Protocol Profiles for Constrained Devices

Jürgen Schönwälder (Jacobs University, Germany)
Tina Tsou (Huawei Technologies, USA)
Behcet Sarikaya (Huawei Technologies, USA)

February 11, 2011

1 Introduction

In certain application scenarios, there is a range of devices involved with very different capabilities. In the core of the Smart Grid, one will find full fledged servers. Close to metering points, one will more likely see embedded PC-like devices that still offer plenty of resources when it comes to the implementation of network protocols. Within industrial settings or households, we will likely find so called constrained devices consisting of 8-bit or 16-bit microcontrollers with wireless network interfaces such as 802.15.4 radios [1]. In scenarios involving a range of very different devices, there is usually a desire to restrict the set of different protocols used to a certain subset that can scale across device types. Experience with protocol gateways translating between protocols providing similar services tells us that such gateways can cause nasty operational problems since protocol semantics are often not 100% translatable in certain corner cases. Hence, it is often desirable to implement multiple protocols even on constrained devices since this approach allows easier integration with existing software components and a reduction of complexity (no protocol gateways needed), which in turn usually leads to a reduction of operational costs. Of course, in application scenarios where only constrained devices are used (or they clearly dominate), the development of protocols optimized to the specific scenario reduces device complexity and may be desirable.

In this position statement, we argue that there should be a choice and that both the development of a single streamlined protocol interfacing to applications via gateways as well as the definition of profiles of existing protocols for constrained devices is desirable. We also petition for a discussion that is grounded on implementation experiences and takes a system perspective instead of discussing protocols in isolation. For example, application protocols running over UDP may need to implement their own retransmission logic while protocols running over TCP essentially share the retransmission logic. Similarly, protocols sharing the same security mechanisms will be more resource friendly than protocols assuming different security solutions. In other words, in order to understand the resource requirements of a given protocol on constrained devices, it is necessary to also understand the resource requirements of related protocols and the operating system functions the protocol depends on.

In the following, we outline a possible multi-protocol approach for constrained devices currently prototyped at Jacobs University. We provide numbers concerning the memory usage of protocol implementations integrated into the Contiki [2] operating system. While the absolute numbers depend on many platform specific factors and can thus be debated almost endlessly, the focus should be on the relative size of the numbers we provide. A discussion should also take into account that the processing and memory capabilities of so called constrained devices have changed over time and continue to change. And the same holds true for wireless technology. While the current 802.15.4 technology, with its severe restrictions in frame sizes and data rates, is commonly assumed as the obvious choice today, it is not unlikely that in a few years other wireless technologies such as low-power WiFi or an extended version of 802.15.4 may offer an attractive alternative, avoiding the need of adaptation layers such as 6LoWPAN [3].

2 Multi-Protocol Architecture

The protocol stack currently being prototyped at Jacobs University is shown in Figure 1. We use the UDP/TCP/IPv6/6LoWPAN implementations available in Contiki [4]. Our target platform is the AVR Raven, which provides as the primary processing unit an 8-bit ATMega128P microcontroller with 16KB of RAM and 128 KB of flash memory. The Contiki operating system compiled with the 6LoWPAN implementation, including TCP, UDP, and HTTP, takes up 6KB of RAM and 35 KB of ROM, leaving about 10 KB of RAM and 93 KB of ROM for any other programs.

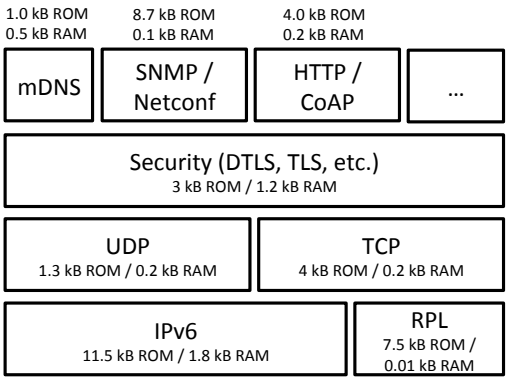

Figure 1: The protocol stack overview and the memory consumption of each component on the AVR Raven platform using the Contiki OS.

While we believe that most constrained devices will not be RPL [5] routers but rather participate as nodes interfacing with RPL routers that have additional resources, we mention the size of the Contiki RPL implementation for completeness in Figure 1. Security protocols and in particular the cryptographic functions consume quite some code space and have profound impact on the run-time CPU usage. As a consequence, we believe that it is crucial that the security layer is shared by all application protocols. For constrained devices, we envision a streamlined DTLS/TLS layer, working only with pre-shared keys (omitting public key cryptography and certificate handling).

