Special Issue Article | ICADET 2017[b]

Turkish Journal of Electromechanics & Energy

ISSN-2547-975X

Contiki OS Usage in Wireless Sensor Networks (WSNs)

Erhan Sesli[1*], Gökçe Hacıoğlu[2]

[1] Sürmene Abdullah Kanca Vocational School of Higher Education, Karadeniz Technical University, Trabzon, Turkey
[2] Department of Electrical and Electronics Engineering, Karadeniz Technical University, Trabzon, Turkey

Received: 26 October 2017; Revised: 2 December 2017; Accepted: 15 December 2017; Published: 30 December 2017
Turk J Electrom Energ Vol.: 2 No: 2 Page: 1-6 (2017)
SLOI: http://www.sloi.org/
*Correspondence E-mail: erhansesli@ktu.edu.tr

ABSTRACT Wireless Sensor Networks (WSNs) have very wide range of applications from health to agriculture, from military technologies to observing of volcano activities. Developers and engineers frequently need to simulate WSN to ensure developed applications work successfully and to analyze effects of various configurations of wireless nodes. Simulating the designed scenario and embedding the designed algorithms into the wireless modules effectively are the important factors for the developers and engineers in this field. In this study, Contiki Operating System is proposed as a convenient solution for developers and engineers. Contiki is an open source, Linux based operation system, and developed for Internet of Things (IoT) devices. In this paper, primarily, Contiki OS usage and advantages in WSN were explained, then Contiki OS usage over a sample scenario was given and finally advantages of Contiki OS over other popular operation systems such as Tiny OS and Lite OS were examined. Background information on WSN and Contiki OS to build an example scenario for beginners were provided.

Keywords: Wireless Sensor Networks (WSNs), Contiki OS, Simulation

Cite this article: E. Sesli, G. Hacıoğlu, Contiki OS Usage in Wireless Sensor Networks (WSNs) Turkish Journal of Electromechanics & Energy 2(2) 1-6 (2017)

1. INTRODUCTION

The identification layer of the modern information processing system is completely related to the sensing of six physical phenomena in nature such as; mechanical, electrical, magnetic, thermal, radiation, and chemical [1]. Importance of the sensing leads to develop new approaches and systems in many fields. Wireless Sensor Networks (WSNs) are regarded as a good example to new developments and technologies in this fields.

In recent years, WSNs find applications from health to agriculture, military technologies to observing of volcano activities which have vital importance at many points [2, 3]. Many of the mentioned applications have to operate in environmental conditions that are stand-alone and energy-limited. Therefore, localization algorithms [4, 5] and novel energy-efficient clustering algorithms [6, 7] in WSNs have become popular field of studies. Undoubtedly, factors such as system simulation of WSN applications, making various configurations of nodes and embedding the algorithms into wireless nodes effectively are vital for developers. In the application development phase of the WSNs, many interfaces and programs can assist to the developers. One of the appropriate solution is the usage of Contiki OS. Contiki OS is an open-source operating system which is Linux based and developed for Internet of Things (IoT) devices [8]. It has also powerful tools for building complicated wireless communication systems. Contiki OS has been especially developed for low-powered WSN apps. In other words, it has been developed for WSN applications that are able to work with AA type batteries for years. Another specific feature of the Contiki OS is Cooja Network Simulator which provides simulation environment for developed algorithms. Contiki OS, which has an integrated structure through these features, presents researcher based solutions for WSN apps that may be developed.

In the next sections of the current paper, background information is given about WSNs and Contiki OS, then comparative assessments are provided between other alternatives, finally advantages of usage of Contiki OS are examined.

[b]*Initial version of this paper was selected from the proceedings of International Conference on Advanced Engineering Technologies (ICADET 2017) which was held in September 21-23, 2017, in Bayburt, TURKEY; and was subjected to peer-review process prior to its publication.*

2. WIRELESS SENSOR NETWORKS (WSNs)

Main purpose of the WSNs is to sense variable physical phenomena in a certain environment, then to transmit the information through wireless network [2]. Basic working mechanism of a WSN node is shown in Figure 1.

