Enablement of CoAP Stack on Sparrow Wireless Sensor Network

Dan Drăgan, Dan Tudose, Dan Dragomir, *Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest*

***Abstract*—As smart embedded devices tend to become more and more ubiquitous, interconnecting these gadgets with the existing network infrastructure by using simple application layer protocols is highly needed. This article presents a new approach for a specialized web transfer protocol widely available in the Internet of Things used in conjunction with a set of newly designed wireless sensor nodes. The application layer protocol is using an IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) stack in order to facilitate the interconnection of a group of constrained devices with other networks.**

***Keywords*—CoAP, 6LoWPAN, 802.15.4, wireless sensor network, microcontroller, mesh network, Internet of Things.**

I. Introduction

The Internet of Things have become an increasingly growing topic of conversation in the past few years, both in the industry and academia. The greatest contribution to this area came from the use of low-power smart devices in conjunction with web interfaces, in order to interconnect gadgets with existent infrastructures and services. Constrained Application Protocol (CoAP) is a web transfer protocol that is intended for use with reduced-capability, low-power devices. This article describes an existing CoAP implementation over a 6LoWPAN stack and the effort invested in porting the existent environment on a wireless sensor node family, Sparrow v4. The current implementation run on network topologies where each wireless node have one of the following roles: coordinator, router, device. Sparrow nodes use AVR microcontrollers with small amount of flash memory. The greatest achievement of this project was reducing the dimension of the application's executable file in order to fit the flash memory, by reducing some of the capabilities and features the starting point implementation supported, like the dynamic memory allocator and the real-time operating system support. Another important contribution was porting the existent implementation for the coordinator node from an ARM device on AVR microcontroller family.

This paper is divided into six sections. The first two present a brief introduction into the protocols and mechanisms used by the solution and describes the current state of its implementation. The fourth section presents the new architecture of the network topology that supports CoAP and gives details regarding the changes and the process of porting the stack on Sparrow. The fifth section reveals the power consumption of these nodes, monitored while sending periodic messages between them. The last section highlights the conclusion and describes the future work that should be done in order to achieve low power consumption of CoAP stack on this new hardware architecture.

II. Background

A. Constrained Application Protocol

The ubiquity of web services in most applications determined the reimplementation of these services for the embedded world. These web application programming interfaces depend on the fundamental Representational State Transfer (REST) architecture of the Web. CoAP represents an effort for enabling the REST architecture in a suitable form for the most constrained nodes. These nodes often have 8-bit microcontrollers with small amounts of read-only memory and random-access memory. CoAP provides a request/response interaction model between the endpoints of an application, it provides built-in mechanisms of services and resources discovery, and includes key Web concepts such as Internet media types and Uniform Resource Identifier (URI).

CoAP was designed considering some important HTTP features such as extensible header options, resource abstraction, RESTful interfaces, and also the uses of a compact binary representation, thus making parsing simpler. One major difference between CoAP and HTTP is that the former is using UDP instead of TCP, thus allowing CoAP to be used for one-to-many and many-to-one communication patterns.

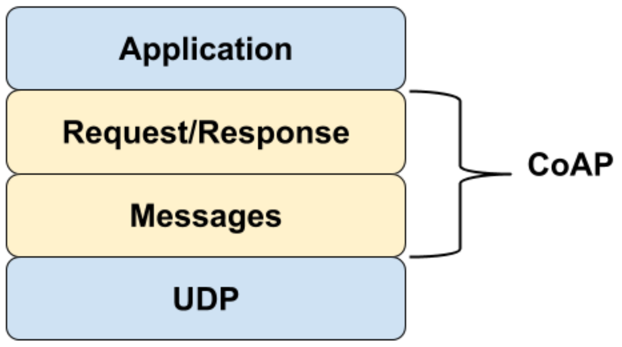

Fig. 1. Two layer structure of CoAP

As one may see in the figure above, CoAP contains a message layer responsible for UDP communication and reliability, and a second layer responsible for request or response interactions. CoAP also uses asynchronous message exchange between end points and the messages are marked as requests or responses using some embedded method codes and response codes. CoAP messages may be sent reliably or non-reliably as described in [1]. Reliable or conformable messages are retransmitted with exponential timeouts until the receiver acknowledges them or the maximum number of retransmissions is reached. CoAP was also designed to provide group communication via IP multicast.

