Management of Resource Constrained Devices in the Internet of Things

Anuj Sehgal, Vladislav Perelman, Siarhei Kuryla, Jürgen Schönwälder

***Abstract*—The embedded computing devices deployed within the Internet of Things (IoT) are expected to be resource constrained. This resource constraint not only applies to memory and processing capabilities, but the low-power radio standards utilized further constrain the network interfaces. The IPv6 protocol provides a suitable basis for interoperability in the IoT, due to its large address space and a number of existing protocols that function over IP and its flexibility. We investigate how existing IP based network management protocols can be implemented on resource constrained devices. We present the resource requirements for SNMP and NETCONF on an 8-bit AVR based device.**

I. Introduction

The appearance of cheap embedded computing devices capable of wireless communications is leading to the emergence of an Internet of Things (IoT). The ability to network embedded devices opens up opportunities to develop new applications. From home automation to an energy balanced smart electricity grid, the IoT is expected to introduce new computing services that integrate existing software services already available on the Internet with the control and data-gathering capabilities of embedded devices.

IoT devices are likely to be deployed in large numbers in highly competitive markets. Technology improvements following Moore's law will most likely be used to make embedded devices cheaper, smaller and more energy efficient but not necessarily more powerful. Typical embedded IoT devices are equipped with 8- or 16-bit microcontrollers that possess very little RAM and storage capacities. Resource constrained devices are often equipped with an IEEE 802.15.4 radio, which enables low-power low-data-rate wireless personal area networks (WPANs) with data rates of 20-250 kbps and frame sizes of up to 127 octets. The IETF developed the 6LoWPAN [1] standard, which is an adaptation layer enabling the exchange of IPv6 packets over IEEE 802.15.4 links.

Management of resource constrained networks using IP-based protocols is, as such, important to the development of the IoT. The SNMP [2] and NETCONF [3], [4] protocols were chosen as the two network management protocols of interest to be implemented in our study. Both protocols have their own unique approach to network management and implementing both provides us the ability to compare resource utilization for the two different network management protocols.

In the following sections, we first provide an overview of resource constrained hardware in order to understand the specific limitations that must be contended with. We then present details on the development environment we utilized for implementing the SNMP and NETCONF management protocols and their security mechanisms. We conclude the paper with a discussion of the experiences of implementing these protocols on resource constrained devices.

II. Constrained Devices

Embedded devices used in the IoT need to possess computational capabilities for the task they have to perform and networking abilities allowing integration with the Internet. To minimize product costs, IoT devices are likely to be equipped with low-power embedded computational devices. Table I presents an overview of a few typical low-power constrained devices.

Table I
An overview of different low-power constrained devices.

Type	CPU	RAM	Flash/ROM
Crossbow TelosB	16-Bit MSP430	10 KB	48 KB
RedBee EconoTAG	32-Bit MC13224v	96 KB	128 KB
Atmel AVR Raven	8-Bit ATMega1284P	16 KB	128 KB
Crossbow Mica2	8-Bit ATMega 128L	4 KB	128 KB

From Table I it becomes clear that these resource constrained devices do not provide much storage or memory capabilities. While the RedBee EconoTAG provides 96 KB of RAM, its execution model does not allow for all of this to be used for storing data since before execution, the contents of flash memory (excluding the bootloader) must be copied to RAM.

The memory constraints make it important that any networking technologies are built with resource constrained devices in mind. Therefore, it is desirable to identify a minimal IP-based protocol set that can be effectively used to manage the IoT. It is also important to analyze the bare-minimum feature-set that must be kept in order for protocols to still be identifiable as themselves and usable with existing tools, thereby enabling true interoperability, while still being functional on resource constrained devices.

We attempt to identify the minimum resources required on prospective IoT devices, by implementing different IP-based management and security protocols on resource constrained devices.

III. Development Environment

In order to implement and test a set of management protocols for resource constrained devices it is important to choose a suitable device and operating system. The operating system (OS) of choice must not only provide a 6LoWPAN implementation to enable IPv6 connectivity over IEEE 802.15.4 radios,

but it must also support UDP and TCP for the implementation of management protocols.

