2011

Design of high fidelity building energy monitoring system

Hao Wang
Michigan Technological University

Design of High Fidelity Building Energy Monitoring System

By

Hao Wang

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Electrical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2011

This report, “Design of High Fidelity Building Energy Monitoring System ”, is hereby approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE IN Electrical Engineering.

Electrical and Computer Engineering

Signatures:

Report Advisor _____

Department Chair _____

Date _____

ABSTRACT

Building energy meter network, based on per-appliance monitoring system, will be an important part of the Advanced Metering Infrastructure. Two key issues exist for designing such networks. One is the network structure to be used. The other is the implementation of the network structure on a large amount of small low power devices, and the maintenance of high quality communication when the devices have electric connection with high voltage AC line.

The recent advancement of low-power wireless communication makes itself the right candidate for house and building energy network. Among all kinds of wireless solutions, the low speed but highly reliable 802.15.4 radio has been chosen in this design. While many network-layer solutions have been provided on top of 802.15.4, an IPv6 based method is used in this design. 6LOWPAN is the particular protocol which adapts IP on low power personal network radio. In order to extend the network into building area without, a specific network layer routing mechanism-RPL, is included in this design.

The fundamental unit of the building energy monitoring system is a smart wall plug. It is consisted of an electricity energy meter, a RF communication module and a low power CPU. The real challenge for designing such a device is its network firmware. In this design, IPv6 is implemented through Contiki operation system. Customize hardware driver and meter application program have been developed on top of the Contiki OS. Some experiments have been done, in order to prove the network ability of this system.

Index Terms— Advanced metering infrastructure(AMI), smart plug, 802.15.4, IPv6, 6LOWPAN, RPL, Contiki OS.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Introduction to building wide smart meter in Advanced Metering Infrastructure

The earth is starving on its energy source because of human behaviors. Among all the energy consuming activities, using the electricity plays the largest part. Everything in our daily life depends on electricity, while on the other hand, we have been wasting it in our household and office buildings day after day. We are losing our energy “unconsciously” in the building: when hundreds of computers stay in sleep mode; when thermostat are tuned up when there is nobody in the room; or when thousands of light bulbs are forgotten to be turned off. We have been losing electricity energy efficiency by utilizing the high-power items at improper time. E.g., turning on the air conditioner or change the electric vehicle at peak hours, when the grid is under the lowest energy delivery efficiency of the day(and the unit price of electricity is the highest).

To improve those aspects simply by human behavior is inefficient. Today’s renovating embedded system and network technologies could be applied to help customers rid themselves off their bad energy consuming habits, save money on their bills, and eventually conserve more energy on our planet.

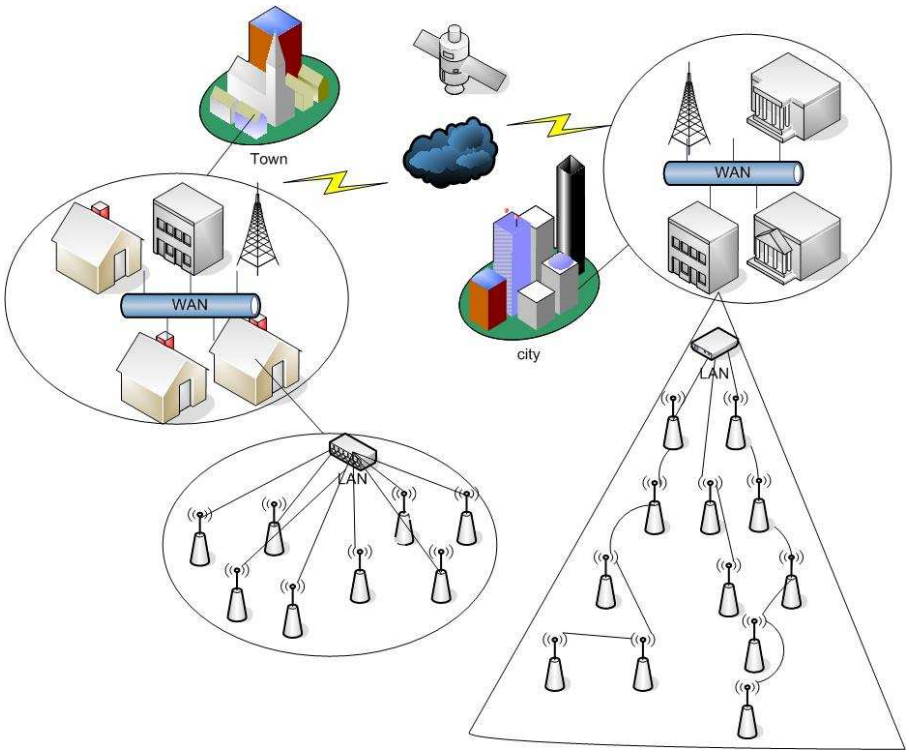

Figure 1: LAN in the AMI architechture

The industry of energy distribution network is undergoing a tremendous revolution. The main contributor for this change is the replacement of legacy electrical meters. Metering management will no longer be limited to manual reading. Instead, it is evolving from Automated Metering Reading (AMR) to Advanced Metering Infrastructure (AMI). The function of AMR is to perform automatic collection of power consumptions, load curves, alarms, as well as automatic billing. AMI will provide further capabilities with two way communications, in which a set of new and advanced applications such as dynamic pricing, demand response(DR) and grid monitoring, could be provided[1].

The main player in AMI is the smart meter. For the grid, smart meter is the outdoor monitor that can send the load curve, report outage and fault information to the grid control center through wide area network (WAN). On the other hand, the central concept of smart meter has been extended to the customer level, in which a per appliance energy management system is available, by exchanging meter data through local area network (LAN). This includes mechanisms and incentives for utilities, business, industrial, and residential customers to cut energy use during times of peak demand or when power reliability is at risk [2].

1.2 Challenges of per-appliance monitoring device

Our goal to address the above issue is to construct a large number of electricity meters on every wall plug in the building, interconnect the meters and monitor them with a server computer. As the result, two major requirements have to be met: one is to build a large scalable network; the other is to make a meter with low power consumption.

The network of this system should be reliable, but not necessarily very "strong". Instead, the size of the network should get more attention; imagine how many appliances could be running in a building. Besides, the monitoring system should be easily accessible to everyone living or working in the building. It will be perfect if anyone can monitor the building's energy chart at any time, through web

browser on a PC or their portable device. It is better if no special interfaces or gateways are needed.

The most fundamental requirement for the metering node is that it must have communication capability, with its own unique ID, so that the customer could locate each of their appliances. Such a device, which should integrate energy measurement and communication module, cannot consume too much power itself. Its power consumption proportion to the daily consumption amount of the building, must be limited to an almost neglectable value. Otherwise, the whole system is meaningless.

2. NETWORK SOLUTION

Smart meter in buildings is usually a kind of per-appliance energy management system, which belongs to Wireless Building Automation Network. Several solutions, on top of different kinds of radio frequency(RF) communications, have been given [3]. Zigbee is a four-layer protocol based on IEEE 802.15.4, operating in 868MHz, 915MHz and 2.4MHz; Z-WAVE is a five-layer network protocol that operates in 900MHz ISM bands; INNSTEON is a composition of power-line communication and 904MHz RF with FSK modulation; WAVENIS, operating in the 433MHz, 868MHz, 915Hz and 2.4GHz, is a three-layer protocol with upper layer APIs.

With regard to Demand Response, every home appliance with high power consumption has to be monitored. The NO.1 challenge is scalability. Optimized network mechanism is needed to handle the “heavy-traffic” data transmission through the large network that consists of many small objects. The main problem of the above non-IP solutions is that none of them can support a large number of devices. In addition, each of them is a closed network, and an application gateway is needed to connect the building network to the Wide Area Network(WAN). Those gateways are very complex to design and require much daily work to maintain. Therefore, they are soon going to be replaced by IP solution. Besides, considering there are a large number of devices, IPv6 should be applied instead of IPv4.

2.1 6LOWPAN

Because the per-appliance metering devices have to be low cost and low power consuming, they usually have limited resource. [4] introduced the principle about how to deploy IPv6 into the resource constraint embedded platform with relatively unstable, low bandwidth communication channel. First of all, meshed-over IP which rely on the link layer routing, is not feasible, because the resource-constraint node can never maintain a large dynamic routing table. Instead, each node must keep a neighbor cache to store the IP address of its neighbor. Secondly, IPv6 header has to be compressed. The compression is based on two key ideas: remove redundant information and assumption on common values. In addition, sampled listening mechanism should be introduced, considering the fact that idle listening of radio device may consume most power for a metering node. The time-based event triggering mechanism also utilizes the CPU's sleep mode to save power.

The above protocol of IPv6 adaptation is called 6LOWPAN, whose latest version is defined in [5].

One extinguish benefit of 6LOWPAN is that it introduces a set of head compression mechanism to deeply reduce the IP packet size. As we know, AMI messages or data are both small packets by nature; sometimes an IPv6 header might occupy the major portion of a packet (Same thing happens in sensor network, which is one reason why IP was not being used in the past). In traditional IP based P2P communication, flow-based compression is used because most packet traffics are long live flows [6]. The compression is basically to elide portions that rarely change within

a flow. Both sides of the traffic keep the same compressor and decompressor used for lifetime of the flow. In AMI application, however, the path of a flow changes very often. And changing a next-hop route means migrating compression state to the new route, so non-flow based compression must be introduced. Generally speaking, there are two ways for stateless compression. One is to remove redundant information, e.g., IPv6 header fields can derive from link-level information. The other way is to make assumption on common values, e.g., the version byte is always IPv6. Moreover, there is mechanism for subheader compression, which can further compress some kinds of flows. For example, when the datagram is small, the fragmentation header is elided; also, mesh header is elided for singe radio hop transmission. The first Network and upper layer compression mechanism in 6LOWPAN is HC1. It relies on the following assumptions:

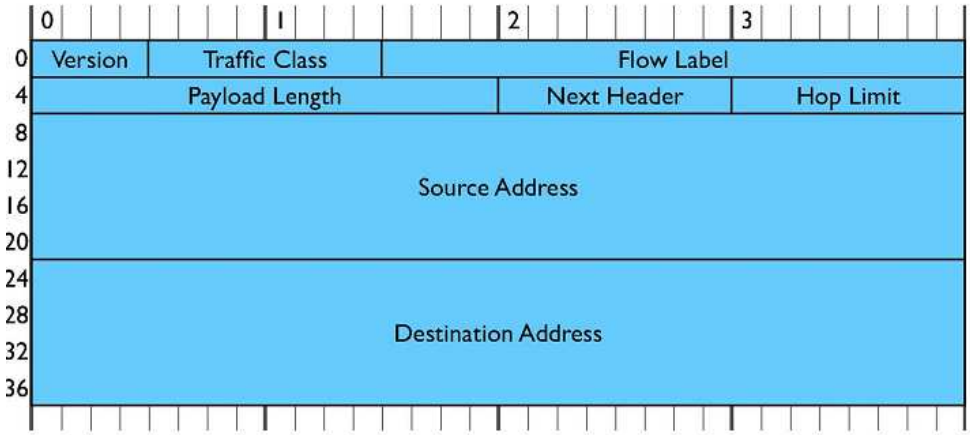

Figure 2: IPv6 hearder

First, HC1 is optimized for link-local addresses, the IPv6 interface ID can be inferred from the link layer MAC address. Second, the packet length can be inferred

from the frame length field of the IEEE 802.15.4 frame. Besides, transport layer protocol will just be one of UDP, TCP, and ICMP, so its header requires only 2 bits instead of a whole byte. These observations allow a considerable reduction of the protocol overhead. The only IPv6 header field that cannot be compressed is the 1-byte hop limit field. This leads to only 3 bytes instead of the 40-byte IPv6 header: 1 byte for the dispatch byte, followed by a 1-byte HC1 byte, and 1 byte for the hop limit field [7].