B. IPv6 over Low-Power Wireless Personal Area Networks

6LoWPAN is a simple low cost communication protocol allowing wireless connectivity for devices with limited power running applications with relaxed throughput requirements. It provides IPv6 networking over IEEE 802.15.4 networks formed by devices that are compatible with this IEEE standard which is characterized by low bit rate, low power, short range and low cost. A reduced function device or lower processing capability sensor node in a 6LoWPAN may send data packet to an IP-enabled device outside the Personal Area Network network by first sending the packets to higher processing capability sensor nodes, also called full function device. These devices act as routers in the PAN and will forward data packets to the gateway as described in the image below. The gateway connects the PAN with the IPv6 domain and forwards the packet to the destination device by using the IP address.

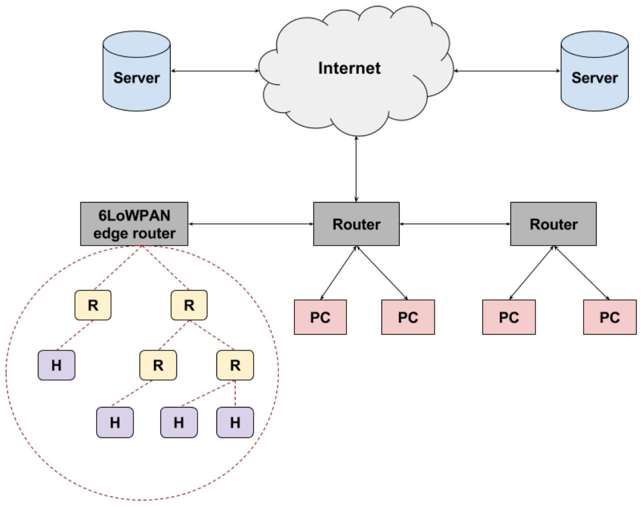

Fig. 2. IPv6 network connected with Wireless Personal Area Network

The reference model for 6LoWPAN protocol stack is very basic as presented in [2]. The IEEE 802.15.4 standard PHY and MAC layers are used as base layers while using IPv6 in the network layer. The physical layer is responsible for power optimization implementing specification for low-rate wireless personal area network, while the MAC layers is handling error correction that may occur in the physical layer during receiving and transmission.
The frame structures of MAC layer defined in IEEE 802.15.4 standard are: data frame, beacon frame, MAC command frame and acknowledgment frame. A data frame is used for data transfers while the beacon frames are used by the PAN coordinator to transmit periodic information. Sending proper responses in case of successful frame reception and managing all MAC peer entity control transfers is handled by the acknowledgment frame and the MAC command frame. All frames except acknowledgment frame have MAC Service Data Unit which is prefixed by MAC Header and appended by MAC Footer.
The maximum MAC frame size defined by IEEE 802.15.4 is only 127 octets with 25 bytes reserved for frame overhead and only 102 bytes left for payload, even if the minimum allowed maximum transmission unit (MTU) for an IPv6 packet over IEEE 802.15.4 is 1280 bytes. If the link-layer imposes further overhead for security purposes, the situation is getting worse by inserting an Auxiliary Security Header in the MAC frame, thus in the worst case scenario leaving only 81 bytes for IPv6 packets and a full IPv6 packet will not fit in an IEEE 802.15.4 frame. In addition to this, the IPv6 header in an IPv6 packet is 40 bytes, so there are only 41 bytes left for higher level layers. The 8 bytes User Datagram Protocol (UDP) header is preferred over Transmission Control Protocol (TCP) header for transport layer, although IPv6 packets contain only several bytes for application data. IETF 6LoWPAN working group proposed that an adaptation layer between MAC layer and the network layer to be added in order to achieve header compression, fragmentation and layer-two forwarding.