A few different embedded OSs can be chosen to support these requirements, amongst which are Contiki [5], TinyOS [6], Ethersex [7] and many Embedded Linux options. Contiki is an open source, highly portable, multi-tasking operating system for embedded devices that has been ported to multiple devices. Contiki has plenty of hardware drivers for each supported device thereby reducing the development time and providing some level of code portability. Furthermore, the μIPv6 stack used by Contiki supports the 6LoWPAN standard. TinyOS is an OS designed for use in embedded networks, with good support for the relevant protocols. At the time we started our implementation efforts, Contiki appeared to have a more active developer community and better support for IPv6. Embedded OSs like Ethersex, are suitable for simple embedded networking projects but they lack a mature networking stack with support for IEEE 802.15.4. On the other hand, Embedded Linux would be a suitable choice given the flexibility it provides, however, the memory requirement is in the order of 1 MB.

The Atmel AVR Raven was chosen as the target device since Contiki has good support for it. It is also quite representative of resource constrained network devices (Table I). The AVR Raven consists of two parts, the Raven board itself and the RZUSB stick, which is used to interface the IEEE 802.15.4 based network with regular IP networks. The RZUSB stick masquerades as a USB Ethernet interface when plugged into a computer. This USB Ethernet interface can then be used to access IEEE 802.15.4 networks. The AVR Raven board hosts two microcontrollers. The ATMega1284P is used as the primary microcontroller on which all the processing and interfacing with the radio, an AT86RF230, occurs. The second microcontroller, an ATMega3290P, is almost exclusively used to control an on-board LCD.

A barebone installation of Contiki, supporting 6LoWPAN TCP, UDP and HTTP, on the AVR Raven, uses about 6 kB of RAM and 35 kB of ROM, thereby leaving about 10 kB of RAM and 93 kB of ROM for any other programs.

IV. Protocol Stack Overview

There are multiple protocol options that could possibly be adopted in resource constrained networks. While on the one hand brand new purpose-built protocols (such as CoAP [9]) could be used on top of the IPv6 infrastructure, on the other hand an existing protocol set could be implemented to provide interoperability with the existing infrastructure. One advantage of the second approach is that the IoT could be integrated into the existing Internet easily without modifications, allowing for all existing tools to remain functional and applicable to IoT devices.

This choice, of course, depends upon the requirements of a network make-up. For instance, in a deployment scenario where there are no (or very few) other devices, besides the IoT devices, it may make sense to use one of the newer protocols written from scratch for resource constrained networks. This provides the obvious advantage that these protocols have been optimized to suit this operating environment.

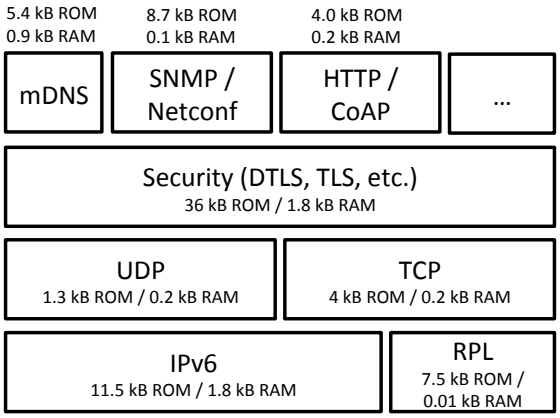

Figure 1. The protocol stack overview and the memory consumption of each component on the AVR Raven, when built using the Contiki OS.

On the other hand, there are also deployment scenarios that have a mixture of different types of network devices, many of which may already utilize the IP network infrastructure. In such a scenario, it would make far more sense to deploy existing network protocols to meet network management and monitoring needs. Furthermore, since IPv6 is likely to be used in both scenarios, the network protocol stack of a generic IoT platform is likely to look similar to that shown in Figure 1. It should be noted here that typically NETCONF is used along with SSH as the security mechanism, however, since the protocol itself can be used with other security mechanisms, we chose to use TLS. The depicted ROM and RAM usage values are obtained through experimentation and implementation experience, but the values for UDP, TCP and IPv6 are obtained from previous publications [10].