2.2 Multi-hop solution: RPL

As the Local Area Network shown on the left of figure 1, radiate topology is effective for the residential customers, whose appliances could be all within the range of Home Energy Network gateway. However, such network cannot be applied into commercial or industrial customers, where building wide network, with multi-hop capability will be necessary, like the LAN on the right of figure 1.

It is common that our LAN metering nodes are immobile, because the heavy appliances rarely need to be moved. However, since low power radio is being used, the link quality of 802.15.4 is unstable. As a result, effective routing path re-computation methods are needed [8], which means special routing mechanism is required.

RPL addresses on the lossy links in building area network, as well as the limited resource of its nodes [9]. In RPL protocol, one or more nodes are configured

as DODAG roots. A node discovery mechanism based on newly defined ICMPv6 messages is used by RPL to build the DODAG. RPL defines two new ICMPv6 messages called DODAG information object (DIO) messages and destination advertisement object (DAO) messages. DIO messages (simply referred to as DIO) are sent by nodes to advertise information about the DODAG, such as the DODAG ID, the OF(objective function), DODAG rank, the DODAG sequence number, along with other DODAG parameters such as a set of path metrics. When a node discovers multiple DODAG neighbors (that could become parents or sibling), it makes use of various rules to decide whether (and where) to join the DODAG. This allows the construction of the DODAG as nodes join. Once a node has joined a DODAG, it has a route toward the DODAG root (which may be a default route) in support of the MP2P traffic from the leaves to the DODAG root (in the up direction).

The DODAG formation is governed by several rules: the RPL rules used for loop avoidance (based on the DODAG ranks), the advertised OF, the advertised path metrics, and the policies of the configured nodes.

Figure 3 shows the process to maintain the DAG. Each time instant, a node reports its upper neighbor, which will forward the DIO message all the way up to the root. The root then send the message containing information of current rank, packet sequence number, link attribute back to the sender. In turn, the sender will update its rank information, and change its father node if necessary. The interval between each solicitation time instant is not a constant value. It is based on RPL trickle algorithm.

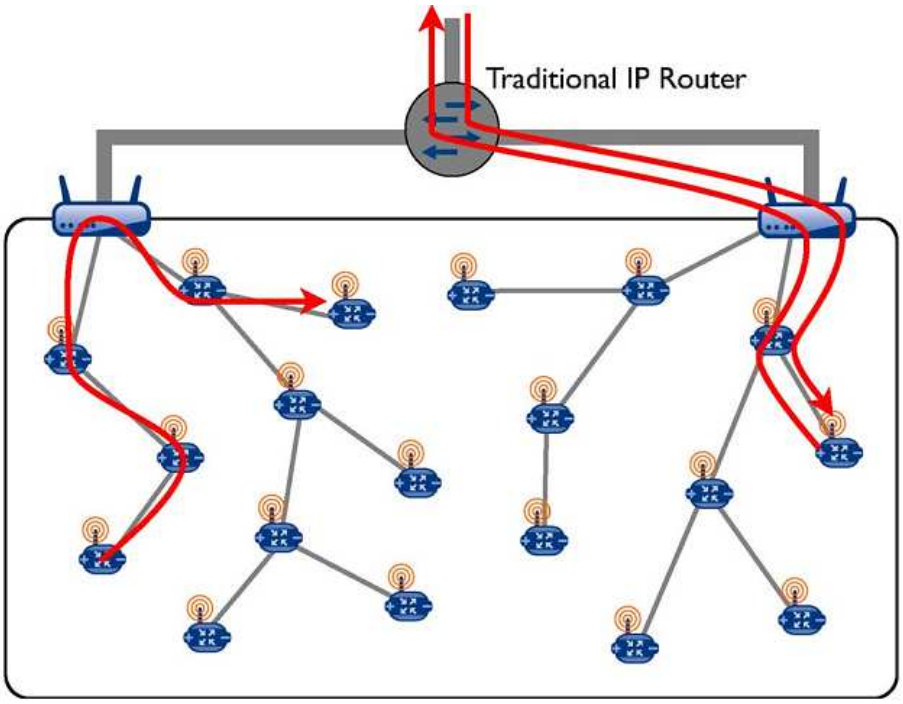

Figure 3: Routing along a DAG. [4]

The trickle algorithm uses an adaptive mechanism to control the sending rates of control plane traffic such that nodes hear just enough packets to stay consistent under various circumstances. In the presence of change nodes send protocol control packets more often and control traffic rates are reduced when the network stabilizes. The trickle algorithm does not require complex code and states in the network. This is an important property considering the constrained resources on the nodes (some implementations only require 4 – 7 bytes of RAM for state maintenance).

RPL treats the DODAG construction as a consistency problem and makes use of trickle timers to decide when to multicast DIO messages. When an inconsistency is detected RPL messages are sent more often, and then as the network stabilizes RPL messages are sent less often.

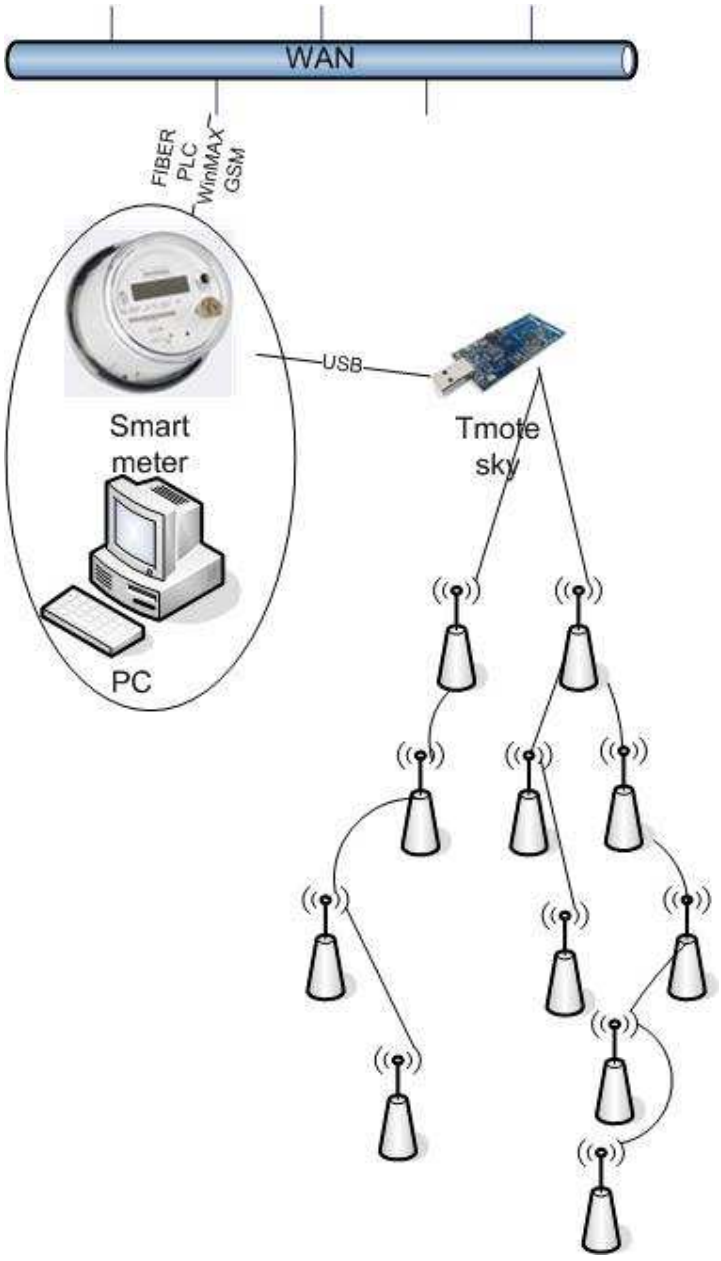

Figure 4: Temporary solution for building energy management

For the first evaluation, a temporary solution for this building energy network has been provided. In figure 4, each appliance is deployed with a digital energy monitoring circuit and wireless terminal device. Those wireless devices construct the RPL DAG, with one of moteiv's Tmote sky sensor node as its root node. A special border router application code will be programmed in the root node, which enables

the PC or smart meter to monitor the RPL network through the USB interface. -The smart meter should have all general interface of a compact PC. The PC or smart meter can read the metering data of the whole network, using a serial terminal API, telnet server or even web browser. Then it transmits the data to the remote server of utilities, through GSM, optical fiber, power line carrier, or the more recent WinMAX. In that way, the local DAG network is able to exchange its metering data with the AMI wide area network, through the PC or smart meter which acts as a bridge.

The above solution is being used temporarily, because currently not much work regarding to the gateway device has been done. In the future, the PC+Tmote solution will be replaced by a specialized router, with standardized building energy system firmware, as is shown in figure 5.

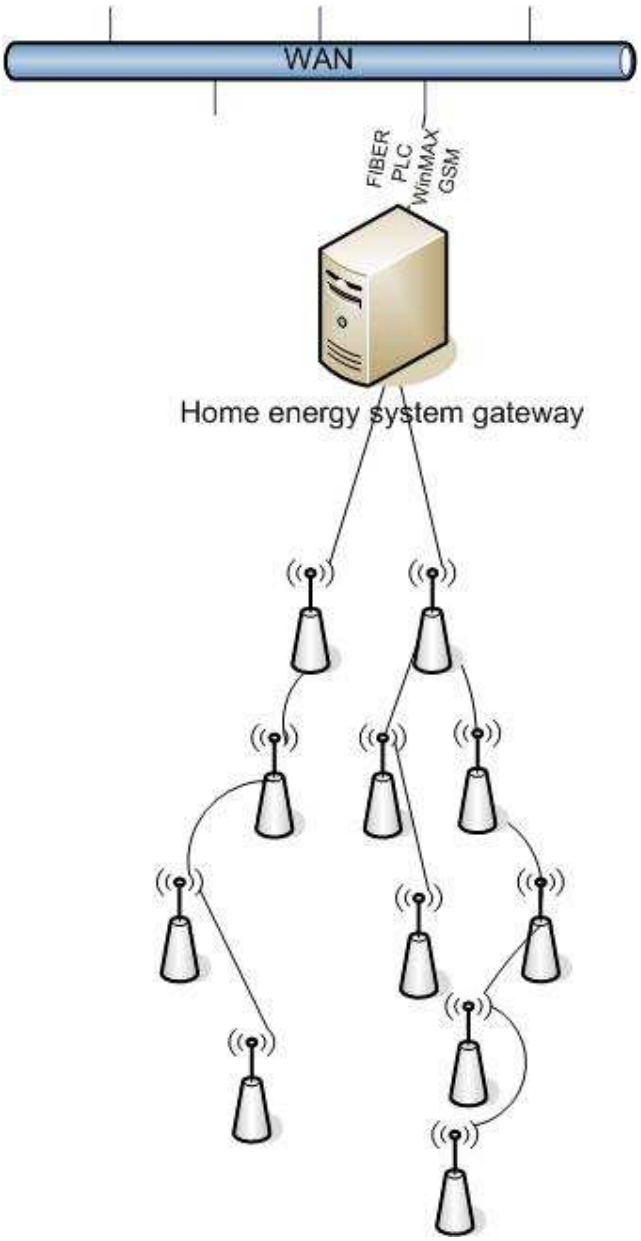

Figure 5: Ultimate system solution

3. DESIGN OF THE SMART WALL PLUG

The last chapter has discussed the network solution at the protocol level. The following chapters will explain its implementation.

