Embedded Kernel customization to optimize performance and power management. An application to IoT

José Antonio Álvarez Bermejo[1], Angel Jesús Varela Vaca[2], Francisco G. Montoya[3], Juan Antonio López Ramos[4] y Consolación Gil Montoya[5]

Resumen— **Embedded systems are nowadays a corner stone in research for areas such as Internet of Things. An interesting aspect regarding these systems is the fact that they are tied to strong restrictions (computing power and power source). The work we present here is related on how to easily create kernel images tailored to the architecture of our system. Several conclusions are drawn on performance and power consumption.**

Palabras clave— **Kernel customization, performance, boot-up time, power consumption**

I. Introduction

THE Embedded System consists of microprocessor and memory embedded at the hardware, it has a limited missions. It is different from general-purpose computer, and it is widely used in mobile communications, intelligent robots, telematics and home appliances. In the Embedded System, operating system is divided into various types to the purpose, lets add an issue about the relationship between new or customized hardware and the need to tailor a kernel for it. In particular, the Linux System is based on open-source and provides stability. Hence it is widely applied to the Embedded Sytem. Many information appliance companies participated in establishing CELF (Consumer Electronics Linux Forum) and announced CELF Specification 1.0 [1]. And it is an effort to apply Linux to home appliances products. The specification is focused on such issues as boot-up time, power-management, real-time and kernel size. Packing the root filesystem is crucial in order to make faster boot-up time in the Embedded Linux [2]. Therefore, having the choice to select layers of funcionality, removing funcionality by layers instead of manually, selecting versions of libraries, compile them optimized for the processor-chipset couple and choosing your root filesystem seems to be mandatory.

Canonical, the lead commercial brand behind the open-source Ubuntu Linux operating system, is getting into the embedded device market in a bid to secure the Internet of Things (IoT, in advance). This unveils how IoT boosts the research for embedded systems. At this point, a key issue to focus is which base software to upload into the board. Ubuntu, the flaghsip of Canonical is understood as an operating system for servers, cloud and desktops. Why we refer to Canonical? Now Canonical is positioning Ubuntu to be relevant for embedded devices and IoT. Strategical moves in its kernels will make the whole board of researchers take clues on what to focus on. It is worth to note that they are working on Snappy, an Ubuntu Core technology.

Snappy can be defined as a minimal version of Ubuntu, it is not strange that one spend time unloading conventional kernels to later install them into an embedded system. This is an error-prone practice, and also performance-aware as aspects as the file systems, compilers used, etc. could harm performance in the embedded core. On the other hand, the compiled kernels were not usually fitted to the architecture of the board where the embedded core was installed into. These aspects are hot spots where Canonical is working with the optimized Snappy system that can improve security and application updates. This is what we want to cover in this paper: provide an efficient environment (kernel) with an appropriate set of libraries (cryptographic) to secure user data and communications. In order to achieve that, it is important to (apart from the built-in security) provide a fast mechanism where one can propagate patches to all the nodes of the IoT network (for example).

Taking into account the steps taken from Canonical, Snappy still depends on a third party. The rest of the research vectors are in the right direction: try to provide an efficient light-weight kernel, which is not actually customizable at all. We identified some time ago that for mobile devices we really had to raise the boundaries on the reliability, efficiency and security of the update mechanisms, as well as the isolation of apps from one another. We also consider that taking control on the update mechanism is crucial. This is, actually, one of the keys we pursue when seeking for a kernel customization: being able to control the propagation of patches in our own embedded systems. Snappy updates can be delivered as smaller, more efficient transactional updates. Heartbleed and Shellshock are two open-source technology vulnerabilities have been disclosed in 2014. Both is-

[1]Dpto. de Informática, Universidad de Almería, e-mail: jaberme@ual.es.
[2]Dpto. de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, e-mail: ajvarela@us.es.
[3]Dpto. de Ingeniería, Universidad de Almería, e-mail: pagilm@ual.es
[4]Dpto. de Matemáticas, Universidad de Almería, e-mail: jlopez@ual.es.
[5]Dpto. de Informática, Universidad de Almería, e-mail: cgilm@ual.es.

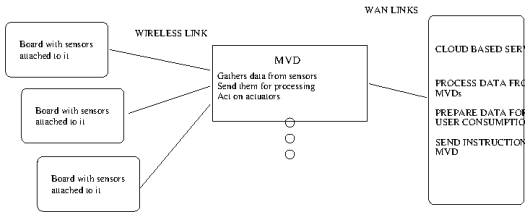

Fig. 1. Sensor network

sues left embedded devices running Linux exposed to risk, as the impacted vendors had to roll out patches for users. With IoT, anything and everything can be connected to the Internet, even potentially a lawn-mower, and it is usually up to the vendor to provide patches for any security issues. This is a worrying aspect that need also to be considered.

