2018)
12. Li, M., Moll, E., Chituc, C.: IoT for Healthcare: An architecture and prototype implementation for the remote e-health device management using Continua and LwM2M protocols. In: 44th Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 2920–2926 (2018)
13. Karaagac, A., VanEeghem, M.: Extensions to LwM2M for Intermittent Connectivity and Improved Efficiency. In: IEEE Conference on Standards for Communications and Networking (CSCN). pp. 1–6 (2018)
14. Lee, H., Lee, K., Kim, H.: Wireless Information and Power Exchange for Energy-Constrained Device-to-Device Communications. IEEE Internet of Things Journal **5**, pp. 3175–3185 (2018)
15. Yoon, S., Kim, J.: Remote security management server for IoT devices. In: International Conference on Information and Communication Technology Convergence (ICTC). pp. 1162–1164 (2017)
16. Aditia, K., Altaf, F., Singh, M.: Optimized CL-PKE with Lightweight Encryption for Resource Constrained Devices. In: Proceedings of the 20th International Conference on Distributed Computing and Networking. pp. 427–432 (2019)
17. Lustro, F., Sison, M., Medina, P.: Performance Analysis of Enhanced SPECK Algorithm. In: Proceedings of the 4th International Conference on Industrial and Business Engineering. pp. 256–264 (2018)
18. Jin, W., Kim, D.: IoT device management architecture based on proxy. In: 6th International Conference on Computer Science and Network Technology (ICCSNT). pp. 84–87 (2017)
19. Pötsch, A., Berger, A., Springer, A.: Efficient analysis of power consumption behaviour of embedded wireless IoT systems. In: IEEE International Instrumentation and Measurement Technology Conference (I2MTC). pp. 1–6 (2017)
20. Tschofenig, H., Fossati, T.: Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things. IETF, RFC 7925 (2016)
21. Keoh, S., Kumar, S.: DTLS-based Multicast Security in Constrained Environments, draft-keoh-dice-multicast-security-08 (2015), IETF, Internet Draft
22. Govind Mukundan: Analyzing the Linker Map file with a little help form the ELF and the DWARF. https://www.embeddedrelated.com/showarticle/900.php (2015), last accessed 10 Aug 2018