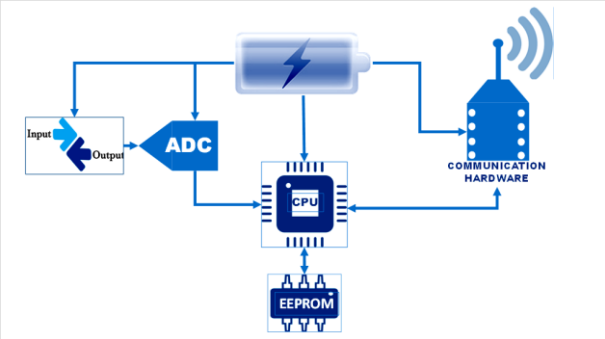

Fig. 1. Basic working mechanism of a WSN node.

A typical WSN node consists of a microprocessor, a power unit and some blocks like I/O, analog to digital converter (ADC), communication and memory block. WSNs generally operate in environmental conditions that must be stand-alone and energy-limited. Hence, power is often provided by energy harvesting methods such as solar panels, piezoelectric equipment etc. Power that is obtained by energy harvesting methods might be used for recharging the batteries.

In WSN, information is sensed by a node named as end-device, that is often at the far point in network. End-devices are manufactured as Reduced-Functioned Devices (RFDs) and have simple tasks such as sensing analog information and transmitting it. End-devices transmit information to the more authorized node in network named as Router. Routers are often manufactured as Full-Functioned Devices (FFDs). FFDs have the ability to communicate with other RFDs and FFDs. Router which is a FFD device can get information from the adjacent node and transmit it to another router or directly to the Coordinator. Coordinator which is the most authorized node in network are manufactured as FFD. Coordinator or in other words sink can be thought as the gateway of the network. General architecture of a typical WSN is shown through a scenario in Figure 2. [9].

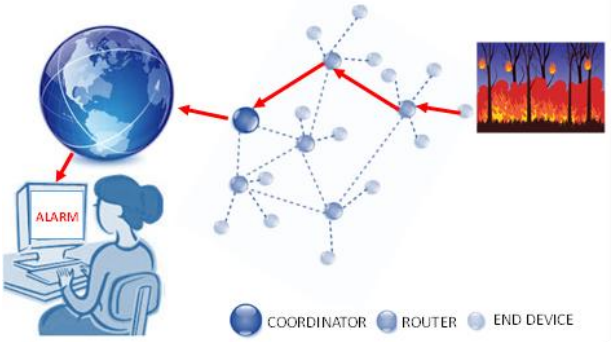

Fig. 2. General architecture of the WSNs

In the scenario, fire incidence information that occurred in a forest is sent to the end user through WSN and the internet. Primarily, fire is sensed by a sensor which is mounted to the end-device. Then, end-device transmits the information to the neighbor router. Next, information is transmitted to the coordinator by routers. After that, information which is received by coordinator is transmitted to the internet and finally the receiver gets the vital information through the internet.

Applied standards for WSNs are determined via IEEE 802.15.4 [10]. There are various available manufacturer solutions to develop applications in WSNs. Nevertheless, some of the solutions are costly. Therefore, searching cost-effective or cost-free solutions become important. On the other hand, cost-effective or cost-free solutions do not have the identical features, they have advantages/disadvantages over each other.

3. CONTIKI OPERATING SYSTEM AND ITS STRUCTURE

Contiki OS is an operating system which is developed by Dunkel et al. [11]. Contiki OS, which is C programming language based and open source, has been developed for lightweight, flexible and low-powered wireless sensor networks.

Working environments of the WSNs are often energy-limited as mentioned. This is one of the most important constraint for WSNs. Likewise, tiny and simple designs of the nodes are the other constraints. For this reason, WSNs should have some important hardware and software features to cope with these constraints. Contiki OS is one of the convenient solutions to cope with mentioned constraints thanks to its flexibility and support of lightweight and low-powered networks [11].

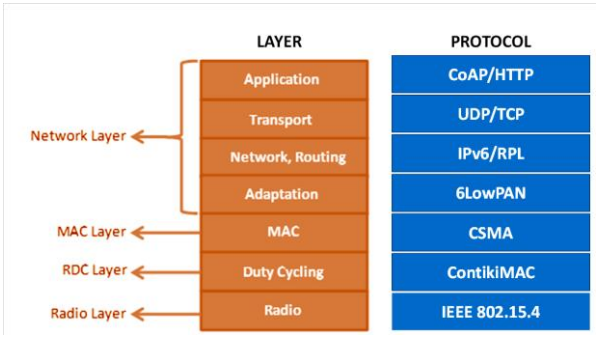

Fig. 3. Contiki network stack

A standard Contiki configuration for a microcontroller is 2kB RAM and 40 kB ROM. Besides that, Contiki can provide communication over IPv4, IPv6 and Rime Network Stack [12]. Contiki Network Stack shown in Figure 3. gives more details for its structure.