II. Internet of things

To help capitalize on the IoT opportunity, many embedded based companies are putting effort into the IoT, many of then by creating their own division within the company. The embedded Linux market is not new, and is one in which Cavium's MontaVista and Intel's Wind River have led with their embedded Linux technologies. If we go around oneself house and look at all the devices that are still vulnerable to Heartbleed or Shellshock, it's embarrassing. The problem is solved by properly customizing kernels that operate them. Efficiency and power consumption can be, therefore, partly achieved by means of a customized and light-weight kernel.

A. Sensor networks

Sensor networks are composed of computational devices provided with light computing resources, also their batteries are an issue. Sensor networks are starting to become a common factor in modern society. They are the basis for the newly defined way of interconnecting light processing nodes that scatter sensor data (from sensor attached to them) to a central node. This central node (also known as Minimal Viable Device or MVD from now) is the responsible of gathering information from these devices forming a small network, as depicted in Figure 1. In addition, IoT is a central part of popular concepts such as smart homes, smart transportation, and traffic control. Sensor networks and IoT as its extension, is therefore a core part of a concept that the society is adopting in its day by day life. A big effort is pushed into developing protocols to provide support to IoT but little care is put in the assurance of the data in the network is sent from an authorized and trusted source and how efficient these protocols are run on the MVDs.

Commonly, see Figure 1 the data produced by sensors is sent to a central node (in the same network) where some sort of processing is carried out. This processing prepares the data to be sent to a cloud where it is processed in order to get metainformation that is provided to consumers (smartphones) or actuators. Unfortunately, the whole process can be easily

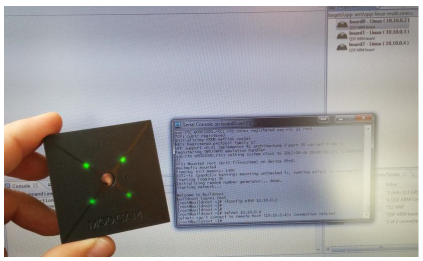

Fig. 2. Minimum Viable Device for IoT

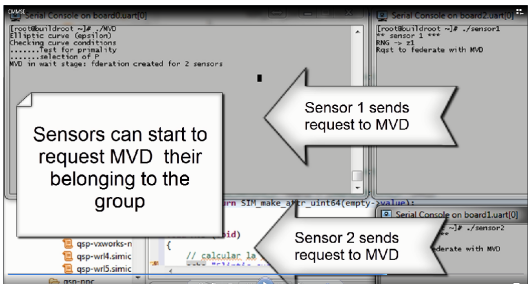

Fig. 3. Requesting federation

compromised whether the data is received from fake sensors or it was corrupted. Figure 2 shows the device that is commonly used to gather data from sensors. This device is an ARM based computing node with an Android OS running on it, which caused it not to perform efficiently (with no screen and dialer modules). This node is connected to a cloud service where data is distributed after being processed, to customers.

Surprisingly, a vast effort in securing the processing parts at the cloud has been put during last five years. Nevertheless, the sensor network side has been left aside.

The kernel customization presented in this paper establishes a mean to create an efficient environment to federate sensors. Figure 3 shows three computing ARM devices emulated: the MVD, and two other boards where sensors were produced as software threads on the board.

We provided three sensors. Sensor 1 and Sensor 2 are allowed to communicate. When the board boots Sensor 1 asks MVD (Minimal Viable Device that acts as a hub gathering data from the Sensors) to be federated, Sensor 2 does the same. Sensor 3 is treated as an illegal sensor.

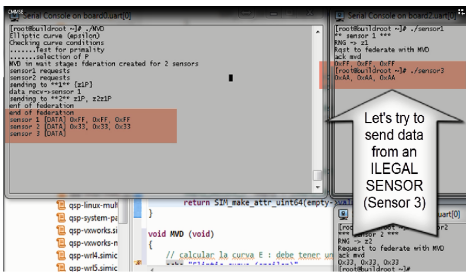

Fig. 4. Sending data

B. Group key agreement on elliptic curves

Since Diffie and Hellman proposed in [7] the first protocol that provides a key exchange through an insecure channel, many authors have introduced schemes of this type, as [12], [13], or [16]. Many of them base their fortress, as it is in Diffie and Hellman's original work, on the difficulty to solve some problems on classical Number Theory. However, works as [14] or [9] where some vulnerabilities of [7] and [12] respectively are shown, as well as the constant advances in computing and capabilities of modern computers has lead to consiider different structures to develop such key exchange protocols. Nowadays, the recommended standard is based in the so-called Elliptic Curve Cryptography. based on the set of points of an elliptic curve, [8] and [10].

