RT-OCF: A Lightweight Device-to-Device Framework for the Internet of Things

Chanhee Lee(✉), Jaehong Jo, Jongsung Lee, Daesung An, Jaehyun Cho, and Rami Jung

Samsung Electronics, Seoul 06765, Korea
{ch2102.lee,jaehong.jo,js126.lee,daesung87.an,
jaehyun3.cho,rami.jung}@samsung.com

Abstract. Responding to rapid growth of Internet of Things (IoT) services and devices, many IoT platforms and frameworks are presented for successful IoT realization. Recently, Open Connectivity Foundation (OCF) which consists of a group of industry leaders emerges to create new standards for IoT platform and deliver an open source implementation and a certification program. IoTivity open-source software framework is one of the project sponsored by the OCF which has been developed to provide interoperability among heterogeneous IoT devices. It enables seamless Device-to-Device (D2D) connectivity and targets various application domains such as home and manufacturing automation, health care, and social networks. IoTivity uses, however, rather lots of memory in view of small devices which have limited hardware resources and runs Real-Time Operating System (RTOS). In this paper, we propose a light-weight IoT framework, RT-OCF that optimizes memory consumption and provides a memory tracer to prevent memory leaks. It has a layered architecture which consists of resource layer, messaging layer, and platform adaptation layer together with modules for security and utility. RT-OCF is able to run not only on Linux but also on TizenRT, an open source RTOS platform runnable on ARTIK053 board. The experiment performed on both Linux and TizenRT shows that more than 20% of peak memory is reduced when compared with IoTivity while preserving a packet latency for GET operations.

Keywords: IoT · D2D · Open source · Framework · OCF · TizenRT · Memory

1 Introduction

Responding to rapid growth of Internet of Things (IoT) services and devices, many IoT platforms and frameworks are presented for successful IoT realization [1–3]. There are three key factors which needs to be considered for the successful IoT realization [4]. At first, specifications should exist and describe correct mechanisms of major functionalities involving device connections and registrations in a secure way. At next, source code implements the specifications and needs to pass certification. Lastly, the source code is deployed to IoT devices of different manufacturers and is interoperable among the devices.

D. Georgakopoulos and L.-J. Zhang (Eds.): ICIOT 2018, LNCS 10972, pp. 179–187, 2018.
https://doi.org/10.1007/978-3-319-94370-1_13

Recently, Open Connectivity Foundation (OCF) [5] which consists of a group of industry leaders, e.g. Microsoft, Intel, Samsung, Cisco, and LG emerges to create new standards for IoT platform and deliver an open source implementation and a certification program. IoTivity open-source software framework [3] is one of the project sponsored by the OCF which has been developed to provide interoperability among heterogeneous IoT devices. It also enables seamless D2D connectivity and targets various application domains such as home and manufacturing automation, health care, and social networks.

IoTivity uses, however, rather lots of memory in view of small devices which have limited hardware resources, e.g., a low-cost processor and a small memory and run a RTOS. This is because IoTivity targets various common operating systems (OS) such as Linux, Windows, and Android.

In this paper, we propose a light-weight IoT framework, RT-OCF that optimizes memory consumption together with providing a memory tracer to prevent memory leaks. It has a layered architecture which consists of resource layer, messaging layer, and platform adaptation layer together with modules for security and utility. To reduce memory consumption, er-coap [6] is ported into the proposed framework. RT-OCF is not only able to run on Linux but also be compatible with TizenRT [7], an open source RTOS platform runnable on ARTIK053 board [8].

The main contributions of RT-OCF can be summarized as follows.

- RT-OCF enables easy use of OCF-based D2D communications on both a real ARTIK board running TizenRT and a Linux machine as open source software.
- Practical memory optimization techniques are introduced for the OCF specification implementation while preserving a packet latency for basic operations.
- The code qualities of both RT-OCF and IoTivity are accessed and analyzed in a quantitative manner using a static source code analyzer, SonarQube [9].

The rest of the paper is organized as follows. In Sect. 2, OCF specification on which our framework is based and TizenRT where RT-OCF is implemented are explained. Then representative D2D frameworks are introduced and compared in Sect. 3. Section 4 describes the details of each layer in RT-OCF and experimental setup and results regarding memory consumption are followed in Sect. 5. At last, the paper is concluded with future work in Sect. 6.