V. SNMP on Resource Constrained Devices

Our Contiki SNMP implementation supports the SNMPv1 and SNMPv3 message processing models and the `Get`, `GetNext` and `Set` operations. The `GetBulk` operation is supported for SNMPv3. The USM security model has been implemented and supports the HMAC-MD5-96 authentication and CFB128-AES-128 symmetric encryption protocols.

In order to optimize SNMP to fit within the constraints of the device, the modular architecture defined for SNMP [2] is not exactly followed. Simplifications are made by modifying the abstract service interfaces to minimize parameter passing overhead. For testing purposes, the `SNMPv2-MIB`, `IF-MIB`, `ENTITY-MIB` and `ENTITY-SENSOR-MIB` modules have been implemented for the AVR Raven.

Our Contiki SNMP implementation [11] has been tested for interoperability against applications from the Net-SNMP suite and the `scli` SNMP command line tool. In order to obtain the memory usage of just the SNMP agent, 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 enabled. The full SNMP implementation uses around 24% of the available ROM on the AVR Raven, and just over 1% of statically allocated RAM. In case only SNMPv1 is

Table II
EXPERIMENTAL RESULTS FOR THE MAXIMUM STACK SIZE (IN BYTES) INCLUDING PERCENTAGES OF USED RAM ON AN AVR RAVEN.

Version	Security Level	Max Stack Size
v1	-	688 (4%)
v3	noAuthNoPriv	708 (4%)
v3	authNoPriv	1140 (7%)
v3	authPriv	1144 (7%)

enabled, the agent uses about 7% of available ROM and under 1% of statically allocated RAM.

The memory usages 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 significantly increases code size. Using functions instead could reduce the code size, but would affect runtime performance. It is worth mentioning that the cryptographic primitives were ported from the OpenSSL library and are not optimized for embedded devices. The USM security model occupies almost half of the statically allocated RAM. This RAM is used to store localized keys and OIDs of the error indication counters.

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 finished, the stack is examined to see how much of it is overwritten. Table II presents the maximum stack size observed during experiments for different versions of SNMP and security levels 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 device.

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 processing the SNMP request is small relative to the time spent in data transfer. The increased header size of SNMPv3 in the `noAuthNoPriv` mode compared to SNMPv1 is clearly visible as additional delay on the low data-rate radio link. For both protocol versions with only trivial security, the processing delay constitutes 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. This is consistent with studies performed on more powerful devices [12].

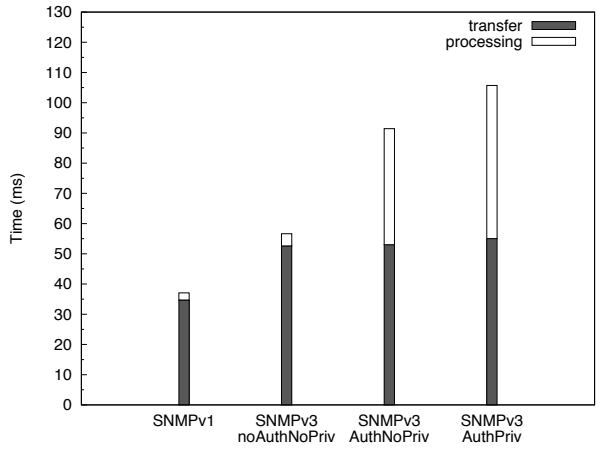

Figure 2. Average time taken for transferring and processing an SNMP request with one variable binding. (variance in the order of 3.5)

VI. NETCONF ON RESOURCE CONSTRAINED DEVICES

NETCONF [3] is a network management protocol that provides methods to install, manipulate and delete the configuration of network devices. It uses an XML based encoding for the protocol messages and also the configuration data. In order to fit NETCONF on resource constrained devices, only the `get`, `get-config`, `copy-config`, `lock` and `unlock` protocol operations were implemented. On resource constrained devices, it is unlikely that sub-tree filtering would be used, owing to the relative simplicity of system configurations. As such, this feature has been left out of the NETCONF implementation designed for resource constrained devices. The same reasoning applies to the `edit-config` operation.

