A Study on OPNET State Machine Model Based IoT Network Layer Test

Young-hwan Ham[1(✉)], Hyo-taeg Jung[1], Hyun-cheol Kim[2], and Jin-wook Chung[3]

[1] Quality Innovation Team, ETRI, 218 Gajeong-ro, Yuseong-gu, Daejeon, Korea
{yhham,htjung}@etri.re.kr

[2] Computer Science, Namseoul University, 21 Maeju-ri, Seonghwan-eup, Cheonan, Chungcheongnam-do, Korea
hckim@nsu.ac.kr

[3] Computer Engineering, SungKyunKwan University, 2066 Seobu-ro, Jangan-gu, Suwon-si, Gyeonggi-do, Korea
jwchung@skku.edu

Abstract. Model based testing can enable automated test case generation for many kind of application. Even test code can be generated from the model by specialized tools. IoT protocols for network layer have many constraints for exhaustive or manual testing because of battery problem and large number of sensor nodes. To solve these testing constraints, this paper proposes an efficient State Machine based test case generation for IoT network layer by using OPNET simulation model and test case generation tool. The size of test suite is compared according to the size of State Machine model from OPNET.

Keywords: State machine · IoT · Network layer test · Test generation · OPNET

1 Introduction

IoT (Internet of Things) normally has hundreds or thousands of sensor nodes and battery constraints in case of outdoor field test. Therefore, it is necessary to efficient testing method for the IoT.

In addition, it is necessary to consider an application layer interaction which is useful for dynamics caused by mobility, failures, and dynamic power modes of IoTs. The traditional layered structure passes a limited set of information over defined interfaces between separate layers of the protocol. It is good for abstraction and development, but bad for efficiency in case that high level information is useful in over layers or vice versa. The examples are power control, overlay service, error control, aggregation, fusion, localization, service discovery, semantic addressing, etc.

In this study, we are going to use State Machine-based testing for the cost saving in test case design, systematic testing and controlling of the model coverage and the number of tests [1, 2]. It can help the early detection of flaws and ambiguities in the specification, and the conformance of implementation to the corresponding State Machine model.

For the State Machine based testing of IoT protocol, application layer and network layer should be reflected on the protocol State Machine to cover the standard specification. It is very critical to limit the number of test case in IoT because of battery power constraints, so it is necessary to draw efficient test cases [3].

2 OPNET Modelling for Test Case Generation

2.1 IoT Network Layer and OPNET Simulation

ZigBee sensor network standard, which is a representative low-power standard for IoT applications, was modelled by OPNET [4].

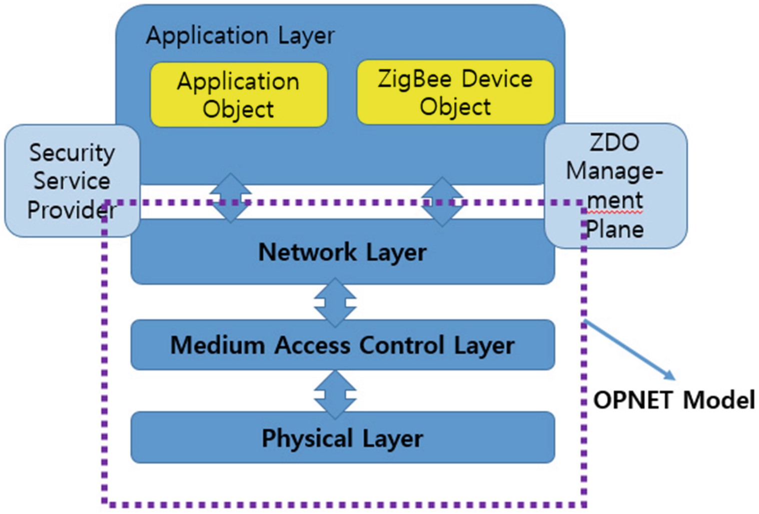

Fig. 1. The OPNET model of zigbee protocol stack

A simulation model based on OPNET was developed for the simulation of sensor networks. Through the OPNET based simulation, various parameters related to the sensor network can be set in advance to find suitable ones for the application system. The candidate technology or structure to be applied to the developed system can be evaluated in advance to receive feedback. Using the existing OPNET library, the interoperability between different protocols and systems can be verified in advance [5] (Fig. 1).

1. Physical layer: This layer is the lowest layer. It consists of two layers, operating in two separate frequency ranges [4].
2. Medium Access Control layer: The responsibility of the MAC layer is to control access to the radio channel using CSMA/CA. The MAC layer provides support for transmitting beacon frames, network synchronization and reliable transmission [4].
3. Network layer: This layer sends and receives data to and from the application layer. It performs the task of associating to and disassociating from a network. This layer network protocol allows us to extend the battery life of the nodes, allowing it to do

only the minimum work when it needs to transmit data [4]. The emphasis is on very low cost communication of neighboring devices with no other wired/wireless network infrastructure. The low cost communication results in lower power consumption, which is even more important.

The following shows the result of simulating a Beacon-enabled ZigBee Network using the ZigBee library.

- OPNET Simulation: End-to-end delay and Receiver-on time of ZigBee network
  - Mode 1: Random Beacon Slot (Beacon Enabled Mode)
  - Mode 2: Proposed Beacon Scheduling (Beacon Disabled Mode)

Figure 2 shows that the delay increases exponentially with Beacon-Disabled mode as sensor node increases. Figure 3 shows that the beacon-enabled mode has much less awake time than the beacon-enabled mode, which is much better in terms of battery consumption.