2 Background

In this section, OCF specification on which the proposed framework is based and TizenRT, a target RTOS on which the framework runs are described briefly.

2.1 OCF Specification

The latest version of OCF specification is 1.3 which consists of 6 categories; Core Framework, Resource Type, Device, Security, Bridging, and Wi-Fi Easy Setup. In Core Framework specification, the OCF core architecture based on the resource-oriented REpresentational State Transfer (REST) [10] architectural style is specified. It mainly

describes functional interactions such as CRUDN, messaging, and discovery as well as core features related to resource models for all OCF resources and their combinations which can be exposed by OCF devices. JavaScript Object Notation (JSON) is used for payload definitions and RESTful API Modeling Language (RAML) [11] is used for the representation for the APIs exposed to the outside of the OCF device by the OCF resources. In Device specification, a set of Device Types for use is defined. Device Types can be either mandatory to be implemented or optional and exposed also. In Security specification, device identity, authentication, and provisioning are defined for the establishment of network credentials and access controls of OCF resources. Finally, Bridging and Wi-Fi Setup specification describe translation functionality such as resource discovery and functional extensions for the capabilities to meet the requirements of Wi-Fi Easy Setup, respectively.

2.2 TizenRT

TizenRT is an open source platform based on a RTOS, called NuttX [12]. The goal of TizenRT is to extend the Tizen platform device coverage to low-end devices typically equipped with Cortex-M/R processors with MPU, less than 2 MB RAM, and less than 16 MB Flash.

It overcomes several limitations of typical RTOS targeting IoT devices. Most of existing RTOS cannot load additional modules at runtime, and support only specific libraries that prevent application developers from using and those from being spread out to various types of IoT devices. To tackle these limitations, TizenRT adopts Linux-style development environments including POSIX API, BSD Socket API, Shell, and Kconfig build configuration. This helps Linux developers build their own business logics easily on top of TizenRT. In addition, TizenRT will adopt the lightweight JavaScript environment, consisting of JerryScript [13] and IoT.js [14].

3 Related Work

There are various IoT frameworks as well as CoAP libraries, the major communication protocol in IoT frameworks. Among them, we summarize and compare two D2D IoT frameworks, i.e., IzoT [15], Thingsquare [16] which are similar to RT-OCF.

IzoT considers three main classes of IoT applications; consumer IoT, two classes of Industrial IoT based on application monitoring and Machine-to-Machine (M2M) communications. To support M2M communication-centric applications, IzoT stack consists of high level protocol services that can run on top of any IPv4 or IPv6 UDP socket interface. The stack supports end-to-end acknowledgements and responses for rapid detection of packet failures as well as fully distributed connections among network nodes to avoid single point of failure. It also provides a duplicate node detection by assigning transaction IDs with a unique mechanism and a priority messaging to allow the easy propagation of emergency messages such as notifications of node failures. Though IzoT provides various useful features for M2M communications as

mentioned above, its proprietary stack without CoAP makes it difficult to be adopted widely to low-end IoT devices.

Thingsquare supports M2M communications based on the stack of Contiki OS [17], cloud-based device governance and boot-strapping. It is composed of three parts; a device firmware, a backend stack, and user frontends. A device firmware resides in a wireless chip of a product such as an IoT lamp and a backend stack such as Thingsquare cloud provides database and message control between the wireless device and user frontends. Thingsquare, however, is only distributed as binaries which also hinder the spread itself.

4 Design and Implementation

The software architecture of the proposed framework is shown in Fig. 1. All modules in the architecture are implemented in C. For each following subsection, the detailed design and key features together with memory optimization techniques are described.

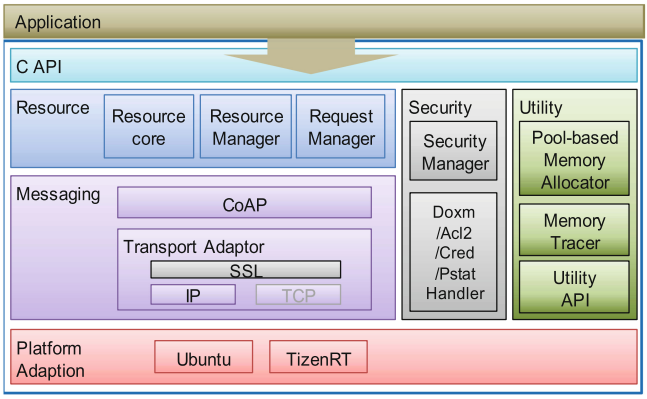

Fig. 1. RT-OCF architecture.