III. Related work

Sparrow v4 is a relatively new wireless sensor node architecture designed at Politehnica University of Bucharest. It consists of an 8-bit Atmel AVR microcontroller with low power 2.4GHz transceiver for IEEE 802.15.4 and a series of temperature, pressure and light sensors. Because of its recent introduction, there is no specific CoAP implementation for Sparrow architecture.
CoAP has working implementations for a couple of platforms like Contiki OS [3], Tiny OS [4] and it is also available in a series of libraries for Arduino. It has various applicability like building and home automation or real time condition based monitoring in smart grids as presented in [5].
CoAP implementation used as starting point for our project was using IPv6 over WPAN mesh networks. This type of network defined three types of devices with different roles: coordinator, routers and devices. The coordinator is the initiator of the Personal Area Network and it will act like a gateway to the IPv6 network, but it will also maintain routing tables and the evidence of the rest of the devices.
The initial implementation uses as coordinator node a development board with an Atmel ARM-based microcontroller and the Routers and Devices were development boards with Atmel AVR microcontrollers. The coordinator in this setup was controlled by a Real Time Operating System (FreeRTOS) with a scheduler designed to provide a predictable execution pattern. The scheduler was used to plan the execution of both initialization tasks for network, low-power IP and CoAP events like responding to device join/leave or other CoAP requests.

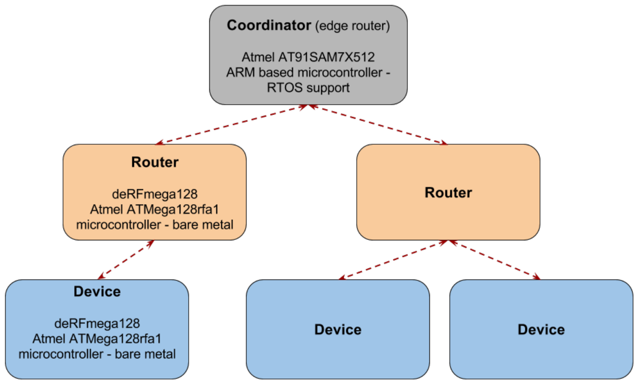

Fig. 3. WPAN mesh network with ARM-based coordinator and RTOS

The RTOS objects like tasks, queues, semaphores, software timers, mutexes and event groups can be created using either dynamically or statically allocated random-access memory. Its scheduler can be preemptive, cooperative or it may support hybrid configuration options, with optional time slicing.
The routers were used in order to send information from an end node to other end node or to the coordinator. These type of nodes maintain a table with all the nodes that joined the network and are attached to it. The routing process is transparent to CoAP devices, thus an end node might send a message to a specific node in the network and the message will be routed hop by hop.
The initial project provided demo applications for all the entities involved in a 6LoWPAN mesh network in order to prove the real utility of the APIs. A sketch that was of interest for our purpose was a client-server application between the coordinator of the Personal Area Network and the devices and routers. The purpose of this example is to transmit some periodic echo messages from the end nodes to the coordinator. These periodic transmissions could be activated using a GET message with a specific payload.

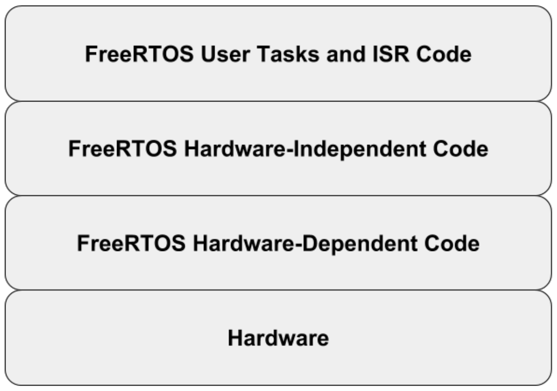
FreeRTOS User Tasks and ISR Code
FreeRTOS Hardware-Independent Code
FreeRTOS Hardware-Dependent Code
Hardware