Our NETCONF implementation was tested successfully with the `ncclient` Python library to ensure interoperability with existing tools. The resource consumption of our NETCONF implementation was measured using the same methods used for testing SNMP. All experiments were done using a configuration file comprising of three leaves containing data and XML wrapping. It was discovered that our NETCONF implementation, excluding security, uses about 25% of the available ROM on the AVR Raven, and approximately 8% of the statically allocated RAM. However, it should be noted that these memory usage metrics include the memory usage contributed by the Coffee File System (CFS) provided by Contiki, which is used to store the configuration and for processing NETCONF messages in order to be able to process large messages, avoiding RAM overflow. CFS occupies nearly 7% of ROM and only 0.5% of static RAM. Furthermore, a software layer for keeping track of the configuration and making necessary updates within Contiki was written, and this contributes 2% in ROM and 3% of static RAM.

Since both, CFS and the configuration layer, are general purpose, it can be deduced that effectively our NETCONF implementation uses about 17% of the available ROM, and 4% of static RAM. It is also worth noting that the core of NETCONF is XML message parsing and generation, which occupies 6860 bytes in ROM and 170 bytes in static RAM.

The maximum stack usage of the NETCONF implemen-

Table III
AVERAGE TIME FOR CERTAIN NETCONF OPERATIONS (EXCLUDING TIME SPENT ON DATA TRANSFERS, VARIANCE CLOSE TO 0).

Operation	Time Taken
<get-config>	0.592s
<copy-config>	0.752s
<close>	0.384s

tation was studied using a similar method to that used for obtaining the SNMP agent measurements. Over multiple experiments it was observed that the maximum stack usage by the NETCONF implementation was approximately 4% of the RAM, which is comparable to the SNMP v1 stack usage. However, since security is not a part of our NETCONF implementation by default, a direct comparison with SNMP v3 is not possible without using TLS/DTLS.

The time taken for processing NETCONF requests is shown in Table III. It can be observed that the time taken to process the requests is considerably higher than that for SNMP. The overhead of having to parse XML encoded messages, combined with having to access flash memory for reading/writing large messages increases the processing time. Our measurements show that one read from flash (open file, read, close file) is very fast; 100 bytes are read in less than 8ms, while 200 short reads take around 0.04s. Writing 100 bytes to flash (opening file, writing, closing file) takes about 0.016s. However, generating XML encoded messages and writing the result to the file takes most of the time, since each XML tag requires at least 3 writes but this can be as many as 10 writes or even more depending on the number of attributes. Specifically, in our tests, parsing of a `get-config` message and generating a response took 0.456s since the entire configuration needs to be written tag by tag to the `output.xml` swap file in flash. The `copy-config` operation takes 0.68s since the new configuration has to be written to the `config.xml` file, used for storing configuration parameters, and an `ok` response message has to be written to `output.xml`. Similarly, the XML processing of the `close` operation takes 0.32s. Overall, XML processing takes between 80%-90% of the total processing time for major operations in our NETCONF implementation. This time could be considerably reduced by using RAM or an XML library that is optimized to work with fewer write operations.

NETCONF has support for running over TLS, thereby making a pre-shared key TLS implementation a good choice for security on constrained devices. Since DTLS is very similar to TLS, we decided to create an implementation of TLS/DTLS for Contiki, which runs over the 6LoWPAN IPv6 adaptation layer.

Due to bandwidth and processing power limitations, the normal RSA or Diffie-Hellman key exchange mechanisms are not a valid option. While 4KB RSA implementations have proven to be possible at Oracle, the number of messages and computing cycles needed to establish the key context is still too large. As such, a pre-shared key method is often used. We use AES 128 encryption in the 8-bit Counter and CBC-MAC Mode (CCM) because AES-CCM is known to be suitable for constrained environments [8]. This cipher suite was also chosen by the IETF CoRE Working Group [9] as mandatory.