Contiki directory in OS, also provides access to system source codes, sample application codes, practical applications, and driver codes for many node types, specific microcontroller files and important tools like Cooja. Thus, besides developing and simulating new projects, Contiki provides the opportunity for developers to use existing samples directly or modify them. With these features, researchers and developers would have effective development environment.

3.1. Cooja Simulator Environment

Cooja is a WSN simulator which enables simulating the developed applications. Thus, developers can make their own applications through these codes, drivers and tools.

In Figure 4, a screenshot of Cooja environment is provided. While node communication information is flowing in colored screen which is in the middle window, node communication directions and communication ranges of the nodes can be seen on the left window.

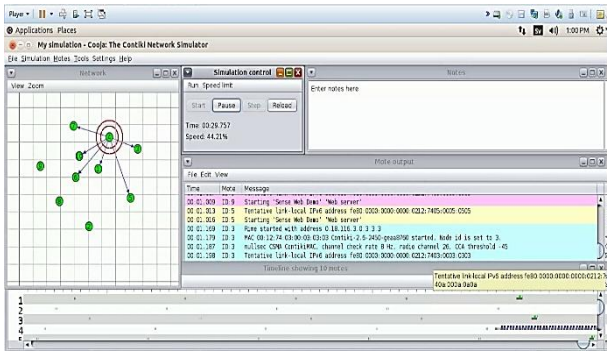

Fig. 4. A screenshot of Cooja environment

3.2. An Example in Contiki OS

Explaining the usage of Contiki OS within an example scenario will be useful for the new developers.

In the scenario three wireless nodes, one of the nodes is receiver node and the others are the energy-limited transmitter nodes, are included. Hence, transmitter nodes would sleep with certain periods for energy saving.

For this kind of example, usage of the Rime Network Stack is useful as it has a structure which makes the wireless network communication easy. If rime directory which is in Contiki OS directory is considered, codes for nodes can be built by modifying the *anonymous broadcast (abc)* sample.

For communication of receiver and transmitter nodes in given example, 25th channel (2.475 GHz) which is defined in IEEE 802.15.4 was preferred. Transmitter and receiver nodes should be configured separately. For receiver node, communication channel should be configured as 25th channel and RDC layer in Contiki Network Protocol Stack should be configured as *nullrdc*. Required configurations can be carried out by modifying the *project-conf.h* file which is in rime directory. Modified *project-conf.h* file for receiver node is given in Figure 5.

```
 * \file
 * Project specific configuration defines for th
 */
#ifndef PROJECT_CONF_H_
#define PROJECT_CONF_H_
#define NETSTACK_RADIO_RADIO_PARAM_CHANNEL 25
#define NETSTACK_CONF_RDC nullrdc_driver
#endif /* PROJECT_CONF_H_ */
```

Fig. 5. Receiver node configuration

Built code for receiver node is shown in Figure 6. Code lines that were defined by numbers include general and important details for building a project in Contiki. Code line numbers are explained as below;

1- PROCESS is one of the most important component for the Contiki. Processes are defined via PROCESS macros.

2- AUTO_PROCESS starts PROCESS automatically via the given arguments.

3- abc_recv is a function which is used to display received messages. abc_recv( ) is designated as a callback function and it is automatically called when a message is received.

4- PROCESS_THREAD is used for proto thread of a process.

5- PROCESS_EXIT HANDLER specifies an action when a process exits. In this example, quitting receiving mode is an action.

6- Beginning of the process is declared by PROCESS_BEGIN macro. It must always be in the PROCESS_THREAD definition.