During our experiments, we found that service discovery is of crucial importance. It is necessary to easily discover smart objects that may appear and disappear dynamically and at the same time it is crucial that smart objects discover the network services they can use. We believe that multicast DNS (mDNS) [6] fits the requirements well, is already widely implemented and deployed, and is relatively easy to implement. For event notification and logging, we make use of SYSLOG [7]. Due to the simplicity of the SYSLOG protocol, we did not highlight it in Figure 1. For the configuration and monitoring interface, we are experimenting with a lightweight version of SNMP [8, 9] (leaving out notifications since we prefer to use SYSLOG) and a lightweight version of NETCONF [10]. Of course, only one of them will be present on a production implementation but our goal is to understand the implementation and run-time costs of both protocols. In the case of NETCONF, we leave out protocol features like subtree filtering and the edit-config operation since these features are likely not needed on constrained devices with a limited number of configurable items. Our current setup utilizes the Contiki built-in HTTP server and we assume that CoAP [11] is of roughly the same complexity as the subset of HTTP implemented in Contiki today.

The memory usage values presented in this figure are either obtained through experimentation or they are previously published numbers. The ROM and RAM usage values for SNMP / NETCONF, HTTP / CoAP, Security and RPL are obtained by compiling Contiki with and without these and then comparing the values. It is important to note that the values presented for the security layer do not take into account full DTLS and TLS, but only pre-shared key authentication using AES and MD5 hashing. Previous implementation experience indicates that a complete implementation of DTLS or TLS is not likely to take up much more memory than already being used by the cryptographic functions, since these are computationally expensive and code intensive operations. The memory consumption values for UDP, TCP and IPv6 are obtained from previous research on this topic [4]. Unlike others, memory consumption values indicated for mDNS are based upon memory usage for other similar applications, since the mDNS agent is still under development[1].

[1] Implementation of mDNS, DTLS and NETCONF is likely to be finished by the beginning of May 2011.

3 Example: SNMP on Constrained Devices

The Contiki SNMP agent implements the SNMPv1 and SNMPv3 message processing models but supports only the `Get`, `GetNext` and `Set` operations in order to ensure that it would fit within the constraints of an AVR Raven. The USM security model has been implemented and supports the HMAC-MD5-96 authentication and CFB128-AES-128 symmetric encryption protocols. Management data are accessed via the MIB C code module, which provides an interface to define and configure accessible managed objects.

In order to optimize SNMP to fit within the platform limitations of the platform, the modular architecure defined for SNMP [12] is not exactly followed. Simplifications are made by modifying the abstract service interfaces to minimize resource consumption. For testing purposes, the `SNMPv2-MIB`, `IF-MIB` and `ENTITY-SENSOR-MIB` modules have been implemented for the AVR Raven board platform.

The implementation has been tested for interoperability with the snmpget, snmpgetnext, snmpset and snmpwalk applications from the Net-SNMP suite. In order to obtain the memory usage of just the SNMP agent, and not the entire OS, we first obtain the memory footprint of the OS without the agent and then subtract this value from the memory usage with the SNMP agent. The full SNMP implementation uses 31220 bytes of ROM, which is around 24% of the available ROM on the AVR Raven, and 235 bytes of statically allocated RAM. In case SNMPv1 is only enabled, the agent uses 8860 bytes of ROM (about 7% of the available ROM) and 43 bytes of statically allocated RAM.

The memory usage of individual components of the SNMP agent were also calculated. The AES and MD5 implementations constitute around 31% and 33% respectively of the agent code size. Almost half of the ROM occupied by the AES implementation is used to store constants. The MD5 implementation intensively uses macro definitions for transformations which results in a huge code size. Using functions instead could reduce the code size, but would also effect the performance. It is also worth mentioning that the cryptographic primitives were ported from the OpenSSL library and are not optimized for embedded platforms. The USM security model occupies almost half of the agent statically allocated RAM. This RAM is used to store localized keys and OIDs of the error indication counters.

Version	Security Mode	Max Stack Size
v1	-	688
v3	noAuthNoPriv	708
v3	authNoPriv	1140
v3	authPriv	1144

Table 1: The experimental results for the maximum stack size (in bytes)

Measuring the stack size is more challenging since it changes dynamically. As such, an experimental approach was adopted to estimate the stack size used by the agent to process a request. Upon receipt of an incoming SNMP message, the memory region allocated to the program stack is filled with a specific bit pattern. When the processing has been finished, the stack is examined to see how much of it is overwritten. Table 1 presents the maximum stack size observed during experiments for different versions of SNMP and security modes used. Most of the stack is occupied by the response message buffer of 484 bytes. The SNMPv1 and SNMPv3 message processing models with noAuthNoPriv security level use approximately the same stack size, which constitutes around 4% of the available RAM. When authentication and privacy are enabled, the stack grows by about 66% up to 1144 bytes, which is about 7% of RAM on the targeted platform.