Table IV
MEMORY USAGE, IN BYTES, OF THE COMPONENTS OF THE TLS/DTLS IMPLEMENTATIONS INCLUDING PERCENTAGES ON AN AVR RAVEN. (COMPONENTS MARKED WITH ⋆ ARE USED BY TLS AND ◇ ARE USED BY DTLS).

Component	RAM	ROM
Contiki `mmem`$^{\star,\diamond}$	516 (3%)	238 (0.2%)
Contiki CFS*	92 (0.5%)	7502 (6%)
AES-CCM$^{\star,\diamond}$	310 (2%)	14058 (11%)
HMAC-SHA256$^{\star,\diamond}$	288 (2%)	3594 (3%)
TLS*	655 (4%)	12048 (9%)
DTLS$^{\diamond}$	847 (5%)	19342 (15%)
TLS Total	1861 (11%)	37440 (29%)
DTLS Total	1961 (12%)	37232 (28%)

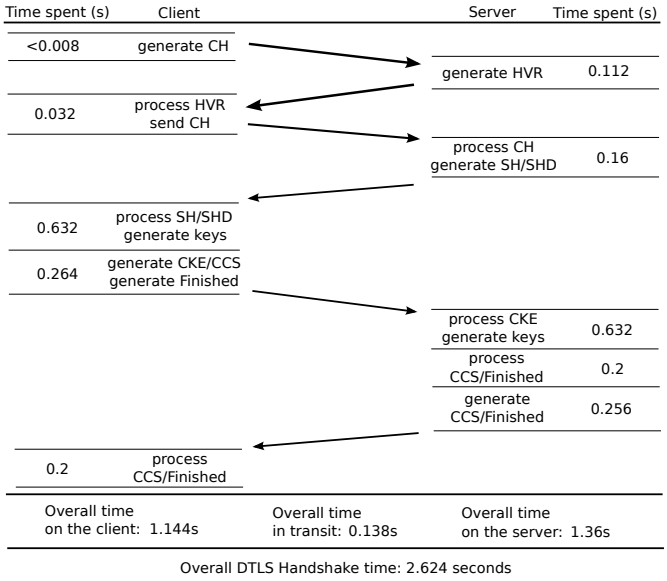

Figure 3. Breakdown of the time required for a DTLS handshake. (CH - Client Hello; HVR - Hello Verify Request; SH - Server Hello; SHD - Server Hello Done; CKE - Client Key Exchange; CCS - Change Cipher Spec)

While not being fully compliant with the TLS and DTLS specifications, since some mandatory cipher suites are too heavy for constrained devices, our implementations follow the specifications very closely. Our TLS/DTLS implementation do not support session resumption, since that would require additional usage of internal storage.

The memory consumption metrics for our TLS and DTLS implementation with PSK-AES-128-CCM-8 can be seen in Table IV. These Contiki components used by TLS/DTLS account for about 30% of RAM usage. As was the case with SNMP, it can once again be deduced that the cryptographic functions contribute significantly to resource usage. As such, if hardware supported encryption can be used, the resource usage of TLS/DTLS can be greatly reduced, to approximately 23kB, which is almost a 37% improvement. The TLS implementation's stack usage is a maximum of 11% of total RAM and 15% for DTLS.

A detailed breakdown of the time consumed by different steps in the DTLS handshake can be seen in Figure 3. Most of the time is spent on generating keys and creating the Finished messages. These computations take long because of the many

HMAC-SHA256 calculations involved. Hashing also causes the HVR to take time, since the cookie, which is sent to the client, is a hash of several fields from the CH message.

The TLS handshake is not shown here because it is similar to that of DTLS and takes a similar amount of time to complete. The differences are that before the handshake starts a TCP connection has to be established, which takes 0.048s; the server does not send the HVR, thereby saving about 0.1s on the server side and 0.032s on the client side; and the handshake has less messages sent, which decreases the time in transit. The TLS handshake takes an overall time of 2.496s.

Table V
MEMORY USAGE, IN BYTES, OF MANAGEMENT AND SECURITY PROTOCOLS INCLUDING PERCENTAGES ON AN AVR RAVEN.