Modern applications of communication systems allow communications among group of users instead of restricting this to a pair of communicating parties. In this sense, Steiner et al. extend in [15] the Diffie-Hellman protocol over a finite field to a group of users. They introduce a suite of three different protocols that generalize that of [7]. Recently, in [11], the author uses one of the protocols to show its scalability using the arithmetic in an elliptic curve. However, although some good results on several tests are presented, no proof on the correctness of the protocol is given in this work, as well as a formal study of its security. Elliptic Curve Cryptography is applicable for group key management and recommendable for networks formed by light devices as a sensor network, where usually, no security treatment of the transmitted information is nowadays considered, although in some situations this information could be of sensible nature.

The sensor secured protocol [17], was coded in C (arm-gcc, neither thumb nor thumb2 was used) and linked with micro-ecc (http://kmackay.ca/micro-ecc/) libraries. Figure 4 shows how the legal sensors send data to the MVD and the data is accepted whereas data from Sensor 3 is not considered as it is not part of the federated network. Such protocols need to run in efficient boards, with customized kernels. Able to be updated for security patches as soon as a fix is available. This paper is the basis to create such an environment where security (or any other networked protocol) can run safe.

C. Frameworks available for embedded systems

The field where Canonical is working is really interesting, the need to customize kernels is real. Although there are options much more reliable than Snappy that is still too tied to a company, see Fig.5. Another effort in the embedded Linux space is the Yocto Project [3], [4] which is a Linux Foundation Collaboration project. This project is largely leaded by Intel and there is no relationship between Snappy Ubuntu Core and Yocto. Yocto is aimed at the world where people roll their own operating system. It provides tools that let us create an entire operating system from scratch. Yocto is a way for a device vendor to build a single operating system image for a device, while Snappy Ubuntu Core enables a vendor to compose a device's operating system out of a set of images from different parties.

The Linux Foundation's collaborative Yocto Project is not about a single board, it is about creating a custom embedded Linux package for a board of your choice (and of any architecture). It is a open source embedded Linux build system, package metadata and SDK generator. Therefore optimized builds of the libraries to use can be uploaded, leading to gain in performance as the code is deployed together with the image and after the observation all the details of the hardware.

Yocto uses the Poky build system at its core. In Poky's default configuration an initial image footprint, can be provided, that ranges from a shell-accessible minimal image all the way up to a Linux Standard Base-compliant image. From these base image types, metadata layers can be added to extend functionalities; layers can provide an additional software stack for an image type, add a board support package (BSP) for additional hardware, even represent a new image type. The final output of the Poky build system is usually an image that can be flashed on your device. At the time of writing the Yocto Project has BSPs for several boards like YP Core's Daisy, Dora and Dylan (and as exposed in Fig. 5 it gets updated with a high frequency).

The point where it becomes interesting is the relationship with Intel in the project. ARM is beyond doubt the current emperor in embedded hardware and Intel has to go a long way to beat them. ARM has solidified its presence mostly by powering Android devices, which is open source. Why not Intel then? Yocto Linux product showcase includes Intel's meta-intel BSP layer.

D. Yocto

Yocto Project is a complex-to-use tool. At first, it requires a lot of work, furthermore, because of the nature of the open source, troubles may occur with legal issues, lack of enterprise support, documentation quality, complexity of the software and coding, and the compatibility between open source and commercial components. The problem is solved by means of HOB which has a graphical tool, here the licence of the included components are controlled easily.

Yocto Project includes Board Support Packages (BSPs), compilers, layers, and additional tools. Although the operating system, embedded Linux, is built automatically, the actual coding of the applications and drivers related to the specific hardware board is not directly related to the Yocto Project. The concept of recipe to describe configurable things. A recipe is a simple text document that contains a set of instructions to build binary packages and images. In other words, they are configuration files with predefined formats and semantics. Recipes can be created for a certain purpose, for example prepare