Fig. 4. FreeRTOS software layers

The initialization methods called by the coordinator provide the device with some static information, like MAC and IPv6 addresses, Personal Area Network identification, child and coordinator ports. The coordinator is also in charge with the assignment of static addresses for the rest of the participants in the PAN and is listening on a predefined port for CoAP messages and acts as a gateway.
All basic operations supported by the coordinator are in correspondence with a callback. The most important of them is the CoAP user defined data callback, which is called when a CoAP message was received. Another important function is the CoAP user defined error callback, called when a CoAP message cannot be delivered after timeout or exceeding the maximum number of retransmissions.
The instruction flow for devices and routers is simpler. They are all the channels or just the default channel if provided, looking over a list with discovered devices and trying to find a coordinator. If the coordinator is found, then the nodes request joining the network. If any issue occurred during the coordinator discovery or join process, the nodes are sent into reset mode, and the operation is retried.

IV. Implementation

The new approach required using Sparrow v4 for all three types of nodes. Porting the Device and Router applications was not very difficult because the previous setup used the same microcontroller, so the only changes were for enabling sensors, LEDs and build configuration files.
On the Coordinator side the effort was more consistent. The figure below shows the new model with a redesigned network. First of all, the build configuration files needed to be rewritten, because of the switch from ARM to AVR, not only for the CoAP application, but also for other dependencies, like ROM, 6LoWPAN subroutines and CoAP subsystem. A series of static libraries containing CoAP and network handles specific for the coordinator were created for AVR arhitecture. The build files for generating these static libraries were rewritten for Sparrow architecture using the same source files that generated the static libraries for ARM. The build files for CoAP examples were adapted from the existent ones and they contain instructions for linking with the newly generated libraries.
Since the flash size was now, reduced to a quarter, the Real Time OS had to be removed. The execution pattern was predictable, containing mainly tasks for initialization and request/response events, so the RTOS scheduler was easily replaced with bare metal support for the initialization tasks and callbacks.
A major improvement added to the initial implementation was enabling the debugging capabilities on the coordinator by sending messages to the serial interface. This proved to be of great help in problem identification and testing purposes. It required changes in a series of build files, because the implementation uses different debugging functions each being associated with a specific layer.
Another essential change was reducing the .data and .bss sections from sizes that required 280% of microcontroller's flash memory capacity to 53%. The significant reduction of the executable file's dimension was fixed by discarding the

dynamic memory allocator, since we plan to use CoAP on wireless networks with relative small number of nodes, thus the overhead for this capability is not justified.

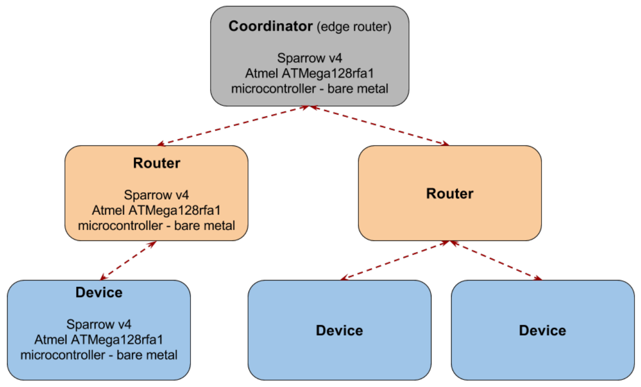

Fig. 5. WPAN mesh network with AVR-based coordinator and bare-metal support

6LoWPAN defines two categories of routing: mesh-under and route-over. Our study is using the mesh-under routing method and layer-two addresses, like IEEE 802.15.4 MAC and short address to forward data packets. In a mesh-under system, routing of data happens transparently, hence mesh-under networks are considered to be one IP subnet. The only IP router in such a system is the gateway router, also known as coordinator. A broadcast domain is established to ensure compatibility with higher layer IPv6 protocols such as duplicate address detection. These messages have to be sent to all devices in the network, resulting in high network load.