4.1 Architecture

RT-OCF is basically a layered architecture except functions for security and utility which are called in a resource and a messaging layer.

4.2 Resource Layer

A resource layer implements core OCF resources in the OCF specification. For the core resources which are classified with four types, i.e., device, introspection, platform and resource, APIs are provided to application users for the resource creations. In addition, there is a resource manager that operates a queue to process inner requests from other layers or request packets from the outside of the device on which RT-OCF is based. The requests are CRUDN operations mentioned in Sect. 2.1 and in general a callback is

registered for each operation in a user application. There is also a request manager to organize a request as a packet to be sent from a controller to a controlee. Before sending a request to a controlee, a controller needs to discover at least one controlee in its local network using Discovery APIs. For example, for an IoT lamp, the lamp is represented as an OCF device and the brightness of the lamp can be one of OCF resources in the OCF device. A user, the owner of the lamp, can increase or decrease the brightness of the lamp through a mechanism such as a GUI panel provided by the controller of the user after discovering and connecting to a controlee. Then the request to configure the brightness value from the controller is sent to the request manager of the discovered controlee, e.g., the lamp, followed by the value modification by the resource manager of the controlee. In addition, when a user wants to get immediate notifications about the status of interesting resources in controlees, e.g., power on/off status or power consumption levels, Observe APIs can be used to set up observers to the controlees and the notifications are sent either in the right moment that the status of the resources are changed or in a periodic manner by Notify APIs provided in this layer.

To reduce run-time memory consumption in this layer, the resource representations based on TinyCBOR [18] are simplified by omitting several wrappings and complex data structures, in contrast to the representation in IoTivity. Instead, we provide Map and Array containers to enable users to build hierarchical resources directly so that internal transformations of the OCF resources required in IoTivity can be skipped.

4.3 Messaging Layer

A message layer is divided more precisely into two layers. The upper layer is implemented with external open sources, i.e., er-coap from Contiki OS. Er-coap is known as the smallest CoAP API until now. We compared er-coap with several CoAP libraries such as libcoap, microcoap, lobaro-coap and some other open source CoAP libraries and chose er-coap with consideration for memory consumption to CRUDN operations, implementation language, and the latest update date. The comparison results are omitted due to page limits. By adopting er-coap much lighter than libcoap used in IoTivity, the run-time memory consumption can also be reduced. And the other layer stands for a transport adaptor and includes Secure Sockets Layer (SSL) and User-defined Protocol (UDP), Transmission Control Protocol (TCP) and Internet Protocol (IP). Various functions such as unicast/multi-cast, normal/secure socket, normal/secure port, and block-wise transfer are implemented in this layer.

4.4 Security Functions

In a target IoT device, all information can be encrypted and then decrypted into one of four types; doxm, acl2, cred, and pstat. Those types cover general information for device identification and description, access authority to each resource, authentication certificate, and cloud provisioning, respectively. A security manager handles initiation and invocation of information transformation to payload of a packet to be transferred and stores the information into a persistent storage in form of encrypted files.

To reduce run-time memory consumption paid for security, we store each type into a separate file and load only the encrypted files of necessary types at run-time instead of storing the whole encrypted information into one large file as IoTivity does. Though this may slightly increase the time complexity at run-time, it can prevent the security manager from loading frequently one large security file which causes the unnecessary accesses to the whole information even in the case that a specific type of the information is needed.

4.5 Utility Functions

The key modules of utility functions are a pool-based memory management module to limit run-time memory consumption and a memory tracer to detect memory leaks. To analyze memory leaks, a specific memory allocation function and a tracer are provided and to log all dynamic memory allocations. In addition, basic data structures such as list and queue as well as string, random, timer, and thread functions are provided as necessary to all the other layers and modules.

5 Experiments

In this section, the performance metrics, i.e., packet latency and run-time memory consumption and the code quality of the proposed framework are evaluated.