First is the hardware: what components and circuits are required in such a meter node? It is an energy meter with wireless connectivity. It should consume as little energy as possible, and has a low cost. Next is the firmware, through which each node should have a unique IPv6 address so that every appliance could be monitored individually. And it will be better if any newly plugged-in appliance could be "automatically" discovered by the other nodes.

Using IPv6 as communication stack, the Berkley team has finished much work on wireless AC plug meter [10]. 802.15.4 Radio is chosen given its low power consumption and relatively high reliability. They used Tiny OS as its embedded operating system, which has the early version of 6LOWPAN integrated. Its hardware composed of an old MSP430 series microcontroller, CC2420 radio, a power meter device from Analog Devices Inc, required components that monitor the AC voltage, current, and AC-DC power supply that steps down from 110V AC input. It is quite a promising work for home appliance monitoring. Yet its MCU hardware should be updated to achieve more complete functionality and better performance. And its software, especially the network OS, should be improved to fully support the renewed 6LOWPAN.

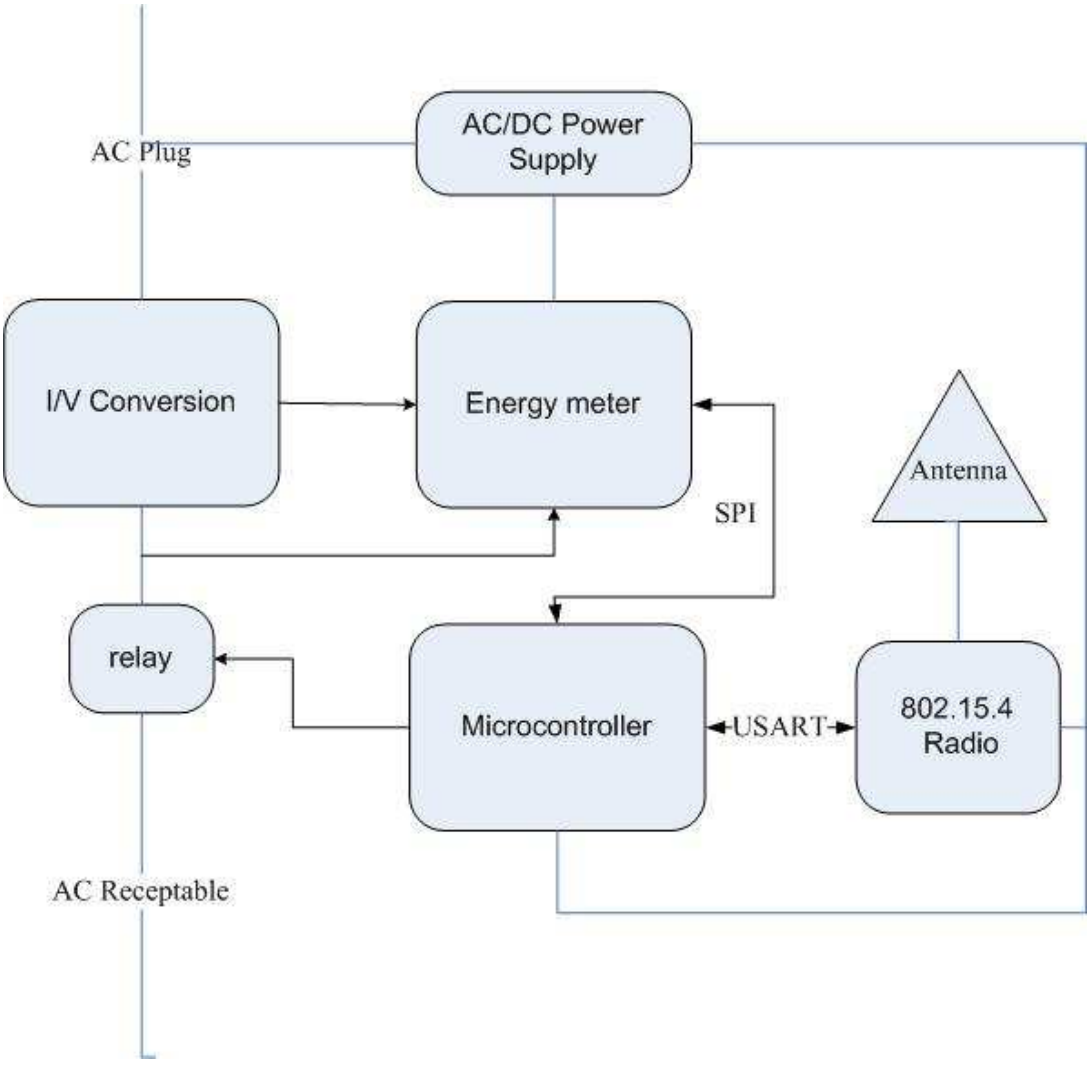

Figure 6: Architecture of meter node

This design is an improvement of their work. It is a combination of state-of-the-art hardware and software techniques, which implements both 6LOWPAN and RPL. This chapter will just briefly discuss hardware design.

Figure 6 is a sufficient solution for a single node of this per-appliance monitoring network. It composes of TI MSP430 microcontroller, 802.15.4 radio, single chip energy meter, relay, circuitry of current/voltage converter, and an OEM module of small AC/DC power supply. Currently, as a separate device, it will have

both an NEMA AC plug and AC receptacle on its enclosure. In the future we will make the platform into every wall AC plug, so that the whole building or house may have AMI on the HAN level. The energy consumption of the embedded platform must be low enough so that it will not adversely increase the standby monitoring load. Therefore, all the selected devices are low power consuming.

3.1 Considerations for AC/DC power supply

The basic motive of smart meter is to reduce the power consumption. Therefore, the efficiency of the AC/DC transformer itself must be as high as possible. In this condition, switching mode AC/DC converter is much better than line frequency transformer. Besides, it will be better if the device can be compatible with different voltage levels of AC input. Thus, a switch mode AC/DC converter, with feedback control, is selected here. As is shown in figure 7, the AC input is regulated by transistor Q1. The feedback coil of the transformer keeps the output coil at constant voltage level regardless of different AC input(220V or 110V). The drive signal “base” is a feedback of the DC output, which stabilizes the output by switching Q1. With transformer between the AC input and output, and opto-coupler between feedback drive signal and DC output, the 5V DC will accordingly be totally isolated from the AC line.

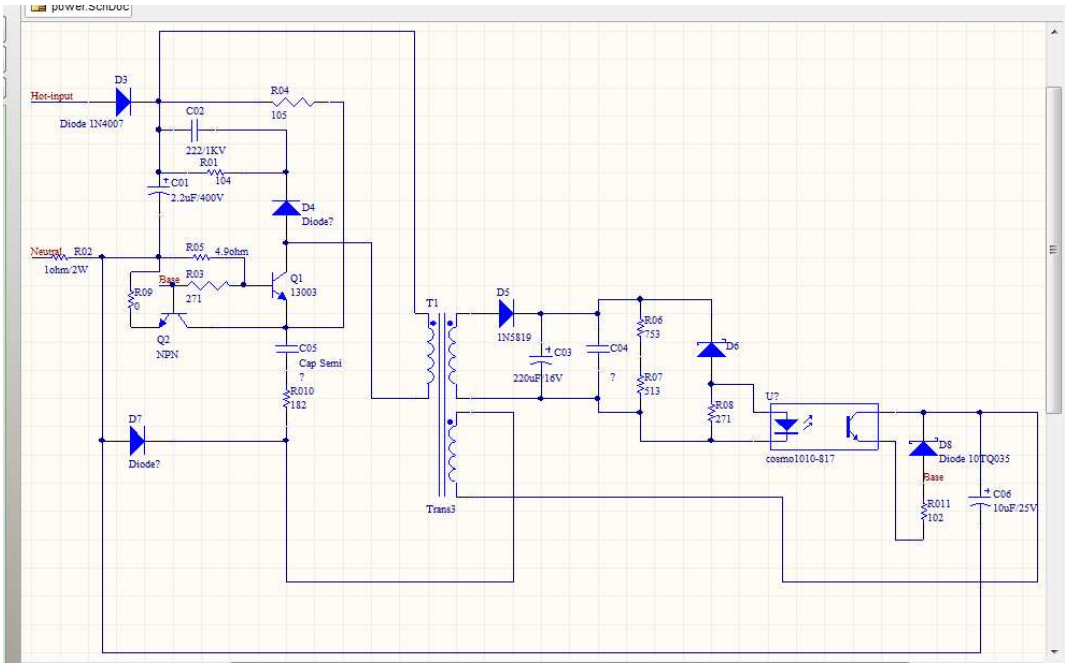

Figure 7: AC/DC power supply

In this application, the RF device is close to AC power line, thus affects to the quality of radio communication from the line frequency noise has to be evaluated. A 30 minutes test was set up for our purpose. In the test one node is plugged into a laptop, programmed as a sender as well as monitor; another two nodes are both programmed as passive receiver, which will send any data received back to the sender. One receiver is using battery power, which is totally isolated and far from the AC line. The other receiver is powered by the AC line through the AC/DC switching mode power converter suggested above. According our result in table 1, it is not hard to believe that the interference is tolerable.

Table 2: 30 minutes test of link quality of RF device

Successful transmission rate(wall plug powered)	Successful transmission rate(battery powered
96.5%	99.5%

3.2 Current to voltage conversion

The direct solution for I/V conversion is to use a shunt resistor. It is a high precision resistor in the range of mΩ. The circuit amplifies the small voltage drop across the resistor to get the AC current. This method will provide higher accuracy, but since the control circuit is physically connected to the AC line, it is in some way relatively unsafe and inefficient to isolate noise, because it requires dumping energy to ground.

