Design and Implementation of a Low-Cost Embedded Linux Gateway for Smart Home Health Monitoring

Mateusz Mittek  
*University of Nebraska-Lincoln, mmittek@unl.edu*

Jay D. Carlson  
*University of Nebraska-Lincoln, jcarlson@unl.edu*

Lance C. Pérez  
*University of Nebraska-Lincoln, lperez@unl.edu*

Follow this and additional works at: [http://digitalcommons.unl.edu/electricalengineeringfacpub](http://digitalcommons.unl.edu/electricalengineeringfacpub)

Part of the [Computer Engineering Commons](http://digitalcommons.unl.edu/computerengineering), and the [Electrical and Computer Engineering Commons](http://digitalcommons.unl.edu/electricalengineering)
Design and Implementation of a Low-Cost Embedded Linux Gateway for Smart Home Health Monitoring

Mateusz Mittek, Jay D. Carlson, and Lance C. Pérez
Department of Electrical Engineering, University of Nebraska-Lincoln,
Lincoln, NE 68588, USA
Email: {mmittek, jcarlson, lperez}@unl.edu

Abstract—Many wireless sensor network applications require a gateway device to interface with services running on the Internet. Because of the software complexity involved in this device, it is often realized using a real-time operating system running on an application processor. Most systems burden the user with developing the protocol handling and device configuration and management inside the application. In this paper, we present the Angelos Gateway—a turnkey, low-cost, Linux-powered WSN gateway that provides a socket-based environment for rapid network-enabled application development. Experimental results demonstrate that the proposed device is capable of high-throughput packet I/O, confirming the efficacy of the proposed implementation.

I. INTRODUCTION

One popular application of wireless sensor networks (WSN) is in smart home automation and monitoring systems. These systems often employ low-cost sensor motes deployed in residential environments to measure environmental conditions, localize individuals living in the home, and actuate windows, lights, and other systems. While many homes are equipped with 802.11 (WiFi) networking, WiFi-enabled motes are expensive and often have power requirements making them unsuitable for minimally-invasive deployment. Because of this, many systems use alternative protocols and topologies that are designed to be low-power and easy to implement on low-cost hardware. One popular example is 802.15 (ZigBee), though other protocols, such as Texas Instruments’ SimpliciTI, are also popular.

In many applications—especially smart home systems—the WSN does not operate alone; it must communicate with other servers, services, and devices—usually over TCP/IP. Because of this, one of the most important elements of a WSN is the gateway, which bridges the local WSN and the Internet (or private intranet). Depending on the gateway architecture and its processing power, it might also provide data processing, as well as a flexible interface allowing status monitoring, reconfiguration, debugging or even firmware updating. The gateway often has significantly higher processing power than a typical WSN end device, and is typically not required to be battery operated. All gateways have a WSN transceiver to communicate with the other WSN motes, a network transceiver (usually in the form of WiFi, Ethernet, or 3G cellular), and a processor for running software linking these two interfaces together. In the research community, a PC often serves the purpose of the gateway in this scenario—but PCs have limited usefulness in real-world deployments because they are bulky, expensive, and power-hungry.

Embedded operating systems provide task and memory management, as well as a unified hardware abstraction layer, pre-built software components, and useful documentation for building embedded applications. While there are many embedded operating systems, GNU/Linux has mature support for many different processors and platforms from all major silicon manufacturers. Running GNU/Linux on an embedded device can provide a rich environment in which to quickly develop applications, while providing stability required for reliable operation. GNU/Linux can comfortably run many WSN applications on embedded processors operating in the 200-500 MHz range; though the latest easily-available ARM application processors have as many as 4 cores, and can operate at speeds in excess of 1.5 GHz. To this end, there is a rich ecosystem to choose from when developing WSN gateway hardware with GNU/Linux.

This paper presents the iterative design process of implementing the Angelos Gateway: a high-performance, low-cost platform designed for smart home health monitoring applications. Angelos Gateway is capable of aggregating WSN data, pre-processing it, and relaying it to a centralized server over the Internet.

II. BACKGROUND AND RELATED WORK

Early work in general-purpose WSNs has shown that a multi-interfaced, embedded gateway utilizing a web-based administration and reporting interface improves the ease of use and flexibility of the WSN [1]. Y.-H. Song et all proposed a design based on a system-on-chip (SoC) with 32Mbytes of Flash Memory and 128Mbytes of SDRAM which allowed it to run embedded operating systems such as Windows CE or Linux along with a web server and services required to collect the data from the WSN. USB has become the standard bus to which the Network Interface Controllers are connected [2].

Implementation of the gateway’s WSN connectivity can be done on different levels. Four possibilities are presented in [3]. In the vast majority of WSN solutions, the gateway uses the Application-level Gateway model.