Feature/ framework	Supported CPU architectures	Founded	Open source/ commercial	Docs	Support services	Updates	Developer communities	Partner organizations
LTIB	ARM, PPC, Coldfire	2005	Open	+	Freescale	Not actively	++	Freescale
Buildroot	ARM, PPC, MIPS, x86	1999	Open	++	No	Stable releases every 6 months	+++	-
OpenEmbed-ded	ARM, PPC, MIPS, x86, x86-64, amd64, avr64	2004	Open	+++	Independent consultants	Regularly	+++	-
Yocto Project	ARM, PPC, MIPS, x86, x86-64	2010	Open	+++	Independent consultants	Stable releases every 6 months	+++	Intel, Huawei, Texas Instruments, Wind River, MontaVista
TimeSys LinuxLink PRO	ARM, PPC, MIPS, INTEL, Atmel	1995	Comm.	+++	Yes	Regularly	+	-
MontaVista Linux 6	ARM, PPC, MIPS, x86, x86-64, amd64	1999	Comm.	+++	Yes	Regularly	+	-
Wind River Linux 5	ARM, PPC, MIPS, Intel	1981	Comm.	+++	Yes	Monthly	+	Freescale, ARM, Intel, Texas Instruments, MIPS

Fig. 5. Available frameworks (see [3])

the needed software for the embedded system to run ECDH security protocols. For example, a sample recipe [5], depicted in Fig. 6, could include:

```
DESCRIPTION = "Simple helloworld application"
SECTION = "examples"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;m
cf8506ecda2f7b4f302"

SRC_URI = "file://helloworld.c"

S = "${WORKDIR}"

do_compile() {
        ${CC} helloworld.c -o helloworld
}

do_install() {
        install -d ${D}${bindir}
        install -m 0755 helloworld ${D}${bindir}
}
```

Fig. 6. Sample recipe

There are four different types of recipes according to the purpose:

- application
- image
- image baseline
- image release

Latter recipe is determined in this work to remove unuseful features and add libraries for cryptography. One image recipe defines configurations, applications, Linux kernel versions, and boot loader versions for one specific product. In addition, creating a new image recipe for the each release of the software product, reproducibility and traceability of the software product is enabled.

The work involved in creating the layer for our test system is as following:

- General configuration : states general aspects.
- Machine configuration : related to the details of the processor.
- Linux kernel recipes : related to which versions and aspects of the kernel to include, enhance, avoid.
- Boot loader recipes : rootfs (as said in section 1, boot-up time is improved).
- Recipes related to the HW platform.
- Application recipes : related to which libraries are needed (i.e. micro-ecc libs for ECDH and ECSA wich tcc compilation support).
- Image recipes.
- Scripts and templates.

Apart from this Yocto images are rarely over 2GB meanwhile others like Linaros' or Snappy's or Ubuntu's are over 3GB.

III. Architecture and tests

The tests were run using an emulated ARM based architecture. The specs of the emulated ARM with the following baseline configuration :

- Frequency 500MHz, McPAT is set to use 65nm technology.
- L1 data and instruction cache: 8kB, latency 5ns, 2-way associative.
- L2 unify cache: 128kB, latency 12ns, 1-way associative

Only one core was used, then a simulation with four cores was run. The testbed was run on a VirtualBox virtual machine that runs Ubuntu 12.04. For the emulation of the processor, we used the Gem5 simulator [6]

A. Gem5 Simulator

Gem5 is a computer system simulation platform. Unlike processor architecture simulator such as SimpleScalar, Gem5 can perform a complete multi-core platform. For processors, Gem5 is capable of simulating a number of ISAs, including Alpha, ARM, MIPS and X86. There are two modes for Gem5 to simulate a system:

- Syscall Emulation (SE): in this mode there is no operation system (OS). All the system calls in the application is emulated by Gem5.

- Full System (FS): in this mode a complete system is simulated, including the OS and all peripherals.

In Gem5 SE mode, the support for multi-threaded application is limited. In this paper, the FS mode is used and ARM ISA is chosen as the target architecture.

B. McPAT : power consumption analysis

McPAT is a framework for high-level area, timing and power modeling developed by HP labs. Basically, McPAT reads in (micro-)architectural parameters and event statistics, and estimating the area, timing and power figures for each component of the system. It allows to model different technology nodes from 90nm to 22nm. The accuracy depends on the level of details provided by the input. For this paper we used 65nm technology.