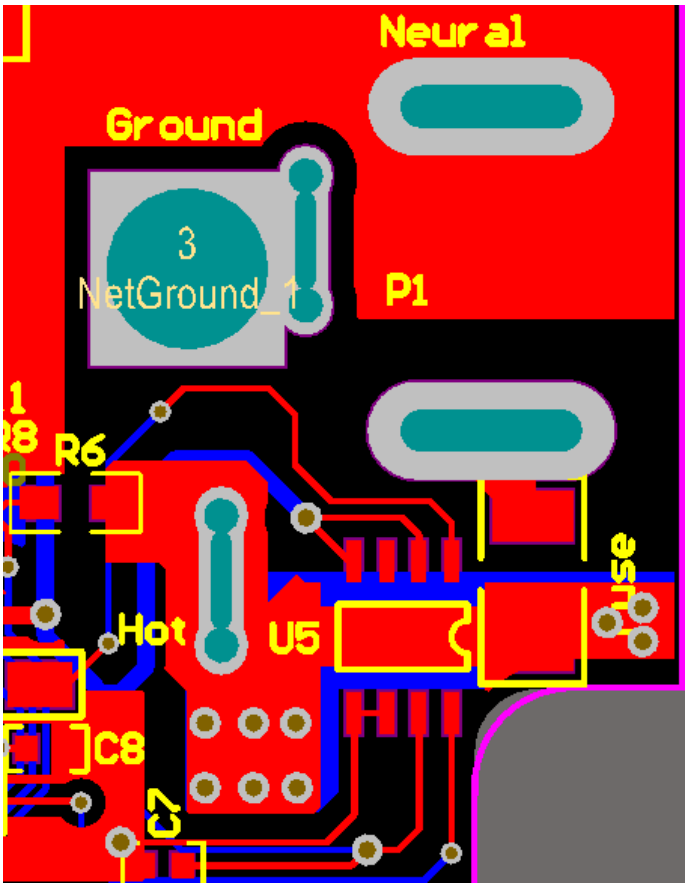

Figure 8: PCB layout of Hall-effect chip

An alternative solution is to use a Hall-effect sensor. Our choice is CSA-1V from Sentron AG. Then IC has a 35mV/A sensitivity when placed on PCB directly

over a current pack. It provides isolation between HVAC and LVDC on the circuit board. It also provides differential current output, which can eliminate zero drift error for the amplifying circuit. The only shortcoming for Hall-effect is that it will introduce a small latency(a few micro seconds). However, since the AC line current being measured is only 50~60Hz, the delay will not be a problem.

While using the on board Hall-effect sensor, there will be some requirements for the PCB Layout. For a 2-layer board, solid copper should be planed on both sides, with several 10mil vias for heat transfer. We have to carefully calculate the size of the current path, to make sure its endurance current is higher than the fuse rating. Figure 8 is a screenshot for the layout of CSA-1V.

3.3 CPU

AMI application requires the CPU has sleep mode, and maintains a very low power consumption. It should also have enough built-in flash for the code of network stacks, and internal RAM to storage data of all kinds of AMI applications. The Tmote sky developed by moteiv was using TI's MSP430F1611, which is already outdated and inefficient compared to its new generations. The best candidates now are Atmel atmega128L series and MSP430F26x series. In this design, MSP430F2618 is the choice. It has low variable supply voltage range from 1.8 V to 3.6 V. It features an ultra-low power consumption: 365 µA in active mode at 1MHz, 2.2V, and only 0.5 µA in Standby Mode (VLO). Its maximum frequency is 16MHz, which is fast enough

for low overhead network applications. It also provides several different clock settings, which can generate both short period delay and long timers(like clock ticks in second or even minute). The CPU has four Universal Serial Communication Interfaces (USCIs), so that one of them could be configured as UART, to exchange information with PC; the other three as SPIs, for RF chip, serial flash memory, and metering IC.

MSP430F2618 has 116KB+256B Flash Memory, 8KB RAM [11]. Considering the embedded OS plus the IP stack need about 45 KB Memories, this chip will well match this application.

3.4 USB-Serial Interface

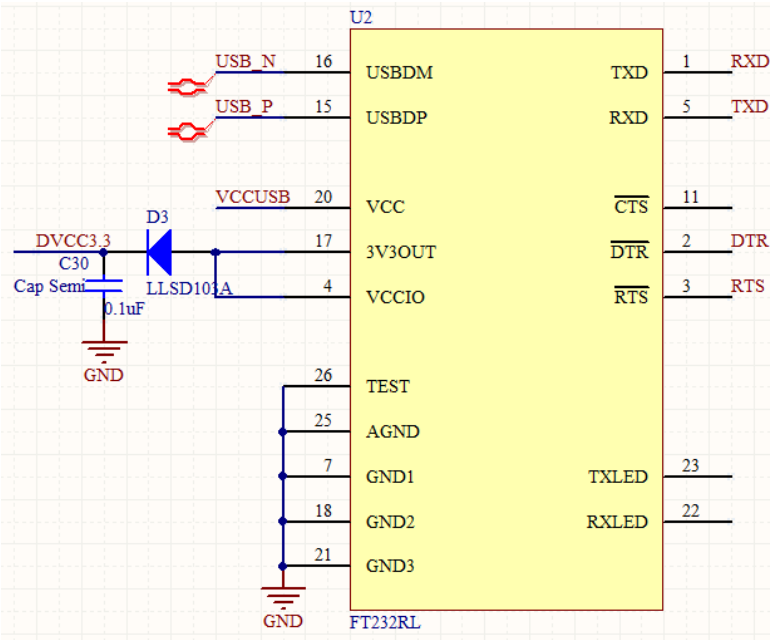

Figure 9: FT232 interface

MSP430 Bootstrap loader is being used to load system program, through UART0 and another two pins. The tool will be discussed in detail in the next chapter. Nowadays, since common PC may not have a RS-232 port, a state-of-the-art chip FT232BL is chosen to convert UART to USB, so that the user can use any PC to update the firmware program. This also provides possibility for the grid company to reprogram the node firmware “on line”. In that way, they can update their energy network monitoring system without any physical replacement. This “soft renovation” of grid seems to be another great prospective of AMI. Moreover, the USB-Serial interface is also used for gateway nodes, since those nodes need exchange information with host PC and network router.

3.5 Radio module

The speed of wireless communication is not that critical, but the robustness must be guaranteed. In this condition, the 802.15.4 WPAN radio is better than 802.11, or Wi-Fi.

In order to release the device with basic functionality as soon as possible, a RF module with integrated antenna is selected. The module is Easybee. It is using an IEEE 802.15.4 compliant RF transceiver IC, CC2420 from Texas Instrument. It enables designers to easily add ZigBee/IEEE 802.15.4 wireless capability to their products without the RF or antenna design expertise. The module contains all RF circuitry, including integral antenna and controller in a simple-to-use, plug-in or

surface mount module. A 4-wire SPI port is provided as interface to a baseband microcontroller. This module is an application-ready solution for IEEE 802.15.4 communications, providing the developing engineer unlimited choice of host controller and stack firmware provider [12].

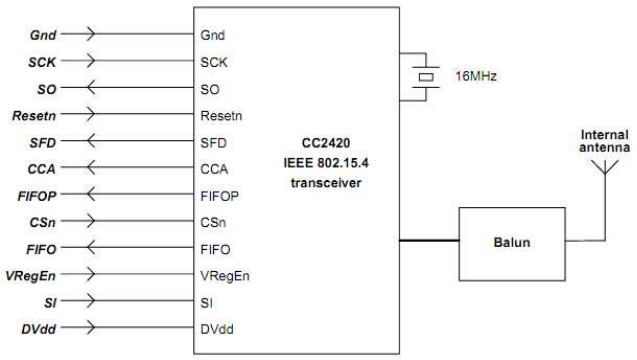

Figure 10: EasyBee Functional Block Diagram

3.6 Energy metering

Two methods can be applied to measure the power. One is using two on-chip Analog-Digital Conversion channels, which simultaneously sample the AC current and voltage signal. And calculate the real and reactive power through Discrete Fourier Transform. This approach will cost less and consume lower power, since no external device is needed. However, the accuracy of the measuring cannot be guaranteed, especially when power consumption on the node is low. First reason is that the delay between readings two different channels can only be reduced, but never avoided. Moreover, the result may also be confined by limited processing ability of low power Microprocessor, with regard to DFT calculation of a large number of points.

Therefore, another method is chosen for energy measurement, utilizing the special metering IC. The ADE7753 features proprietary ADCs and DSP for high accuracy over large variations in environmental conditions and time. The chip incorporates two second-order 16-bit Σ-Δ ADCs, a digital integrator (on CH1), reference circuitry, temperature sensor, and all the signal processing required to perform active, reactive, and apparent energy measurements, line-voltage period measurement, and rms calculation on the voltage and current. The selectable on-chip digital integrator provides direct interface to di/dt current sensors such as Rogowski coils, eliminating the need for an external analog integrator and resulting in excellent long-term stability and precise phase matching between the current and voltage channels [13]. The ADE7753 provides a SPI interface, and a pulse output frequency (CF) which is proportional to the active power. In that way, the microprocessor can precisely get the real-time energy consuming value by merely sending its command and reading the serial data through SPI interface.

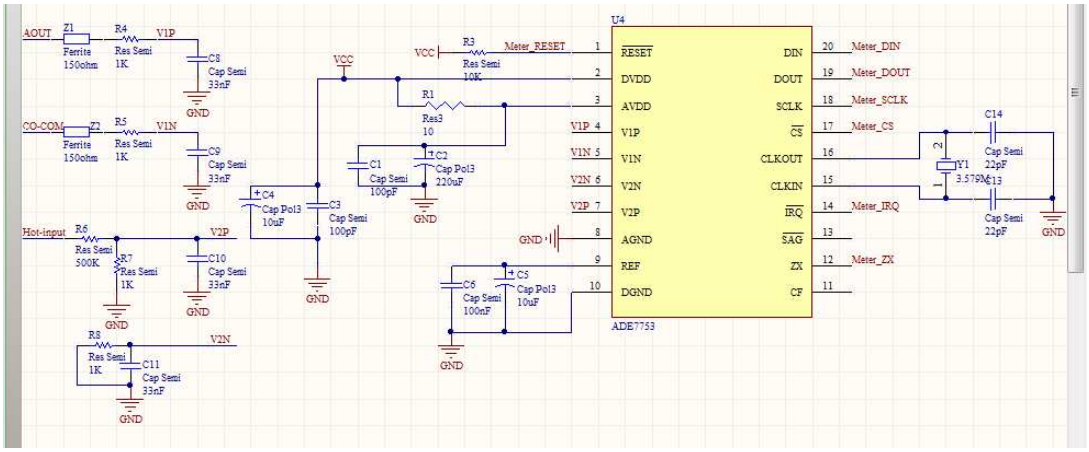

Figure 11: Energy reading circuit

Figure 11 is the energy measuring circuit using ADE7753. The high voltage AC is connected to the channel 2 using a voltage divider formed by resistor R5 and R7. “AOUT” AND “CO-COM” are the differential output of the Hall-effect IC. The output will then be applied into channel 1 of ADE7753, through a low pass filter. In this way, the CPU could always get the true RMS value of current, eliminating the transient surge.

4. IMPLEMENTING THE FIRMWARE

Researchers and engineers used to believe that IP is too heavy for embedded platforms. Recently, some new emerging embedded operating systems have integrated lightweight TCP/IP stack, among which Contiki is the most popular one. Thus our solution will be based on this operating system.

This chapter first briefly introduces structure and key features of Contiki, then focus on how to implement Contiki OS in the above platform. A bunch of developing tools that would be required for running the firmware in MSP430 CPU will be discussed, followed by the explanation of designing our hardware drivers.