Many applications run Linux because of its ease of use and development flexibility [4], [5], [6]. Most of the time, however, off-the-shelf hardware solutions are chosen, with
research focusing on software for data and network topology management.

Lower-level approaches focus only on the link/network level connectivity between the WSN and the WAN, and are presented in [7], [8].

Erratt et al [9] presented a useful design outline for a gateway:

- Gateway should be agnostic to the underlying WSN protocol stack, so the gateway applications are not tied to a specific WSN network stack. This should allow the gateway to work with virtually any type of WSN as any sort of stack-specific customization has to be defined in a specific module.
- The gateway should easily interconnect the WSN and the WAN in a well-defined, natural and standard way preferably using existing protocols.
- The gateway should allow easy application development including two-way communication, reporting, and control. The final user should not have to modify any core functionality to implement his application. To achieve this, the core communication functionality has to be separated from the application framework.

The communication between the WSN transceiver and the gateway application is an often-overlooked component of the system. Most approaches use a WSN transceiver that has an embedded microcontroller connected to the gateway device using a high-level hardware interface such as USB or asynchronous serial (UART/RS232). Usually, this device is accessed through a generic Linux character device, accessible through special device file in "/dev" directory. As a character device, operations are limited to read and write, which means any radio configuration (for example, operating frequency, modulation scheme, and bitrate) must be either hard-coded in the transceiver’s firmware external to the Linux system altogether, or a custom, userspace protocol must be written to expose an API to modify these configuration parameters.

This ad-hoc aproach ignores a mature, well-established framework for sending and receiving packets over a configurable interface – the Linux networking stack. By implementing the WSN transceiver as a Linux network device, a rich API exposing packet handling, queuing, and device configuration operations is exposed to the userspace application.

III. IMPLEMENTATION

The proposed gateway device was designed to facilitate rapid application development, while being extremely low-cost, making it useful for a wide variety of WSN applications including (and outside of) smart home health monitoring. The proposed gateway consists of an ARM application processor running GNU/Linux, a 900 MHz transceiver (to communicate with the WSN), and USB connectivity supporting both 802.11 (WiFi) and cellular modem communication. A Linux kernel module exposes the WSN transceiver as a Linux network device,

---

**TABLE I**

COMPARING DIFFERENT WSN GATEWAY PLATFORMS

<table>
<thead>
<tr>
<th></th>
<th>Mod5270</th>
<th>Imote2</th>
<th>Stargate</th>
<th>Proposed System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>32-bit (Coldfire)</td>
<td>32-bit (ARMv5)</td>
<td>32-bit (ARMv5)</td>
<td>32-bit (ARMv5)</td>
</tr>
<tr>
<td>MCU</td>
<td>MCF5270</td>
<td>PXA271</td>
<td>PXA255</td>
<td>i.MX233</td>
</tr>
<tr>
<td>Max Frequency</td>
<td>147 MHz</td>
<td>416 MHz</td>
<td>400 MHz</td>
<td>454 MHz</td>
</tr>
<tr>
<td>RAM</td>
<td>2 MB</td>
<td>32 MB</td>
<td>32 MB</td>
<td>64 MB</td>
</tr>
<tr>
<td>Storage</td>
<td>512 KB</td>
<td>32 MB</td>
<td>32 MB</td>
<td>16 GB</td>
</tr>
<tr>
<td>Features</td>
<td>UART, SPI, I2C, Ethernet</td>
<td>UART, SPI, I2C, SDIO, USB, I2S, AC97</td>
<td>UART, I2C, PCMCI, CF</td>
<td></td>
</tr>
<tr>
<td>Development</td>
<td>GCC</td>
<td>Linux</td>
<td>Linux</td>
<td>Linux</td>
</tr>
<tr>
<td>Environment</td>
<td>$79</td>
<td>&gt; $400</td>
<td>$425</td>
<td>$22</td>
</tr>
</tbody>
</table>

![Fig. 1](image_url) The proposed Angelos Gateway uses a Freescale i.MX 233 SoC coupled with a Texas Instruments CC1101 wireless transceiver to provide a WSN-to-Internet gateway for our smart home health monitoring system.
which allows the userspace application to harness the Linux network stack for queuing, while also providing an agnostic interface to the WSN, satisfying Errat’s first requirement. The WSN-is-a-network-device approach also means routing between the WSN and the Internet can be accomplished entirely using robust, mature off-the-shelf software, such as Linux’s `iptables` system, which satisfies Errat’s third requirement.