7- PROCESS_END macro is used for quitting from the process.

```
#include "contiki.h"
#include "net/rime/rime.h"
#include "random.h"
#include "dev/button-sensor.h"
#include "dev/leds.h"
#include <stdio.h>
/*---------------------------------------------------------------------------*/
PROCESS(example_abc_process, "ABC example"); 1
AUTOSTART_PROCESSES(&example_abc_process);
/*---------------------------------------------------------------------------*/
static void
abc_recv(struct abc_conn *c) 3
{
  printf("message received'%s'\n", (char *)packetbuf_dataptr());
}
static const struct abc_callbacks abc_call = {abc_recv};
static struct abc_conn abc;
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(example_abc_process, ev, data) 4
{
  static struct etimer et;

  PROCESS_EXITHANDLER(abc_close(&abc);) 5
  PROCESS_BEGIN(); 6
  abc_open(&abc, 128, &abc_call);
  PROCESS_END(); 7
}
```

Fig. 6. Code for receiver node

Transmitters are energy-limited nodes. They should sleep in certain intervals for energy-saving. For this reason, channel check rate of the transmitter nodes was determined as 8 times in a second. Configuration of the channel check rate provides the sleeping in certain intervals. Besides, since there are two transmitter, there may be two signal in channel at the same time. Accordingly, Radio Duty Cycle (RDC) layer in Contiki Network Protocol Stack should be configured as *contikimac*. Thus, transmitter initially listens to channel; if channel is idle then it transmits its own signal. Required configurations for these were made by changing *project-conf.h* file as seen in Figure 7. Built codes for transmitter nodes are seen in Figure 8.

```
 * \addtogroup cc2538-examples
 * @{
 *
 * \file
 * Project specific configuration defines for the
 */
#ifndef PROJECT_CONF_H_
#define PROJECT_CONF_H_
#define NETSTACK_RADIO_RADIO_PARAM_CHANNEL 25
#define NETSTACK_CONF_RDC contikimac_driver
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 8
#endif /* PROJECT_CONF_H_ */
```

Fig. 7. Transmitter node configuration

```
#include "contiki.h"
#include "net/rime/rime.h"
#include "random.h"
#include "dev/button-sensor.h"
#include "dev/leds.h"
#include <stdio.h>
/*---------------------------------------------------------------------------
PROCESS(example_abc_process, "ABC example");
AUTOSTART_PROCESSES(&example_abc_process);
/*---------------------------------------------------------------------------
static void
abc_recv(struct abc_conn *c)
{
}
static const struct abc_callbacks abc_call = {abc_recv};
static struct abc_conn abc;
/*---------------------------------------------------------------------------
PROCESS_THREAD(example_abc_process, ev, data)
{
  static struct etimer et;

  PROCESS_EXITHANDLER(abc_close(&abc);)
  PROCESS_BEGIN();
  abc_open(&abc, 128, &abc_call);

  while(1) {

    etimer_set(&et, CLOCK_SECOND * 2+ random_rand() % (CLOCK_SECOND * 2));

    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

    packetbuf_copyfrom("Here I am", 13);

    abc_send(&abc);

    printf("abc message sent\n");
  }

  PROCESS_END();
}
```

Fig. 8. Codes for transmitter node

A random time among 2-4 seconds was determined by *etimer_set (struct etimer*, clock_time_t)* function. Thus, event - timer was set up that time interval.

Process waits until an event occurs via *PROCESS_WAIT_EVENT_UNTIL (etimer_expired(&et))* function. When event-timer expires, event occurs and if channel is idle, *"Here I am"* information is sent as a broadcast message. Then, "*message sent*" information is written on information screen.

Written codes for nodes in scenario were compiled and scenario was built in Cooja environment. Screenshot of the scenario that was carried out in Cooja environment is provided in Figure 9.

In Figure 9 instantaneously, *sender ID2* transmits a broadcast message. While *receiver ID1* receives *"Here I am"* information, *sender ID3* waits until channel is idle. When channel goes idle, *sender ID3* transmits its own broadcast message.

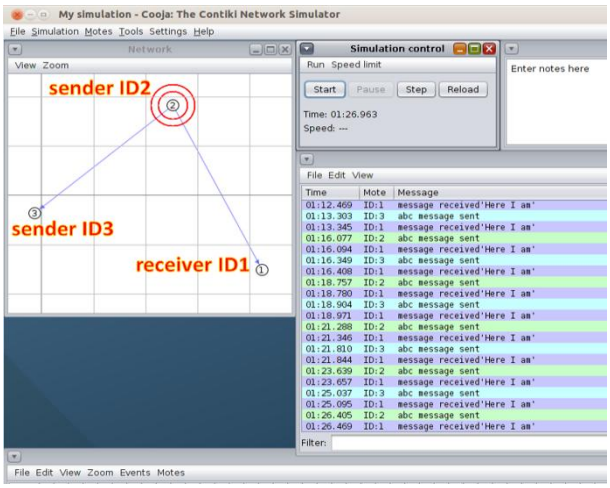

Fig. 9. Simulation in Cooja environment

In this section, general issues such as structure of the Contiki and the usage of the Contiki have been explained. In next section, other alternative WSN operating systems are explained and differences between them are examined.