Application software design specific for the energy meter network will be explained in next chapter.

4.1 Introduction of Contiki

Contiki is an open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and wireless sensor networks. The OS is designed for microcontrollers with small amounts of memory [14].

Contiki is an event driven operating system, which means every execution of a part of the application is a reaction to an event. The entire application has an OS kernel, many core libraries and some user application programs. It may contain several processes that will execute concurrently. In Contiki, a process is a C function most likely containing an infinite loop and some blocking macro calls. Different

processes usually execute for some time, and then wait for events to happen. While waiting, a process is said to be blocked. When an event happens, the kernel executes the process that is passing information about that event.

The Contiki uIP communication stack is quite flexible, which can be categorized into two basic types: uIP and Rime. The Rime communication stack provides a set of lightweight communication primitives, ranging from best-effort anonymous local area broadcast to reliable network flooding. The flexibility of Rime makes itself another name “chameleon” [19].

The main contribution of Contiki is a lightweight TCP/IP stack, which implements RFC-compliant IPv4, IPv6, TCP and UDP. Needless to say, the IP has to be tailored. It is highly optimized. It will not support all specific applications of normal TCP/IP stack. Instead, only the required features are implemented. Particularly, the OS provide the programmer a set of APIs, written in C programming language. Table 2 shows the directory structure of Contiki.

Table 2: Contiki file system

apps/ – architecture independent applications(One subdirectory per application)
core/ – system source code(Subdirectories for different parts of the system)
cpu/ – CPU-specific code(One subdirectory per CPU)
doc/ – documentation
examples/ – example project directories(Subdirectories with project)
platform/ – platform-specific code(One subdirectory per platform)
tools/ – software for building Contiki, sending files

The Contiki team has recently released their support for the ROLL RPL-ContikiRPL, which includes most RPL protocols mentioned earlier in this paper.

In ContikiRPL, the DODAG joining process and re-computation sequence are both done by the OS kernel.

Once the OS reboot on a node, it will automatically search for the DAG root. If none of its neighbors claims to be root node, it will send its DIO message to all the neighbors. The neighbor will pass its message to the root, and the root will decide the rank information of that node, and send it back(again forwarded hop by hop). Those APIs are defined in the file rpl-dag.c. The time interval for the node to send solicitation message depends on the stability of its link status. If its rank information changes more often, the message will be sent more frequently. Such kind of mechanism is defined in the file “rpl-timers.c”. It uses callback timer called ctimer, to set the DIO timer.

The routing protocol uses Contiki’s modular IPv6 routing interface. This interface has three functions: activate, deactivate, and lookup. The activate function initializes the DAG construction by sending a DAG Information Solicitation (DIS). Neighbors that belong to a DAG will reset their Trickle timers, and shortly thereafter the node will receive at least one DIO. The deactivate function de-allocates internal structures and sends a no-DAO to the node’s neighbors. After deactivation, the module stops responding to route-lookup requests, but may be reactivated later. If uIPv6 detects that the destination for a packet is not an immediate neighbor, it asks its location in the DAG using the lookup function [15].

4.2 Developing tools

Several tools are needed to build customize hardware drivers into Contiki OS.

In this design, the CPU is TI MSP430. Bootstrap loader(BSL) is being used for programming the CPU. BSL provides a method to program the flash memory during MSP430 project development and updates. It can be activated by a utility that sends commands via the UART protocol. The BSL enables the user to control the activity of the MSP430 and to exchange data using a personal computer or other device. To avoid accidental overwriting of the BSL code, this code is stored in a secure memory location, either ROM or specially protected flash. To prevent unwanted source readout, any BSL command that directly or indirectly allows data reading is password protected. To invoke the bootstrap loader, a BSL entry sequence must be applied to dedicated pins. After that, a synchronization character, followed by the data frame of a specific command, initiates the desired function [16].

The contiki OS is developed under Linux, under which MSPGCC is the most popular compiler for the Texas Instruments MSP430 family ultra low power MCUs. It includes the GNU C compiler (GCC), the assembler and linker (binutils), the debugger (GDB), and some other supporting tools. These tools can be used on Windows, Linux, BSD and most other flavours of Unix. However, the full debug environment is currently limited to Windows, Linux and BSD [17]. The latest version is mspgcc3.2.3 for old msp430 family and mspgcc4.4.1 for the new msp430x26x family.

Using the GNU as compiler, the Contiki Build system has multiple Makefiles to manage its complex file structure. While running a project on Contiki OS, the user can just type the following line:

“Make TARGET=$PLATFORM PROJ.upload”.

With that command, the whole application program, bundled with the OS, will be uploaded into the MSP430 CPU of the custom platform.

In most cases, the application project will be built in the /example directory, but hardware specific code should not be done here. Instead, hardware APIs are written in /platform directory, and some frequently used upper level APIs are defined in /apps. When the compiling begins, the compiler first looks into the current directory /example/$PROJ and subfolders, then searches for /platform/$PLATFORM, and /cpu/$CPU thereafter. Finally it will look into the /core directory. When making the OS compatible with a new platform, some modifications are needed for hardware specific files located in /core or /cpu. However, instead of doing that, it is better to write a new file with the same name in the /platform. The compiler will recognize the new file based on the above searching sequence, so that the OS kernel stays unchanged.

The rules for compiling are actually managed by Contiki Makefiles. Those Makefiles specifies source files, as well as directories of source files in use. The platform-specific Makefile, residing in /platform/$platform, will include CPU-specific Makefile. The CPU-specific Makefile-Makefile.$(CPU), is located in /cpu/$(CPU), or simply in platform/dev. It defines CPU-specific source files, makes rules for C

compiler (in this case is MSPGCC), or the virtual path for GNU to search for .c files [18]. Yet Like the CPU-specific Makefile, Makefile.$platform should never be invoked directly either. A top-level Makefile-Makefile.include, will include all the platform-specific Makefiles. The programmer will develop their application code in the /example/$project, where there is a Makefile that includes the Makefile.include. This is the final location to run the “make” command. The following shows a few lines of Contiki Makefiles:

```
# …
CONTIKI_TARGET_DIRS = . dev apps net
# …
CONTIKI_TARGET_SOURCEFILES +=
# …
include $(CONTIKI)/cpu/msp430/Makefile.msp430
# …
```

4.3 Design the hardware specific driver

Since an earlier family of TI MSP430 CPU-MSP430F1000x, has already been fully supported by the Contiki OS, the amount of work in this project could now be greatly reduced. In other words, some msp430 files in /cpu/msp430 could still be used. Following are modifications to update the hardware divers for MSP430F2000 family. First, in order to support new hardware devices, some files located in platform/, or /platform/dev have to be reedited. What is more, certain modification of cpu files are needed to update from older msp430 architecture.

Contiki is event driven system, so event timer is the first objective when porting to a new platform. The timer interrupt setting from the clock module(which locates in the clock.c), has to be redefined. Fortunately, the Makefile.msp430 in the /cpu does not have to be changed. Most hardware specific constants like I/O address, bit rate of serial port, or sensors reference value, is defined in the fine platform-conf.h. So the device specific ".c" files stay the same even if the different interfaces are used. In this design, the MSP430 DCO and ACLK frequency divider have been redesigned, in order to get speed resynchronization for more robust UART. Other I/O resource redefinitions are SPI bus, Interrupt pin and GPIO, etc.

The Contiki main file locates in the /platform/$PLATFORM directory. It is the main function of the whole program. It begins with all the initialization sequence of the CPU resources and peripherals, followed by some network parameters definitions and system function initialization. The main loop of program is right after, which periodically to restarts the watchdog timer, calls the necessary poll handlers, then processes a number of events that are waiting in the event queue, finally put the CPU to sleep when there are no pending events. The flowchart of main file is in Figure 5. Parts of the code can be deduced from the Tmote sky main file: Contiki-sky-main.c, whereas some code, such as the devices not being used, must be cut. Moreover, the initialization function of the energy meter ADE7753 should be included.

The ADE7753 driver is located in the subfolder platform/$PLATFORM/dev. The driver composes of initialization APIs and an interrupt function. The interrupt function is no more than some lines of SPI sending and reading sequence,

accompanied by occasional GPIO operation, such as reset. Following is an example of reading the SPI buffer data.

```
while ((UC1IFG & UCB1RXIFG) == 0)
…
a= UCB1RXBUF;
```

Although CC2420 driver is already included in the Contiki OS, updated CPU and changes of its connections to the CPU, require further modification of that driver. In fact, only two files need to be modified. One is cc2420-arch.c, which defines FIFOP as MCU external interrupt; the other is cc2420-arch-sfd.c, which defines SFD (Start of Frame Delimiter/digital mux output) as MCU timer input. The original CC2420 files are all in the /core directory, in order not to modify the kernel, the two modified files are put in the /platform folder.

4.4 Contki network stack initialization

The Contiki communication stack can be set to all kinds of different topologies. In this design, an IPv6 based structure is chosen, as is shown in figure 12.

The initialization of transport layer, such as UDP, ICMP, will be done in the application code. Here the main job is to define the uIP and its lower layer.

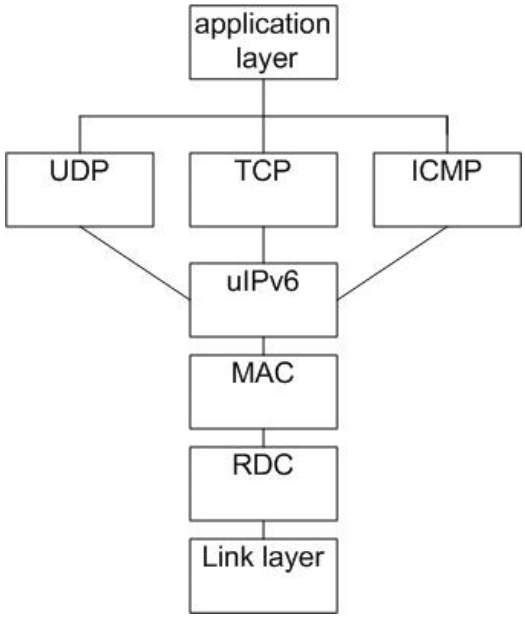

Figure 12: Configured Contiki network stack

The centre part is the uIPv6 layer, which implements 6LOWPAN and RPL. The uIP configuration can be done by simply setting up some flags in the Contiki-conf.h located in the target platform folder, like the following:

```
#define UIP_CONF_ROUTER                 1
#define UIP_CONF_IPV6_RPL               1
…
…
#define UIP_CONF_IPV6                   1
…
#define UIP_CONF_BUFFER_SIZE        140

#define SICSLOWPAN_CONF_COMPRESSION_IPV6        0
#define SICSLOWPAN_CONF_COMPRESSION_HC1         1
```

In order to reduce packet size, HC1 header compression for 6LOWPAN should be enabled. At the same time, ContikiRPL flags must be properly set to initialize the uIPv6 routing mechanism.