A. Early Prototypes

Our early work on building a smart home health monitoring system used .NET Micro Framework (NETMF) as a software platform, with hardware from GHI’s .NET Gadgeteer [10] line. NETMF works by running a virtual machine on the target processor, which executes NETMF intermediate bytecode. Programs are generally written in C#, which is a high-level object-oriented language providing native support for dynamically-allocated objects, exceptions, and events. The NETMF environment provides implicit OS functionality through threading (and thread-safe data types), as well as hardware and communication abstraction through a hardware access layer (HAL). On the hardware front, the NETMF implementation used the STM32F4 microprocessor, which is a 168 MHz Cortex-M4 processor.

Results from this work showed that the development time can be drastically reduced as development of entire WSN network stack can be accomplished within hours. NETMF however, introduced a severe performance handicap, and provides poor low-power support, which is critical for most WSN systems [11].

Because of these challenges, we migrated to a bare-metal C implementation of the WSN. This is the traditional way embedded applications are developed; C provides a minimal performance hit when compared to assembly language, while allowing modular software development that is much more efficient than assembly programming. The gateway design used the same STM32F4 microprocessor used in the NETMF development. This processor was connected to a Texas Instruments CC3000 WiFi SoM (system-on-module) [12] and a SIMcom SIM900 3G GSM cellular radio SoM [13]. Both of these modules use embedded TCP/IP network stacks, providing a high-level interface to the MCU.

Handling multiple communication modules (wireless, radio, and cellular) required using the tasking capability of a Real-Time Operating System (RTOS) – CooCox OS[14] was selected for its open-source license and simplicity. Although the WiFi and GSM cellular radio modules provided a TCP/IP stack, the user application had to manage these devices in a relatively low-level fashion that made complex device-to-server communication cumbersome. As a result, an embedded Linux-based solution was explored.

B. Proposed Embedded Linux Hardware

The proposed device, while being a research platform, was also designed from the ground up to be easily manufactured, assembled, and deployed. In the future, the smart home health monitoring system will be deployed across many homes simultaneously, so using traditional prototyping techniques (hand-wiring together expensive development boards) was out of the question. The Freescale i.MX 233 consumer SoC (system-on-chip) was selected as the gateway’s microprocessor. This ARM9 processor is clocked at 454 MHz and provides extremely high integration – the chip provides DC/DC converters, linear regulators, USB host support, audio CODEC and headphone amplifiers, an LCD interface, and several USARTs. While obviously designed for handheld portable media players, the processor’s high level of integration makes board design and layout only marginally more complicated than that of a single-chip bare-metal microcontroller platform.

Like most application processors, the i.MX 233 has no onboard flash memory, and has only a minimal amount of SRAM. Therefore, external storage and RAM are required to bring up the platform for any meaningful work. Instead of relying on PCB-mounted NAND flash memory, the proposed board boots directly from a MicroSD card, which dramatically simplifies device upgrades and booting from different development images; the MicroSD card is simply swapped in the field with a replacement. For RAM, the proposed device uses 64 MB of DDR memory.

The proposed device uses the same wireless transceiver — the Texas Instruments CC1101 — used in previous designs. This is an SPI-interfaced sub-GHz transceiver designed for the 900 MHz ISM band. We have found the 900 MHz
band works well for smart home applications, due to better signal penetration through building materials than the 2.4 GHz systems.

C. WSN Network Interface

The Gateway communicates with the transceiver using the Serial Peripheral Interface (SPI) and a single GPIO interrupt pin. The interrupt pin is asserted on successful packet reception; this fires a hardware IRQ on the application processor which invokes the packet receiving procedure in a kernel thread. This contrasts with the traditional approach of using a separate embedded microcontroller responsible for managing the transceiver. While providing a lower-cost solution that unifies all firmware on a single microprocessor, this introduces a stringent requirement on the application processor: this interrupt must be serviced for each packet received; if the Linux scheduler is unable to run this task in time, the packet will be lost. The Linux kernel module was developed following the guidelines from [16].

To make the Angelos Gateway applications agnostic to a particular WSN transceiver radio protocol, it was decided to implement all hardware-related details within a kernel driver module that exposes a network interface. The main principle of this approach was to use components, APIs, and interfaces already present in the GNU/Linux operating system, such as network packet routing, queuing, and IO-abstraction. An overview of the difference between traditional implementations of an embedded Linux gateway and that of the Angelos Gateway is presented in Figure 2.

Although there is a raw packet interface that allows arbitrary arrays of bytes to be sent and received through the interface, using this approach would require address, port, and protocol processing to be implemented by the userspace application. Instead, the Angelos Gateway exposes an OSI-compliant interface allowing the device to bind to ports and perform IP-like addressing.