V. Results

After enabling the support for Sparrow v4 architecture on all types of nodes, we were able to set up a wireless network containing a coordinator and a couple of routers and devices in order to carry out some basic tests for CoAP.
The most important criteria for us was the power consumption of end devices, because of their power cycles, not being able to sleep even if no data are transmitted, thus this factor might be improved.
The Sparrow wireless nodes contain sensors like temperature and humidity sensor, light sensor, barometer. First of all we wanted to see how will the architecture behave when all the sensors are active.
The testing setup consisted of a coordinator device and an end node whom was attached a battery support. The end node was sending periodic messages to the coordinator, which replied with an acknowledgement. In order to perform current measurements we used a Keysight 34461A digital multimeter and connected it in series with our end node. In the below figure one may see that during the monitoring, the current measured was around 20.8 milliampere (mA).

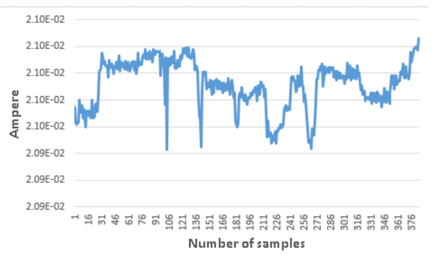

Fig. 6. Current measurement with all sensors on

After that, we considered that it will be relevant to see how much power will our end node consume without any sensor enabled in order to estimate and analyze their power consumption. The below figure shows as that the current measured was on average 17.8 milliampere with very small variation.

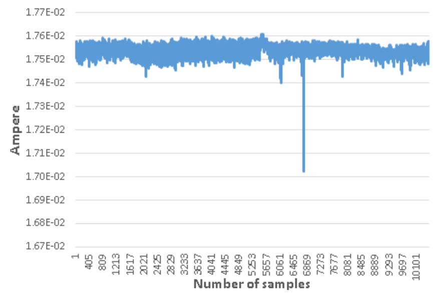

Fig. 7. Current measurement with all sensors off

As a next step we plan to enable periodic sleep modes and retry another series of current measurements. A massive improvement is expected on the average power consumed by the end host. The routers are responsible for transmitting messages from their child nodes to other entities in the WPAN, thus their sleep modes will have to be infrequent. However, it may be possible that the router devices consumption to be optimized if the number of children in their routing table is not very large.

VI. Conclusion and Future Work

The most important achievement this project brought is the enablement of a highly popular web transfer protocol stack on a new power efficient wireless device with reduced memory and computational capabilities.
We were able to erase much of the computational intensive features like the Real Time Operating System and reduce the flash memory usage, preserving the legacy capabilities. This was achieved by replacing the custom dynamic memory allocator.
The proposed solution is behaving in the same manner as the old one, but using four times less flash memory for the coordinator node and consuming fewer power.
The next objective will be implementing an efficient sleep mechanisms for all the members of the network except the

coordinator, which is required to be awake in order to route data sent between end nodes.
The main issue that we are concerned about is the clock drift, a phenomena where a clock does not run at exactly the same rate as a reference clock and after some time the clock gradually desynchronizes from the other clock. The lack of synchronization is related to the quality of the quartz oscillator and it will be cause great problems in handling the sleep times of each device in the network.
First of all, we will start with the end nodes and try to determine the optimal synchronization between their sleep times using various mechanisms. As a starting point, we will choose a method based on time guards, so when a node will came out of its sleep mode, it will wait for a while in order to give a change to the other neighbors to wake up. The next step will consist in more sophisticated mechanisms, feedback loops, like in the case of PID controllers or temperature compensation in order to reimburse the clock drift between various nodes.

REFERENCES

[1] Z. Shelby, K. Hartke, C. Bormann, The Constrained Application Protocol (CoAP), Request for Comments: 7252, 2014

[2] Jonas Olsson, 6LoWPAN demystified, Texas Instruments Incorporated, 2014

[3] Matthias Kovatsch, Simon Duquennoy, Adam Dunkels, A Low-Power CoAP for Contiki, 2011

[4] Alessandro Ludovici, Pol Moreno and Anna Calveras, TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS, 2013

[5] Manveer Joshi, Bikram Pal Kaur, CoAP Protocol for Constrained Networks, 2015

[6] Gee Keng Ee, Chee Kyun Ng, Nor Kamariah Noordin and Borhanuddin Mohd. Ali, A Review of 6LoWPAN Routing Protocols, 2010