The time taken for transferring and processing a single SNMP request is shown in Figure 2. All experiments were done for requests with one variable binding referring to the same MIB object. The first observation is that the time spent in the SNMP request processing is small relative to that spent in data transfer for SNMPv1 and SNMPv3 in the `noAuthNoPriv` mode. It constitutes only around 6-7% of the total latency. However, enabling authentication and privacy significantly increases this metric, by almost 228% in the worst case. It is also interesting to note that encryption adds an overhead of about 21%, whereas authentication costs 58% more compared to no authentication and no encryption.

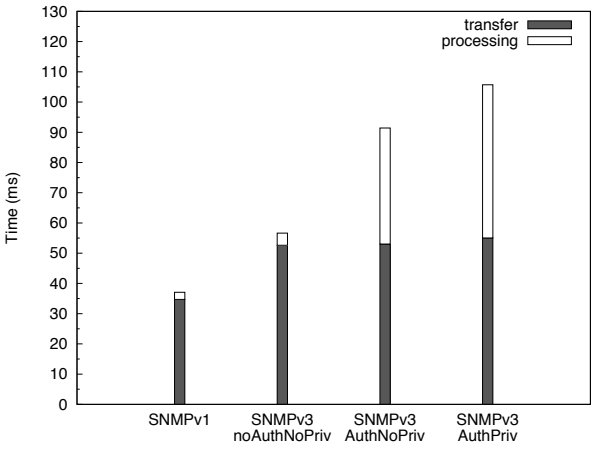

Figure 2: Time taken for transferring and processing an SNMP request

4 Conclusions

In certain application areas, it will be desirable to utilize protocols that can scale across different types of devices since this reduces software integration costs, avoids protocol gateways, and leads to a reduction of operational costs. Concerning the protocol work done in the IETF (or IRTF), we consider two items particularly useful targets to work on:

- Definition of a common lightweight security layer (e.g., based on DTLS/TLS) that can be utilized by all protocols. The idea is to agree on a core set of security protocol features that are commonly implemented and provide interoperability of the security mechanisms.
- Development of protocol profiles for existing protocols that define a subset of protocol features to be implemented on constrained devices. The profiles should enable to scale the usage of a protocol across a large range of different devices.
- Concerning the management and monitoring aspects of constrained devices, there is a need to define the instrumentation that needs to be made available to management protocol interfaces. Again, it is crucial that the instrumentation can be shared with whatever protocol is selected to carry the information.

References

[1] IEEE. IEEE Std 802.15.4-2003. Technical Report 802.15.4-2003, IEEE, October 2003.

[2] A. Dunkels, B. Gronvall, and T. Voigt. Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proc. 29th IEEE International Conference on Local Computer Networks (LCN'04)*, 2004.

[3] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, Microsoft Corporation, Intel Corp, Arch Rock Corp, September 2007.

[4] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels. Making Sensor Networks IPv6 Ready. In *Proc. of the 6th Conference on Networked Embedded Sensor Systems (SenSys 2008)*, Raleigh, North Carolina, USA, November 2008. ACM.

[5] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and JP. Vasseur. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. Internet-Draft (work in progress) <draft-ietf-roll-rpl-18>, Cisco Systems, Sigma Designs, LIX, Arch Rock Corporation, Ember Corporation, Stanford University, Dust Networks, February 2011.

[6] S. Cheshire and M. Krochmal. Multicast DNS. Internet Draft <draft-cheshire-dnsext-multicastdns-13.txt>, Apple, January 2011.

[7] R. Gerhards. The Syslog Protocol. RFC 5424, Adiscon GmbH, March 2009.

[8] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and Applicability Statements for Internet Standard Management Framework. RFC 3410, SNMP Research, Network Associates Laboratories, Ericsson, December 2002.

[9] J. Schönwälder, H. Mukhtar, S. Joo, and K. Kim. SNMP Optimizations for Constrained Devices. Internet Draft <draft-hamid-6lowpan-snmp-optimizations-03.txt>, Jacobs University, ETRI, Ajou University, October 2010.

[10] R. Enns. NETCONF Configuration Protocol. RFC 4741, Juniper Networks, December 2006.

[11] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. Constrained Application Protocol (CoAP). Internet-Draft (work in progress) <draft-ietf-core-coap-04>, Sensinode, Universitaet Bremen TZI, SkyFoundry, January 2011.

[12] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411, Enterasys Networks, BMC Software, Lucent Technologies, December 2002.