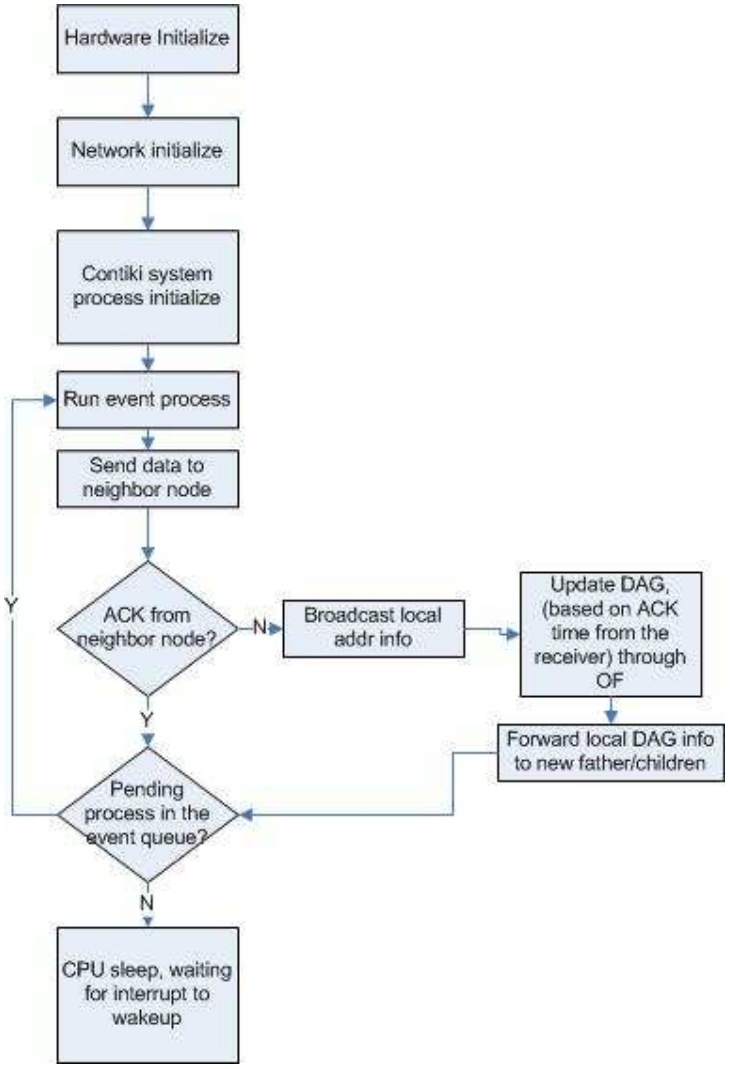

Figure 13: System mainloop(DAG maintenance)

For the lower level communication stack, contikimac is used to combine MAC and hardware driver of radio devices. ContikiMAC is a new radio duty cycling mechanism that uses a combination of link-layer and MAC-layer mechanisms to achieve very low power consumption: during idle listening the radio is switched off between 99.2% and 99.6% of the time. One unicast transmission typically adds between 0.07% and 1.2% additional radio time, depending on packet size.

ContikiMAC uses the standard IEEE 802.15.4 message format and adds no additional headers.

ContikiMAC is simple: it uses periodic two-shot channel sampling during idle listening to keep the radio on-time down. Transmissions are done with repeated transmissions until a link-layer ACK is received.

The following code can explain the top-down topology of Contiki network stack:

```
#define NETSTACK_CONF_RADIO      cc2420_driver
…
#define NETSTACK_RADIO NETSTACK_CONF_RADIO
…
#define NETSTACK_CONF_MAC         csma_driver //set MAC driver
#define NETSTACK_CONF_RDC         contikimac_driver//set RDC
#define NETSTACK_CONF_RADIO      cc2420_driver//set radio driver
…
cc2420_set_pan_addr(IEEE802154_PANID, shortaddr, longaddr); //set radio address
…
#if WITH_UIP6
   memcpy(&uip_lladdr.addr, node_mac, sizeof(uip_lladdr.addr)););//get uIPv6 addr from MAC
…
NETSTACK_RDC.init(); //combine MAC & link-layer RF(radio duty cycle mechanism)
NETSTACK_MAC.init(); //initialize MAC
…
```

After the above initializations, the system begins its infinite main loop, as in figure 13. Notice that although it is a loop, it is conditional, because it will run for only limited times in every certain time period-The duration of the time instant, as well as the duty cycle of when the communication is enabled, are all set by the initializations above. At each loop, the system will check the events in its process

queue, and do the events based on priority. Since RPL has been enabled, it must include some processes sending packet to its neighbors (DIO messages), in order to maintain its DAG. If no response is received after certain amounts of its repeated DIO, the kernel will assume that its RPL map has been changed. In such condition, a new RPL DAG building sequence will start over, beginning with its own bare address broadcasting event.

Accordingly, a separate interrupt routine for RPL neighbor discovery has to be enabled for some time in every user-defined time period. Like figure 14, for each received packet, the system firstly checks the sender's ID with its local DAG information. If it is a new sender, the DAG updating sequence will begin; otherwise, it will save or forward the packet based on its destination address information.

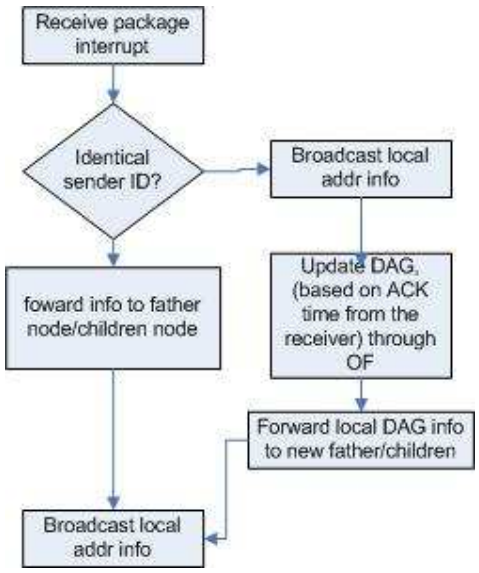

Figure 14: Receive interrupt routine(DAG updating)

5. APPLICATION DESIGN

5.1 Experiment

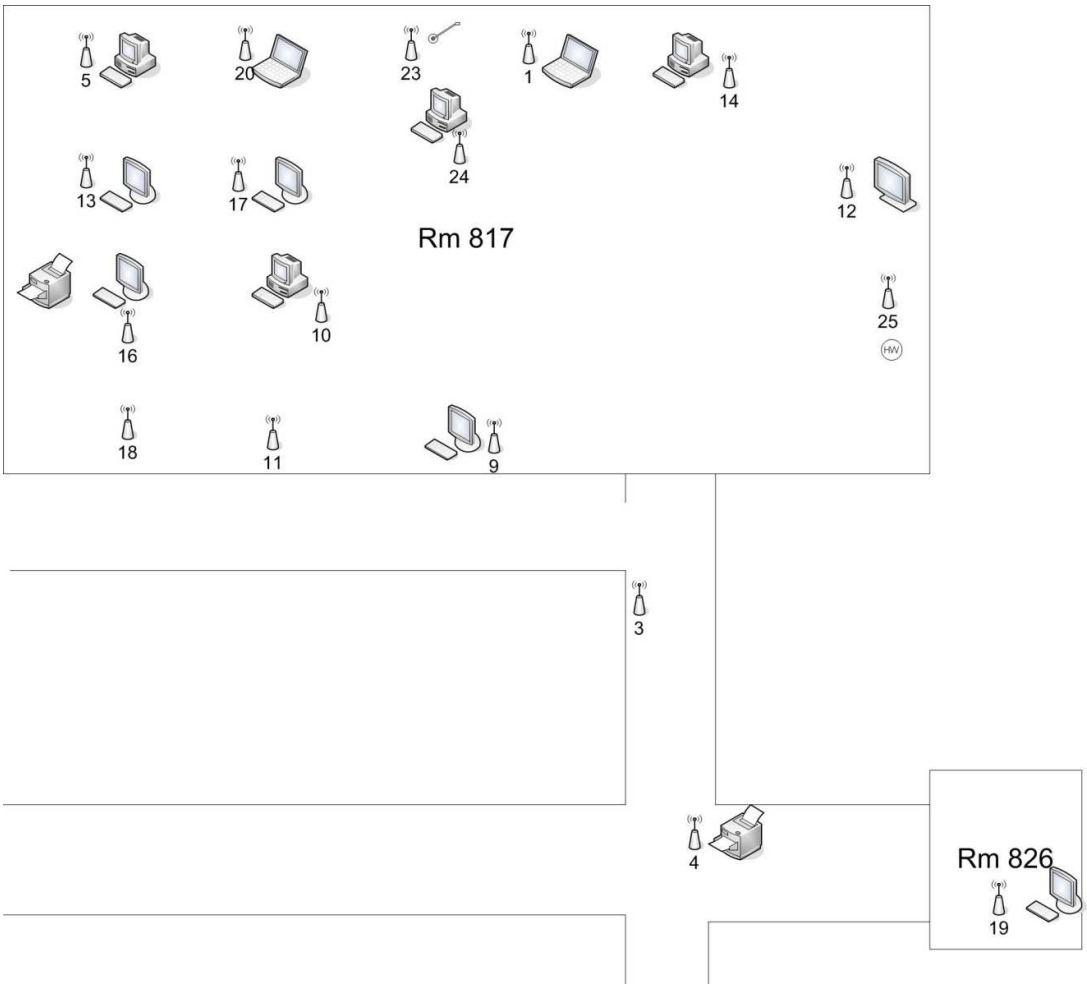

Figure 15: Experiment setup on EERC 8F

A network has been set up in the EERC 8F office, as in figure 15. In that network, there is a meter node near each appliance. Node 1 is used as gateway, with which a PC could monitor the energy network by reading its data through serial-port terminal. Node 23 locates on the AC plug for our soldering iron. Node 12 is for the

television, with node 25 for the plug used by an electrical kettle everyday. All the other nodes in room 817 are on the plugs for laptop, desktop computers and printers. Node 3 is used as intermediate node to forward data into other nodes, which locate in the aisle(node 4) or in other offices(node 25). The nodes in the room could prove the scalability of this network; whereas the node 3,4,19 can evaluate its multi-hop functionality.

5.2 Application software

Since the RPL DAG maintenance sequence is already covered by the kernel. Application designers can directly use the IPv6 APIs to build their network device. The latest release of Contiki has included an IPv6 example, which fully supports Tmote sky. The rpl-udp example contains an UDP server and an UDP client. both files have been successfully tested on the designed platform. Thus, now the building energy network could be built, by simply programming one node as the server, others as clients. On the PC side, by opening a serial port terminal to monitor the server node, the appliances in the network could be monitored.