4. COMPARISON OF CONTIKI OS WITH OTHER ALTERNATIVES

Other operating systems are also available for the WSNs. Among these, it can be said that Tiny OS and Lite OS are other most popular operating systems.

4.1. Tiny OS

TinyOS is a flexible and tiny operating system which consists of reusable components. Commands, events, and tasks are three computational abstractions of the components. Commands and events are used for inter-component communication. Tasks are as used to express intra-component concurrency.

Typical TinyOS configuration for a microcontroller is 16 kB ROM and 400 bytes of RAM. TinyOS has component-based programming model provided by NesC programming language. NesC language which supports features such as extensive cross-component optimizations and compile-time race detection is used in TinyOS [13].

4.2. LiteOS

LiteOS which is presented by Huawei is an open-source, interactive and Unix-like operating system for IoT devices. LiteOS is a tiny OS like others. It has 10 kB memory. As the architecture, it has three basic component as LiteC compiler, OpenSC (Open Sensor Classes) and the LiteOS runtime environment. LiteC compiler is used for compiling codes from C++ to machine language. OpenSC provides an API library to ease software development. LiteOS runtime environment presents process scheduling and resource allocation.

LiteOS has the abilities such as supporting zero configuration, auto-discovery, and auto-networking. Detailed information about LiteOS can be obtained in references [14, 15].

4.3. Comparison of Three Popular Operating Systems

Comparison among these three operating systems is shown in Table 1 [16].

When the Table 1 is examined, the following details can be observed; all three operating systems are open-source, and users can exploit them without any cost. While Contiki OS and LiteOS have dynamic system, TinyOS has the static system. In other words, Contiki OS and LiteOS are good solutions for the environment which is dynamically variable. In addition, they have flexible structure for the developers. While Contiki OS and LiteOS have the modular system, TinyOS has monolithic system. With this aspect, Contiki OS and LiteOS are more convenient for personal network applications that need to be modified frequently. From the point of network support view, Contiki OS has IPv4, IPv6 and Rime network stacks. With this features, both it is provided to communicate ability over internet for Contiki OS and presented a lightweight network stack via Rime for low-powered wireless networks. Contiki OS makes difference with its own Protothread mechanism which is a lightweight thread mechanism. Lastly, all three operating systems have their own network simulators as compared in Table 1. But Contiki OS has three different alternatives which make itself popular.

Table 1. Comparison table of the operating systems

Comparison Criterion	Contiki OS 3.0	TinyOS 2.0	Huawei LiteOS
Source model	Open source	Open source	Open source
System(Dynamic/Static)	Dynamic	Static	Dynamic
System(Monolithic/Modular)	Modular	Monolithic	Modular
Networking support	IPv4, IPv6, Rime	Active message	File-Assisted
Programming language	C	NesC	LiteC++
Multithreading support	Yes (have Protothread mechanism)	Partial (through Tiny Threads)	Yes
Simulator	Cooja, MSPSim, Netsim	TOSSIM, Power Tossim	Through AVRORA

4. CONCLUSION

In this study, primarily Contiki OS which is a lightweight, open source operating system developed for WSN application was reviewed. Then, an example scenario through Cooja in Contiki OS was explained step by step and finally a comparison with other popular operating systems such as TinyOS and LiteOS was carried out

If we scrutinize Contiki OS, it is obviously seen that it has powerful tools for building complicated wireless communication systems. Especially, Rime Network Stack is very important as it presents a lightweight network stack which is very convenient for low-powered WSN's. In addition, Protothread mechanism is one of important factors which makes difference. Likewise, flexible structure of Contiki OS and ability to use in many WSN platforms like cc2538, skymote, MicaZ, Zolertia Z1 etc. increase its preferability. This study aimed to provide background information on WSN and Contiki OS and to build an example scenario for beginners.

Acknowledgement

This study was supported by Karadeniz Technical University Scientific Research Projects Coordination Unit under Grant No: FDK-2016-5410.