Component	RAM	ROM	Stack
SNMPv1+SNMPv3/USM	235 (1%)	31220 (0.2%)	1144 (7%)
SNMPv1	43 (0.2%)	8860 (6%)	688 (4%)
NETCONF	627 (4%)	22768 (11%)	678 (4%)
TLS Total	1861 (11%)	37440 (29%)	1834 (11%)
DTLS Total	1961 (12%)	37232 (28%)	2454 (15%)

VII. CONCLUSIONS

We investigated the question whether the management of constrained networks and devices in the IoT can be accomplished by adopting existing network management protocols. By implementing light versions of SNMP and NETCONF and their security mechanisms, using the Contiki operating system, we were able to obtain detailed information about their resource requirements. We hope that the datapoints we provide will be useful as a baseline for future comparisons with alternative approaches to manage constrained networks and devices on the IoT.

The key challenges we identified are related to message framing issues (message reassembly at multiple layers is to be avoided), session establishment and maintenance issues (preventing efficient mechanisms to apply configuration changes), and security issues (session key generation is very costly, even with pre-shared keys in the case of TLS and DTLS).

Our implementation experience shows that SNMP makes efficient use of resources on constrained devices. Simple SNMP requests are processed on our device within 40-120 ms, depending on which version of SNMP and which security level is used. Compared to that, our NETCONF implementation requires 500-900 ms for processing simple requests, not considering any security costs. While this is a large number in comparison to our SNMP implementation, it can likely be further reduced by optimizing our NETCONF implementation (e.g., avoiding going through flash memory for relatively short messages that may still fit into memory). Our TLS implementation shows that adding security to NETCONF adds significant overhead (in the order of seconds) to the session start-up time. The USM security mechanism of SNMP does not create session keys and instead uses the pre-shared key directly for securing the communication, relying on key changes being done regularly using SNMP operations on the USM MIB module. The overhead introduced by a security layer could be improved significantly by using hardware encryption support, however, currently most of the resource constrained devices lack support for the cipher suite necessary or they lack an interface to access the cryptographic hardware embedded in radio transceivers.

With TLS/DTLS, the generation of a master secret and session keys has a big run-time overhead during session establishment, even if pre-shared keys are used. On the other hand, the authors of [13] showed that elliptic curve public key cryptography is feasible on constrained devices. They report that a full TLS handshake can be completed in less than 4 seconds on the Mica2. Since the AVR Raven and Mica2 are both using versions of the ATMega128 running at 8 MHz, the results obtained from our tests of using pre-shared keys are easily comparable to the elliptic curve public key cryptography results. Even though our implementation could likely be further optimized, the difference of just over 1 second for a full TLS handshake with elliptic curve cryptography is considerably smaller than we originally expected.

The NETCONF protocol assumes that devices can process relatively large XML encoded messages. A recent proposal of a programmatic RESTful interface for accessing data defined in YANG using the data-stores defined in NETCONF [14] might be a more resource friendly alternative to NETCONF for constrained devices since it can take advantage of CoAP's compact encoding of RESTful APIs [9] to produce smaller messages (avoiding TCP session overhead altogether, reducing much of the HTTP verbosity, and supporting JSON data encoding instead of XML data encoding). Since CoAP suggests DTLS to provide transport-layer security, the impact of the security mechanism remains likely unchanged, except that the DTLS security association is not bound anymore to a TCP session. Of course, implementation experience is needed to verify this.

We observed that protocols using small application layer messages that fit into one IPv6 packet are much better suited for memory constrained devices. For instance, the resources used to reassemble records in the TLS/DTLS security layer and NETCONF messages in the application layer are significant. It is also interesting to note that management protocols without explicit session state allow configuration changes to be applied by re-initializing (parts of) a device between protocol transactions, thereby avoiding the need for complex code to adapt to configuration changes dynamically. With session based transports and security mechanisms, such re-initializations have a big impact on the performance of the device since new sessions need to be established and thus they force additional complexity into the operating system in order to avoid re-initializations.