Most initialization is already done by the system kernel and hardware driver, the application just set up default dag and global IPv6 header of the node. A process called “udp_server” is defined, waiting in the event queue, in case it is called by system main loop. It will set its default UDP header, which can be deduced from the lower network layer. Since its ultimate receiver should be the network gateway, it

combines its local UDP port with gateway's remote UDP port (the gateway should be its only destination). The initialization code of the meter's UDP process is defined like the following:

```
PROCESS_THREAD(udp_server_process, ev, data)
{
    rpl_dag_t *dag;
    rpl_set_root((uip_ip6addr_t *)dag_id);
    dag = rpl_get_dag(RPL_ANY_INSTANCE);
    uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
    rpl_set_prefix(dag, &ipaddr, 64);
  …
```

A loop other than the system mainloop, will run after that initialization. The application loop will yield to hardware events such as timer expiring or packet receiving. In the loop, every meter node is programmed to measure its metering data and put it into its data buffer. Because it is the system kernel that takes care of the receiving and forwarding data routine, as is explained in Table 3. Basically, that loop will not run until a tcp/ip event wakes it up. It should not be waked up by any background event. The time instant between two continuous wake-up events is set by the root node, using the etimer_set() handler. Before the timer expires, the meter node should be in background running mode, in which non-protocol transmission, instead of IPv6, is used maintain its DAG information and monitor its map change. This mechanism managed to guarantee minimum power consumption. Following code is part of the application main loop:

```
  …
    while(1) {
      PROCESS_YIELD();
      if(ev == tcpip_event) {
```

```
            …
            meter_data=meter_spi();
            sprintf(buf, "distance=%d", meter_data);
        //
            uip_ipaddr_copy(&server_conn->ripaddr, &UIP_IP_BUF->srcipaddr);
            uip_udp_packet_send(server_conn, buf, sizeof(buf));//send data to root node
            uip_create_unspecified(&server_conn->ripaddr);
            …
        }
          else
                    rpl_repair_dag(rpl_get_dag(RPL_ANY_INSTANCE));
    }
```

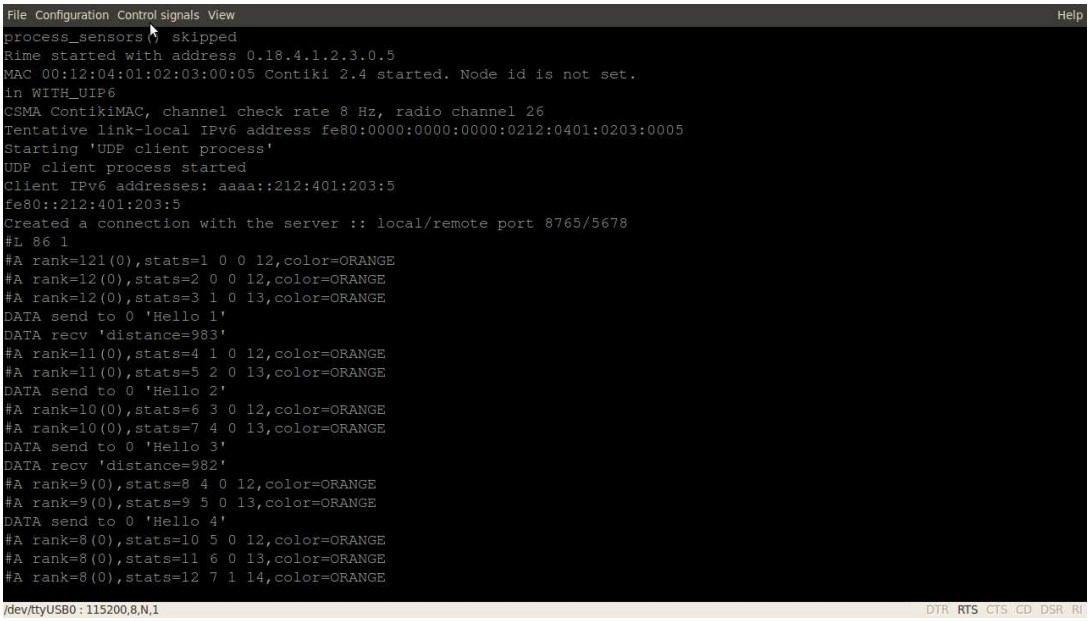

Figure 16: RPL-UDP starting up process

Figure 16 is the serial port terminal reading of the gateway node. After its startup initialization, it first discovers node 23, whose lowest 8bit of address is 86. After its DAG stabilized, it calls the node 23, and receives its the sensor reading.

By the time of writing this paper, the team has not yet populated the meter board they designed. Instead, a UART ultrasonic sensor was being used. The sensor's

data represents the distance between obstacle and itself, to simulate the metering network application, as is shown in figure 17.

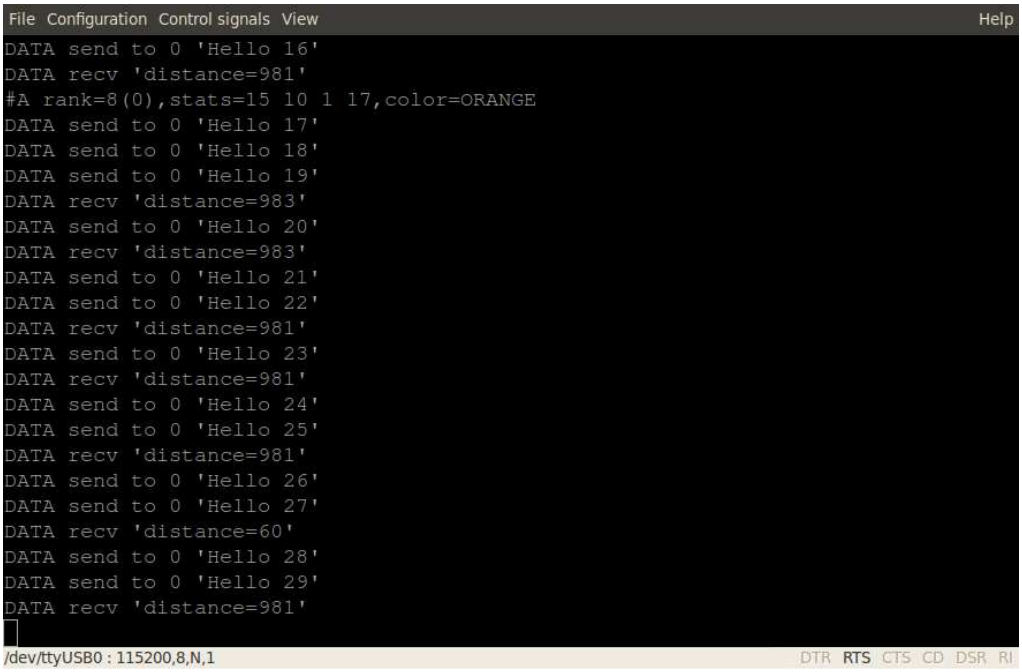

Figure 17: Transferring monitored data through RPL

The design for gateway node is slightly different. Instead of passively waiting for the host to call it ID, like the meter node does, it will broadcast itself, gathering its DAG information, and call all its children and children's children actively.

The Contiki kernel has the capability to "detect" neighbor host automatically, and dynamically store it. Accordingly, the root node can just read the host information (address, node, data packet) through the handler udp_new(), and copy the information to a structure variable, using the function udp_bind(). Furthermore, based on the situation of the network, various send intervals could be set by the gateway. In the real application of energy monitoring, such interval should usually be set from 15

seconds to several minutes. However, for purpose of test, the interval was reduced down to 5 seconds during the time.

Following is part of the application code for the root node:

```
/* new connection with remote host */
client_conn = udp_new(NULL, HTONS(UDP_SERVER_PORT), NULL);
udp_bind(client_conn, HTONS(UDP_CLIENT_PORT));
etimer_set(&periodic, SEND_INTERVAL);
while(1) {
  PROCESS_YIELD();
  if(ev == tcpip_event) {
    tcpip_handler();
  }
  …
  if(etimer_expired(&periodic)) {
    etimer_reset(&periodic);
    ctimer_set(&backoff_timer, SEND_TIME, send_packet, NULL);
  }
}
```

After loading the application code into all nodes in the building energy network setup in figure 15, the serial-port terminal log of the gateway node was recorded, and the network's RPL DAG could then be depicted, as in figure 18. After 4 minutes, data was finally received from the furthest node: node 19. After that, the DAG has been built up, and achieved more than 60% successful transmissions thereafter.

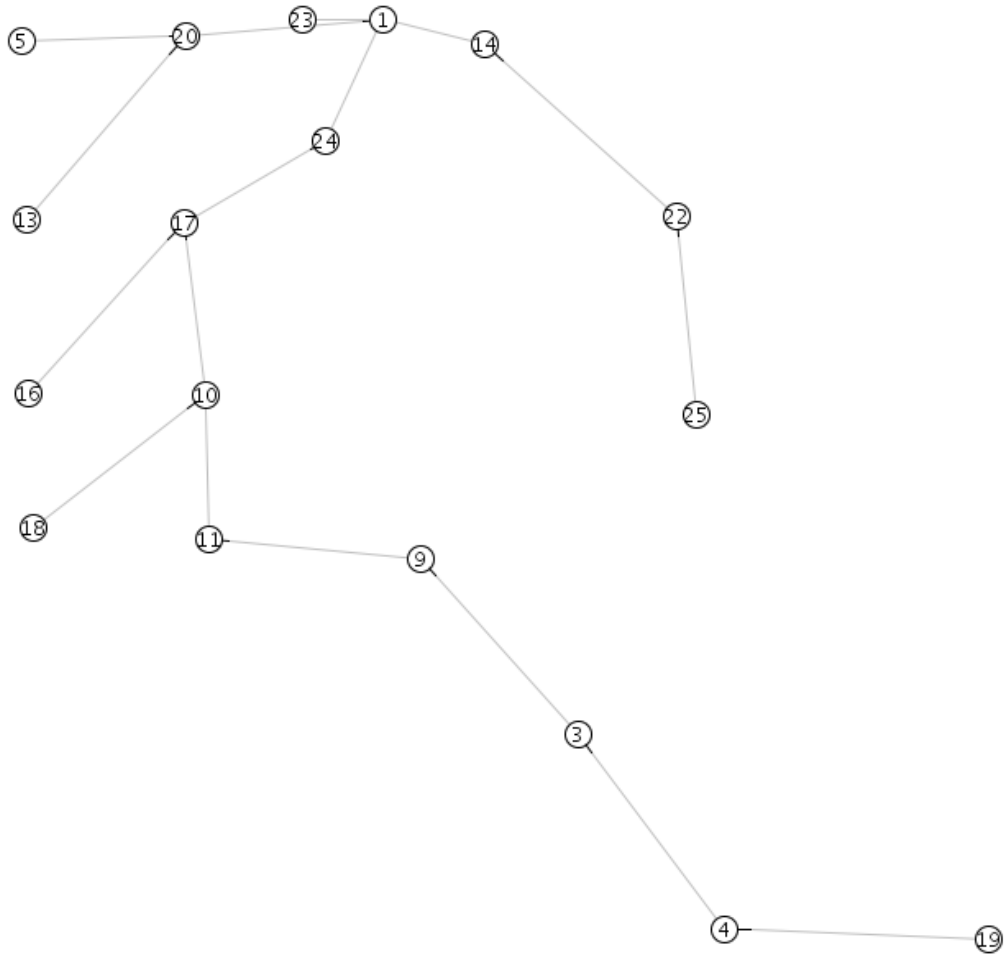

Figure 18: Routing DAG of 8F energy network

The above method is simple. All the information is gathered by the root node, and then transferred to the PC. On the PC side, it is still a serial port(or USB) application. In order to exchange information with service providers, certain applications like slip(serial line IP) are needed to “transform” the serial port into an IP port. The current solution is to program meter node with a webserver firmware, /example/$APPLICATION/websense.c. It is a lightweight webserver with basic HTML and CGI functions, which displays the most reading of the energy meter. For the gateway node, the webpage is designed only to display the neighbors’ IPv6

address, which is automatically discovered, as is in figure 19. Currently, considering that energy measurement is not necessary in the root node, the already fully supported Tmote Sky could be utilized instead. The demo program is rpl-border-router, with which a python application has been developed to support gateway functionality. Through the gateway, the user can browse all meters in any kind of HTML webserver. Simply typing its IPv6 address, which has been shown on the router page in figure19, they can see the energy reading on its webpage of figure 20.