According to the OSI model, Internet Protocol (IP) along with TCP or UDP correspond to the Network and Transport network layers (Figure 3). In general, IP packets are simply datagrams delivered using a best-effort approach but without guarantee of delivery. Transport-layer protocols (such as TCP or UDP) are encapsulated inside IP frames. Both UDP and TCP use the port number to distinguish different applications or services using the same physical interface.

Selecting between TCP and UDP largely depends on the application. For smart home health monitoring, the system is concerned with streaming and capturing continuous, real-time data. The proposed stack is based on the Texas Instruments SimpliciTI protocol. The platform communicates with the WSN nodes using the SimpliciTI frame, while re-tooling the link layer to optimize the protocol for real-time streaming. Unlike the off-the-shelf SimpliciTI protocol, the proposed system uses a connectionless link with a maximum payload size limited by the transceiver’s built-in buffer. Because of this, the Gateway exposes a UDP – rather than TCP – socket interface to communicate with the WSN. Reliable communication is, therefore, the responsibility of the application – not the underlying transport layer. Both UDP/IP and SimpliciTI packets are presented in figure 4. The corresponding header fields are indicated using the same color.

D. Software

The Angelos Gateway firmware uses a custom GNU/Linux distribution built with Yocto, an open-source embedded Linux build system, and package metadata and SDK generator. The Yocto environment provides an easy solution for generating ready-to-use system image files which can be transferred to flash memory and used as the startup drive of the system board. Additionally, the environment automatically cross-compiles toolchains and libraries needed for application development.

IV. Implementation Verification

To validate the functionality of the implementation of the WSN gateway transceiver network device linux kernel driver module, an experiment was designed to measure the maximum packet I/O capabilities of the proposed solution between the Angelos Gateway and a TI CC1111 Development Board [17] connected to a BeagleBone Black [18] embedded GNU/Linux development board. The BeagleBone Black and CC1111 serve as a reference platform to compare the proposed system against.

The experiment focused on the ability to quickly send or receive multiple short packets at a relatively high rate; this approximates the characteristics of an indoor monitoring WSN in a scenario, where the potential packet collisions are resolved by a suitable medium access protocol (e.g. TDMA). Packet size was set to 16 bytes (12 bytes of header and 4 bytes of payload). The measurements were taken in two scenarios: Gateway as a transmitter, and Gateway as a receiver. Packet generation rate was varied from 1 to 300 packets per second, in increments of 10. The upper limit of 300 packets per second was chosen based on the maximum data transmission rate of the reference platform, found empirically. For each trial, packets were sent for 60 seconds. Three trials were performed for each rate. The device under test (DUT) was placed 103.7 cm from the reference platform. The reference platform was oriented so its antenna null was orthogonal to the line-of-sight to the DUT.

Both endpoints were connected to the PC using Ethernet. A control script running on the PC was responsible for
Starting transmitter and receiver applications on the endpoints, collecting the results, and comparing the number of packets generated with the number of packets received. The results are presented in Figure 5.

As Figure 5 shows, the proposed device can sustain near-perfect transmit rates in excess of 200 packets per second. This rate falls off to approximately 10% packet loss at 250 packets per second as the device nears the maximum limits of the reference transceiver.

Receiving packets showed similar performance. With the radio receive logic operating on the application processor (instead of an embedded microcontroller, such as in the reference platform), all packet receiving must be done in a kernel thread, which introduces a nondeterministic component to the packet processing. Unlike the reference platform, which provides data queuing on the CC1111 itself, attaching the wireless transceiver directly to the application processor means that each received packet must be read out of the device by the Linux kernel driver before another packet arrives.

These results validate the implementation decision to eliminate a secondary microprocessor for managing the radio; the main application processor has sufficient resources to manage the CC1101 transceiver directly. This provides measurable cost-savings in terms of PCB bill of materials, and also system complexity arising from having multiple microprocessors running different firmware.

V. CONCLUSION AND FUTURE WORK

The proposed Gateway solution is useful, feature-rich, and inexpensive (Table I). Forward compliance is ensured by using Yocto Project-based custom GNU/Linux distribution. Using the Linux networking stack allows rapid application prototyping, as the communication is handled independently of the user’s code. The development of network-enabled applications itself can be easily and quickly performed by following generic socket programming techniques. Connectivity measurements have shown satisfying success rate in both transmission and reception. The powerful ARM9 platform, running Linux, allows performing additional processing and network management without the need of bare-metal, low-level C programming.

At this point, only unicast IPv4/UDP packet interface is available for the user. Additional investigation of newer
network-layer protocols like IPv6 will be performed, as it is anticipated that the platform will be used in the future as a product or common research platform. Additional network features, like multicasting, will also be explored.

Simultaneous, on-board processing and networking is a demanding task for the Gateway. Future performance analysis will determine the processing capacity bounds for network-enabled WSN applications.

REFERENCES