5.1 Setup

For fair comparisons with IoTivity v1.3.0, the performance metrics are measured upon both TizenRT and Linux. A network model used when measuring the performance metric on TizenRT are shown in Fig. 2(a). The model consists of two nodes, i.e., a controller and a controlee. As a controller, a simple android application which follows OCF 1.0 specification and can send a group of GET messages to the controlee is developed and used for the experiments. A Galaxy S6 phone is used to execute the application. RT-OCF runs as a controlee upon TizenRT OS flashed into an ARTIK 053 board. Note that the ARTIK 053 board is changed into a desktop PC running Ubuntu 14.04 in case Linux is used rather than TizenRT.

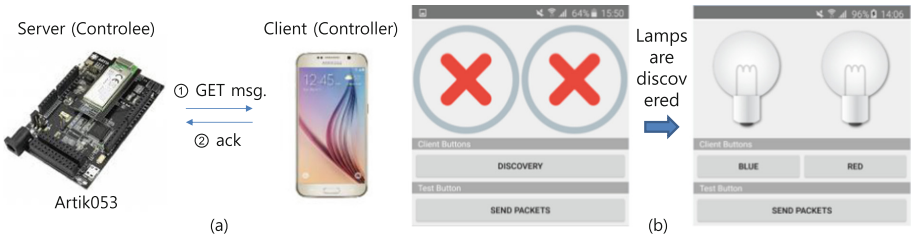

Fig. 2. (a) Network model for performance tests (b) Android application GUI for a controller.

The relevant user interface (UI) of the application for the test with GET operations is shown in Fig. 2(b). At first, a discovery to find the resources on the controlee are performed in the controller after the android application is executed. Once the discovery is finished successfully, GET messages are sent in succession from the controller to the controlee after each ack message of the former packet is received.

5.2 Results

To measure and compare memory consumption, peak memory during a GET operation at run-time is used. It is calculated as the sum of stack and heap memory usage measured by valgrind [19] and heapinfo for Linux and TizenRT, respectively. Heapinfo is an internal memory analysis tool in TizenRT. The results of peak memory normalized with that of RT-OCF when the memory tracer is disabled are shown in Fig. 3. For both Linux and TizenRT, IoTivity consumes 416.7% and 213.8% more memory than RT-OCF, respectively. About 58.5 Kbytes memory reductions of RT-OCF on TizenRT can be achieved by the adaptation of the lighter CoAP library, the simplification to data structures of OCF resources, the reduction of the number of Send/Receive threads in the messaging layer, and the separation of the encrypted information files for each type. The peak memory when the tracer is enabled during the test is also measured to validate the availability of the memory tracer. The tracer increases the memory only by 13.8%, i.e., 7.1Kbytes on TizenRT.

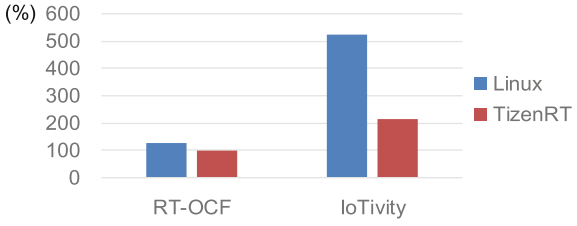

Fig. 3. Peak memory consumption of RT-OCF and IoTivity on Linux and TizenRT.

On the other hand, we also measure an average packet latency during the peak memory measurements to check whether performance degradation is serious or not due to memory optimization techniques. The packet latency is measured with the most representative operation among CRUDN operations, i.e., GET. The results were about 77 ms and 66 ms for RT-OCF and IoTivity on TizenRT, respectively. For Linux, the latencies of both RT-OCF and IoTivity for a GET operation are almost similar as about 32 ms meaning that the performance degradation in RT-OCF is insignificant.

Lastly, the code quality of our open source framework is evaluated using SonarQube. It can estimate code quality metrics such as Lines of Code (LoC), code complexity, duplication lines, and maintainability which mean the sum of total lines of source files, degree of the branch usages such as if, switch and while, duplicated source codes except function calls, and modification efforts to follow standards such as coding rules. The results for both RT-OCF and IoTivity are shown in Table 1. For exact comparison, only

D2D parts of source codes in IoTivity except source codes related to cloud connections are identified and used for this evaluation. The results show that all quality factors are greatly improved than those of IoTivity. Especially for duplicated lines, RT-OCF overwhelms IoTivity. Through code reviews, it is found that the same structures and functions are declared as static and used repetitively in IoTivity. Note that the code quality factors are not proportional to LoC. The LoCs for both frameworks are represented as references for the scale of the frameworks.