Following is part of the websense program for the meter node:

```
#define SEND_STRING(s, str) PSOCK_SEND(s, str, (unsigned int)strlen(str))
…
#define ADD(...) do {                                                    \
      blen += snprintf(&buf[blen], sizeof(buf) - blen, __VA_ARGS__);          \
   } while(0)
…
PT_THREAD(send_values(struct httpd_state *s))
{
   PSOCK_BEGIN(&s->sout);

   SEND_STRING(&s->sout, TOP);

   if(strncmp(s->filename, "/index", 6) == 0 ||
        s->filename[1] == '\0') {
      /* Default page: show latest sensor values as text (does not
           require Internet connection to Google for charts). */
      blen = 0;
      ADD("<h1>Current readings</h1>\n"
            "energy: %u<br>"
            get_meter_reading());
    SEND_STRING(&s->sout, buf);
    …
  }
```

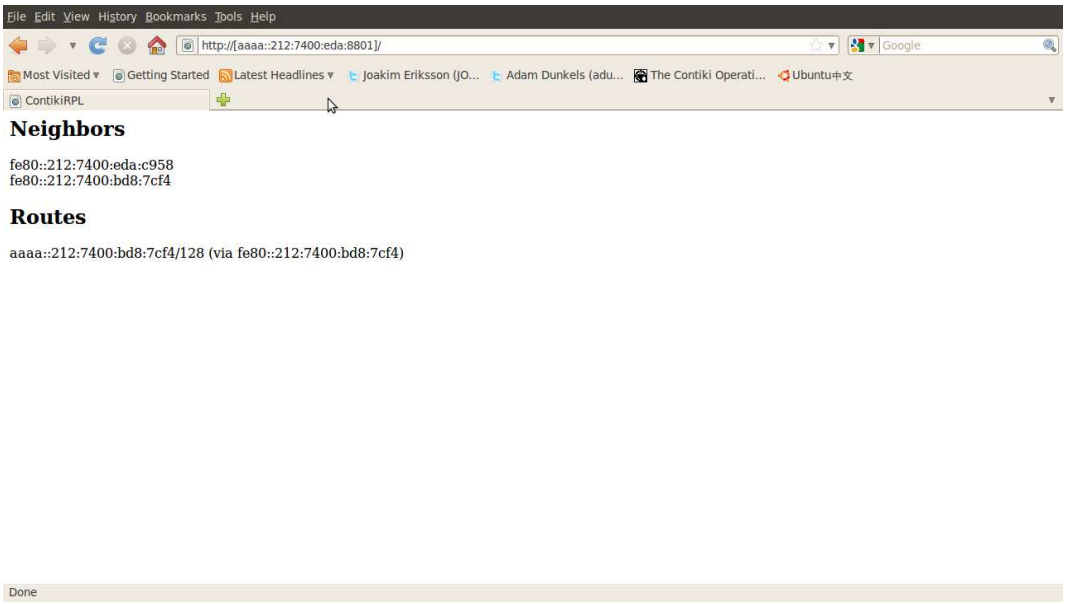

Figure 19: Webpage of gateway node

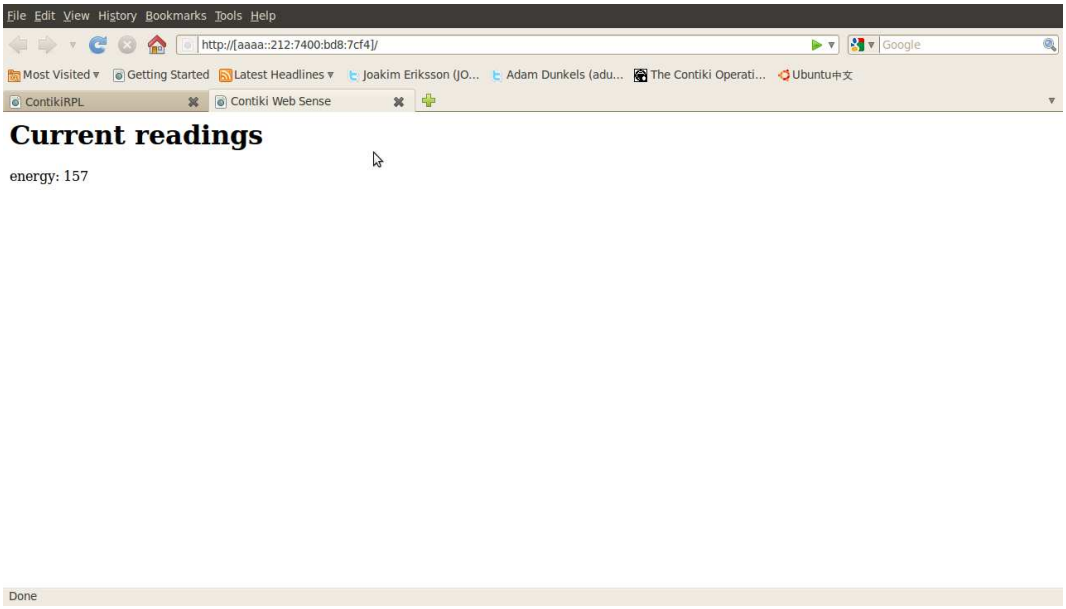

Figure 20: Webpage of meter node

Table 3 is a summary of our application software design on top of Contiki OS.

On the left are the Contiki system functions that have been utilizing. On the right are

customize source code. As is shown above, the Contiki has greatly helped develop such an IP based metering network.

Table 3: User developed codes in the Contiki file system

Jobs done by the Contiki kernel	Jobs done by application designer
OS common API Part of CPU specific code 6LOWPAN tcp/ip stack RPL automatic routing	rest part of CPU specific code: /platform/msp430.c peripheral specific code: /platform/$TARGET application code for meter: /example/meter_node.c compact web page: /example/web_sense.c

6. FUTURE WORK

So far, the design of a large energy metering network in the building has been briefly discussed. However, some works still needs to be finished.

First, a complete solution for the meter node has been provided, but there is only a temporary solution mentioned above for the root node. The next step is to design the gateway device. Currently, the team is evaluating Contiki OS on a stm32 ARM platform, which is a powerful RISC command embedded CPU. A more powerful antenna will be deployed on the 802.15.4 radio. And GSM, Wi-Fi or Ethernet module is also to be connected to the CPU, in order to transfer data to the grid's WAN network.

Secondly, more experimental data is needed to test the reliability of the system, such as the network QoS evaluation; or the efficiency, such as the idle state power consumption.

Moreover, the team is updating the radio chip itself. Actually, the module EasyBee may be replaced to another OEM 802.15.4 radio module: AMBZ300-EM, which is developed by amber wireless. Its radio CC2520, which is a new generation of CC2420, has more functionalities such as packet sniffling and will be able to improve the network quality. Currently the device can be connected to MSP430F2618, through a Z-stack firmware provided by Texas Instrument [20]. The Contiki developers have not yet released CC2520 driver, but considering it could be

configured as identical peripheral interface with CC2420, it should not be a problem [21].

Finally is development of the application on the server, given that this system supposes to be part of the smart grid's AMI. The application interface to the utility company, like the on demand data profiling, energy curve prediction, dynamic pricing and peak hour reminder, etc, could be developed on the server PC or the embedded meter concentrator.

REFERENCES

[1] H. Labiod and M. Badra, “New Technologies, Mobility and Security”, 2007, pp 597-606, An Advanced Metering Infrastructure for Future Energy Networks.

[2] The Smart Grid Interoperability Panel – Cyber Security Working Group;, “Smart Grid Cyber Security Strategy and Requirements”, DRAFT NISTIR 7628: pp. 44-48, Feb 2010

[3] Khalifa, T; Naik, K; Nayak, A; , "A Survey of Communication Protocols for Automatic Meter Reading Applications," *Communications Surveys & Tutorials, IEEE* , vol.PP, no.99, pp.1-15, Oct.2010.

[4] Hui, J.W.; Culler, D.E.; , "IPv6 in Low-Power Wireless Networks," *Proceedings of the IEEE* , vol.98, no.11, pp.1865-1878, Nov. 2010.

[5] IETF Work group, draft-ietf-6lowpan-hc-13.

[6] Hui, J.W.; Culler, D.E.; , "Extending IP to Low-Power, Wireless Personal Area Networks," *Internet Computing, IEEE* , vol.12, no.4, pp.37-45, July-Aug. 2008

[7] Jean-Philippe Vasseur, Adam Dunkels, “Interconnecting Smart Objects with IP: The Next Internet”, pp231-250, Elsevier Inc, 2010.

[8] Di Wang; Zhifeng Tao; Jinyun Zhang; Abouzeid, A.A.; , "RPL Based Routing for Advanced Metering Infrastructure in Smart Grid," *Communications Workshops (ICC), 2010 IEEE International Conference on* , vol., no., pp.1-6, 23-27 May 2010

[9] Jean-Philippe Vasseur, Adam Dunkels, “Interconnecting Smart Objects with IP: The Next Internet”, pp251-288, Elsevier Inc, 2010.

[10] Xiaofan Jiang; Dawson-Haggerty, S.; Dutta, P.; Culler, D.; , "Design and implementation of a high-fidelity AC metering network," *Information Processing in Sensor Networks, 2009. IPSN 2009. International Conference on*, vol., no., pp.253-264, 13-16 April 2009.

[11] “SLAS541F: MSP430F241x, MSP430F261x MIXED SIGNAL MICROCONTROLLER”, Texas Instruments, June 2007

[12] FlexiPanel Ltd, “EasyBee: 2.4GHz ZigBee ready IEEE 802.15.4 RF transceiver”, www.FlexiPanel.com, Mar,2008

[13] Analog Devices, Inc, “Single-Phase Multifunction Metering IC with di/dt Sensor Interface: ADE7753”, www.analog.com,2003–2009

[14] http://www.sics.se/~adam/contiki/docs/

[15] Nicolas Tsiftes, Joakim Eriksson, and Adam Dunkels, “Poster Abstract: Low-Power Wireless IPv6 Routing with ContikiRPL”, IPSN’10, April 12–16, 2010

[16] “SLAU319A: MSP430 Programming Via the Bootstrap Loader User's Guide”, Texas Instruments, July 2010

[17] http://mspgcc.sourceforge.net/

[18] http://www.gnu.org/software/make/manual/make.html

[19] Adam Dunkels, Fredrik Osterlind, Zhitao He;, “An Adaptive Communication Architecture for Wireless Sensor Networks”, SenSys’07, November 6–9, 2007.

[20] Amber wireless, “AMBZ300-EM datasheet”, http://www.amber-wireless.de , 2010

[21] “SWRS068: CC2520 DATASHEET”, Texas Instruments, December 2007.