References

[1] G.C.M. Meijer, Smart Sensor Systems, Wiley, Wiltshire, 55-57, (2008).

[2] D. Puccinelli, M. Haenggi, Wireless sensor networks: applications and challenges of ubiquitous sensing, IEEE Circuits and Systems Magazine, 5(3), 19-31, (2005).

[3] Y. Karan, N. As, Electromagnetic radiation measurement of a high gain wireless network adapter. Turkish Journal of Electromechanics and Energy, 1(2), 17-23, (2016).

[4] A. Pal, Localization algorithms in wireless sensor networks: Current approaches and future challenges, Network Protocols and Algorithms, 2(1), 45-73, (2010).

[5] G. Han, H. Xu, T.Q. Doung, J. Jiang, T. Hara, Localization algorithm for wireless sensor networks: A survey, Telecommunication System, 52(4), 2419-2436, (2013).

[6] M. Ye, C. Li, G. Chen, J. Wu, EECS: An energy efficient clustering scheme in wireless sensor networks, In Performance, Computing, and Communications Conference, (IPCCC 2005) 24th IEEE International, Arizona, USA, 535-540, April (2005).

[7] G. Hacioglu, V.F.A. Kand, E. Sesli, Multi objective clustering for wireless sensor networks, Expert Systems with Applications, 59, 86-100, (2016).

[8] Contiki: The open source operating system for the internet of things, 2017, http://www.contiki-os.org/

[9] K. Akkaya, M. Younis, A survey on routing protocols for wireless sensor networks. Ad hoc Networks, 3(3), 325-349, (2005).

[10] IEEE Standards 802.15.4, (2017), http://www.iith.ac.in/~tbr/teaching/docs/802.15.4-2003.pdf

[11] A. Dunkels, B. Gronvall, T. Voigt, Contiki-a lightweight and flexible operating system for tiny networked sensors, 29th Annual IEEE International Conference on Local Computer Networks, Florida, USA, 455-462, Nov. (2004).

[12] Karadeniz Technical University, Engineering Faculty, Department of Computer Engineering, Computer Networks Laboratory, (2017), http://www.ktu.edu.tr/dosyalar/bilgisayarce12e.pdf

[13] L. Philip et al., TinyOS: An operating system for sensor networks, Ambient Intelligence, Springer, 35, 115-148, (2005).

[14] Q. Cao, T. Abdelzaher, J. Stankovic, T. He, The LiteOS operating system: Towards unix-like abstractions for wireless sensor networks, In Information Processing in Sensor Networks, IPSN'08. International Conference on IEEE, St. Louis, Missouri, USA, 233-244, April (2008).

[15] Q. Cao, T. Abdelzaher, LiteOS: a lightweight operating system for C++ software development in sensor networks, In Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, CO, USA, 261-262, Nov. (2006).

[16] T.V. Chien, H. N. Chan, T. N. Huu, A comparative study on operating system for wireless sensor networks, ICACSIS Conference on IEEE, Jakarta, Indonesia, 73-78, December (2011).

Biographies

Erhan Sesli was born in 1983. He received his B.Sc. degree in Electrical and Electronics Engineering from Karadeniz Technical University in 2005, Trabzon, Turkey. He worked in private sector a couple of years as telecommunication engineer. In 2012, he received his M.Sc. degree from Karadeniz Technical University Graduate School of Natural and Applied Sciences, Trabzon, Turkey. Erhan Sesli is currently a Ph.D. student at Karadeniz Technical University Graduate School of Natural and Applied Sciences. His research interests include range-based localization in wireless sensor networks (WSNs), optimization, metaheuristic algorithms and intelligent transportation systems.
E-mail: erhansesli@ktu.edu.tr

Gökçe Hacıoğlu received B.Sc. degree in Electronics Engineering from Karadeniz Technical University, Trabzon, Turkey, in 2000, the M.Sc. and Ph.D. degree in Electrical and Electronics Engineering with majors in wireless communication diversity techniques from Karadeniz Technical University Graduate School of Natural and Applied Sciences, Trabzon, Turkey in 2005 and 2011, respectively. His current research interests include diversity methods, routing algorithms of wireless sensor networks (WSNs), intelligent transport systems, visible light and power line communications and their applications in sensor networks.
E-mail: gokcehacioglu@ktu.edu.tr