Table 1. Comparison with IoTivity framework in code quality

	LoC	Complexity	Duplication lines	Maintainability
RT-OCF	10,887	24.4	68 (0.5%)	31d (1549)
IoTivity	*48,068*	71.5	841 (1.3%)	58d (2355)

6 Conclusion

In this paper, an OCF-based lightweight open source D2D framework, called RT-OCF is proposed. The key contribution of RT-OCF is to enable D2D communication based on an open source RTOS Platform, called TizenRT minimizing run-time memory consumption which is critical to resource-constrained IoT devices. The performance and code quality are also evaluated upon a real IoT device, i.e., ARTIK 053. The results show that the proposed framework reduces the memory consumption efficiently as well as achieves better code quality than the case of IoTivity. The future work is to improve the memory management method using buddy system to minimize memory fragmentation and measure additional performance metrics such as power consumption with more complex usage scenarios. And functionalities such as onboarding when an IoT device is in out-of-box state, TCP communication, Secure Socket Layer (SSL) based on certificate, and provisioning to clouds will be expanded.

References

1. Perera, C., Jayaraman, P.P., Zaslavsky, A., Georgakopoulos, D., Christen, P.: Mosden: An internet of things middleware for resource constrained mobile devices. In: 47th HICSS, pp. 1053–1062. IEEE, Waikoloa (2014)
2. Razzaque, M.A., Milojevic-Jevric, M., Palade, A., Clarke, S.: Middleware for internet of things: a survey. IEEE Internet Things J. **3**(1), 70–95 (2016)
3. Dang, T.-B., Tran, M.-H., Le, D.-T., Choo, H.: On evaluating IoTivity cloud platform. In: Gervasi, O., Murgante, B., Misra, S., Borruso, G., Torre, Carmelo M., Rocha, Ana Maria A.C., Taniar, D., Apduhan, Bernady O., Stankova, E., Cuzzocrea, A. (eds.) ICCSA 2017. LNCS, vol. 10408, pp. 137–147. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62404-4_10
4. Derhamy, H., Eliassan, J., Delsing, J., Priller, P.: A survey of commercial frameworks for the Internet of Things. In: ETFA 2015. IEEE, Luxembourg (2015)
5. Park, S.: OCF: A new open IoT consortium. In: 31st International Conference on Advanced Information Networking and Applications Workshops, pp. 356–359. IEEE, Taipei (2017)

6. Kovatsch, M., Duquennoy, S., Dunkels, A.: A low-power CoAP for Contiki. In: 8th ICMASS. IEEE, Valencia (2011)
7. TizenRT RTOS. https://source.tizen.org/documentation/tizen-rt. Accessed 30 Mar 2018
8. ARTIK 053. https://www.artik.io/modules/artik-05x/. Accessed 30 Mar 2018
9. Ann Campbell, G., Papapetrou, P.P.: SonarQube in Action, 1st edn. Manning, Greenwich (2013)
10. Feng, X., Shen, J., Fan, Y.: REST: an alternative to RPC for web services architecture. In: 1st ICFIN. IEEE, China (2009)
11. RAML. https://raml.org/. Accessed 30 Mar 2018
12. NuttX. http://www.nuttx.org/. Accessed 30 Mar 2018
13. JerryScript. http://jerryscript.net/. Accessed 30 Mar 2018
14. IoT.js. http://iotjs.net/. Accessed 30 Mar 2018
15. IzoT. https://www.echelon.com/izot-platform. Accessed 30 Mar 2018
16. Thingsquare. https://www.thingsquare.com/. Accessed 30 Mar 2018
17. Dunkels, A., Gronvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: 29th ICLCN, IEEE, Tampa (2004)
18. TinyCBOR. https://github.com/intel/tinycbor. Accessed 30 Mar 2018
19. Nethercote, N., Seward, J.: Valgrind: a framework for heavyweight dynamic binary instrumentation. In: 28th PLDI, pp. 89–100. ACM, San Diego (2007)