IV. Tests and results

For testing the kernel, we run our Yocto image against the ARM runing an Ubuntu image. The code run is a code to calculate elliptic curves for key agreement protocols [17] (very popular in sensor and P2P networks). We used the micro-ecc libraries. During the test, network connection has been skipped thereby the processor was calculating curves with no key exchange. Following table I shows that code :

TABLA I
Time and power wasted running the protocol.

kernel image	time (secs)	peak power (W)
Ubuntu *	3.871756	0.732842
Customized	3.028246	0.706757

* The ubuntu image was created with RootStock (as described in *http://www.m5sim.org/Ubuntu_Disk_Image_for_ARM_Full_System*)

The simulation technology used was 65 nm. Using Long Channel Devices when Appropriate. Interconnect metal projection= conservative interconnect technology projection. Core clock Rate(MHz) 500

- Processor running Ubuntu image
  - Peak Power = 0.732842 W
  - Total Leakage = 0.0295131 W
  - Peak Dynamic = 0.703329 W
  - Subthreshold Leakage = 0.0134206 W
  - Gate Leakage = 0.0160925 W
- Processor running customized image
  - Peak Power = 0.706757 W
  - Total Leakage = 0.0245219 W
  - Peak Dynamic = 0.682236 W
  - Subthreshold Leakage = 0.0122036 W
  - Gate Leakage = 0.0140292 W

V. Conclusions

Embedded systems are nowadays a corner stone in research for areas such as Internet of Things. An interesting aspect regarding these systems is the fact that they are tied to strong restrictions (computing power, power source) This paper has presented a method to create kernels tailored for specific hardware. Doing this, gains in performance, especially when libraries can be run optimized for the target hardware and intermediate layers are avoided. An interesting upturn in power consumption is presented, running kernel code and user code that fits the hardware requirements results in power savings.

We intend to follow this research path in order to commit our intention to create a lower substrate layer that provides efficient and fast conditions for sensor networks.

Agradecimientos

First author is supported by Junta de Andalucia through grant P11-TIC-07176. Second and fourth authors are supported through grant "Análisis de la calidad de la energía eléctrica empleando contadores inteligentes. Optimización y ahorro en el sector productivo y residencial de Andalucía".RNM-6349. Third author is partially suppported by Ministerio de Educacion, Cultura y Deporte grant Salvadorde Madariaga PRX14/00121, Ministerio de Economia y Competitividad grant MTM2014-54439 and Junta de Andalucia (FQM0211).

Referencias

[1] CELF(Consumer Electronics Linux Forum), http://tree.celinuxforum.org/CelfPubWiki/BootupTimeResources, Online, Cited: June 7, 2015.
[2] J. Lombardo, *Embedded Linux*, New Riders, 2002.
[3] A. Leppakoski; E. Salminen; T.D. Hamalainen, Framework for industrial embedded system product development and management, System on Chip (SoC), 2013 International Symposium on , vol., no., pp.1,6, 23-24 Oct. 2013 .
[4] Linux Foundation, "Yocto Project I Open Source embedded Linux build system, package metadata and SDK generator", Online, Cited: June 7, 2015, https://www.yoctoproject.org.
[5] O. Salvador; D. Angolini, Embedded Linux Development with Yocto Project, Packt Publishing,2015 .
[6] Nathan Binkert, Bradford Beckmann et al., The gem5 Simulator, Online, Cited: June 7, 2015, http://research.cs.wisc.edu/multifacet/papers/can11_gem5.pdf.
[7] W.D. Diffie, M.E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22(6), pp. 644–654, 1976.
[8] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48(177), pp. 203–209, 1987.
[9] A.J. Menezes, Y.H. Wu, "The discrete logarithm problem in $GL(n, q)$," *Ars Combinatoria*, vol. 47, pp. 23–32, 1997.
[10] V. Miller, "Use of Elliptic curves in Cryptography," *Advances in Cryptography – CRYPTO'85*, vol. 218, *Lecture Notes in Computer Science*, pp. 417–426. Springer-Verlag, New York, NY, 1986.
[11] Q. Niu, ECDH-based Scalable Distributed Key Management Scheme for Secure Group Communication, J. Computers. 9(1), 153–160, 2014.
[12] R.W.K. Odoni, V. Varadharajan, P.W. Sanders, "Public key distribution in matrix rings," *Electronics Letters*, vol. 20, pp. 386–387, 1984.
[13] T. Satoh, K. Araki, "On construction of signature scheme over a certain non-commutative ring," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 80(1), pp. 40–45, 1997.
[14] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, 26(5), pp. 1484–1509, 1997.
[15] M. Steiner, G. Tsudik, M. Waidner, Key agreement in dynamic peer groups. IEEE Transactions of Parallel and Distributed Systems, 11(8), 769-780, 2000.

[16] E. Stickel, "A new method for exchanging secret keys," *Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05),* Sidney, 2005, pp. 426-430.
[17] J.A. Alvarez-Bermejo,M.A. Lodroman,J.A. Lopez-Ramos. "Group key agreement on elliptic curves" *Proceedings of the 15th International Conference Computational and Mathematical Methods in Science and Engineering,* Rota, Cádiz, 2015. In-press