REFERENCES

[1] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," *IETF RFC 4944*, September 2007.

[2] D. Harrington, R. Presuhn, and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol," *RFC 3411*, December 2002.

[3] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "NETCONF Configuration Protocol," *IETF RFC 6241*, December 2006.

[4] J. Schönwälder, M. Björklund, and P. Shafer, "Network Configuration Management Using NETCONF and YANG," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 166–173, September 2010.
[5] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, ser. LCN '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462.
[6] D. Culler, J.Hill, P. Buonadonna, R. Szewczyk, and A. Woo. "A Network-Centric Approach to Embedded Software for Tiny Devices," in *Proceedings of the International Workshop on Embedded Systems*, ser. EMSOFT '01. Tahoe City, CA, USA: October 2001.
[7] D. Gräff. "Ethersex - the universal AVR firmware", *'http://www.ethersex.de/index.php/Main_Page'*.
[8] D. McGrew, and D. Bailey, "AES-CCM Cipher Suites for TLS ," *draft-mcgrew-tls-aes-ccm-03*, February 2012.
[9] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained Application Protocol (CoAP)," *draft-ietf-core-coap-11*, July 2012.
[10] 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 *Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008)*, Raleigh, North Carolina, USA, November 2008.
[11] S. Kuryla and J. Schönwälder, "Evaluation of the Resource Requirements of SNMP Agents on Constrained Devices," in *5th Conference on Autonomous Infrastructure, Management and Security (AIMS 2011), Springer LNCS 6734*, June 2011.
[12] J. Schönwälder and V. Marinov, "On the impact of security protocols on the performance of SNMP," *Network and Service Management, IEEE Transactions on*, vol. 8, no. 1, pp. 52 –64, March 2011.
[13] V. Gupta and M. Millard and S. Fung and Y. Zhu and N. Gura and H. Eberle and S.C. Shantz, "Sizzle: A Standards-based end-to-end Security Architecture for the Embedded Internet," in *Proc. of the 3rd IEEE Conf. on Pervasive Computing and Communications (PerCom 2005)*, March 2005.
[14] A. Bierman and M. Bjorklund. YANG-API Protocol. *draft-bierman-netconf-yang-api-00*, May 2012.

Anuj Sehgal (s.anuj@jacobs-university.de) is a PhD student of computer science at the School of Engineering and Science at Jacobs University Bremen, Germany, from where he also received an M.Sc. in computer science. His research is currently focused on the topic of underwater acoustic communications, but his interests include wireless sensor networks, disruption tolerant networks and embedded systems. After pursuing his undergraduate education at Brigham Young University-Hawaii, USA, he worked as a Systems Engineer at The I.T. Pros in San Diego, CA.

Vladislav Perelman (v.perelman@jacobs-university.de) received his B.Sc. and M.Sc. degrees in computer science from Jacobs University Bremen. During his time within the Computer Networks and Distributed Systems (CNDS) research group, his interests spanned network management and embedded systems networking. Upon completion of his graduate education he joined 360 Treasury System AG as a software developer.

Siarhei Kuryla (s.kuryla@jacobs-university.de) received an M.Sc in computer science from Jacobs University Bremen, where he was a member of the Computer Networks and Distributed Systems (CNDS) research group. He was granted a diploma in computer science by Yanka Kupala State University of Grodno, Belarus. After completing his graduate education he joined 360 Treasury Systems AG as a software developer.

Jürgen Schönwälder (j.schoenwaelder@jacobs-university.de) is leading the Computer Networks and Distributed Systems (CNDS) research group at Jacobs University Bremen. His main research interests are network management, distributed systems, wireless sensor networks, and network security. He is an active member of the Internet Engineering Task Force (IETF) where he has edited about 30 network management related specifications and standards. He has been co-chairing the ISMS working group of the IETF and he currently serves as co-chair of the NETMOD working group. Previously, he has been the chair of the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF). He currently serves on the editorial boards of the IEEE Transactions on Network and Service Management, the Journal of Network and Systems Management, and the International Journal of Network Management.