In a ZigBee application that generates traffic with a frequency lower than a certain level, such as remote meter reading and environmental monitoring, the battery usage time can be greatly improved when the beacon-enabled mode is applied. However, it can be adversely affected in a heavy traffic environment. Through the event/traffic simulation results, the correct operation of OPNET model including network layer has been verified.

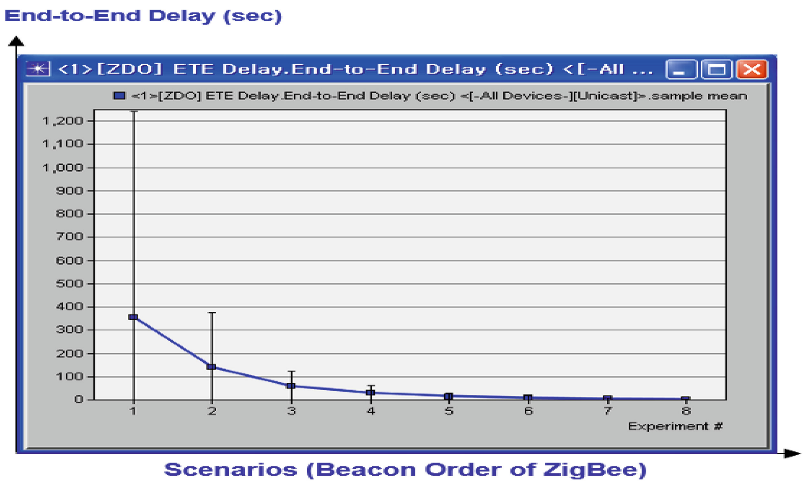

Fig. 2. End-to-End Delay simulation result by OPNET model

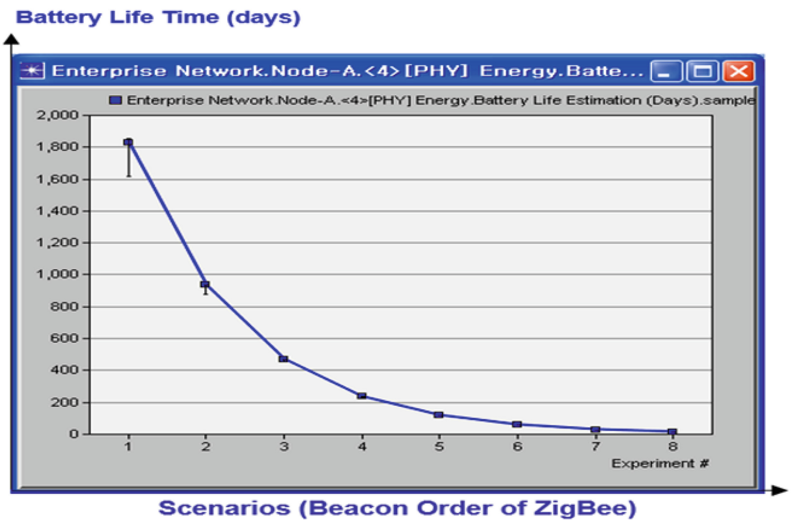

Fig. 3. Receiver-on time (Battery Life Time) simulation result by OPNET model

2.2 Network Layer State Machine for Test Case Generation

OPNET uses a state machine based modeling technique to simulate each layer. Figure 4 shows the network protocol layer state machine of the implemented model. We propose a method to efficiently generate a test case by using the state machine. The circles below represent each state, and the terms on the arrows represent interrupts.

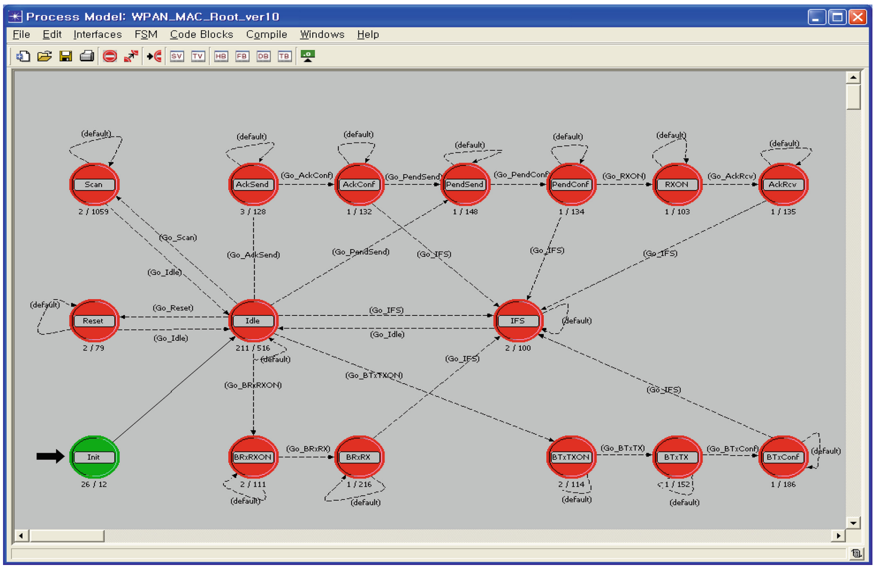

Fig. 4. The State Machine of OPNET model network layer

In this way, the finite state machine that defines the operation of the network layer can check whether the network layer of the actually implemented sensor node is operating properly. An automated tool such as ModelJunit [6] can be used to obtain a test case that can test the operation of the network using the network state transition diagram. ModelJunit is an available and prevalent State Machine based test case generation tool. It is very easy to learn and convenient because it is based on Java language.

3 Experiment Result and Analysis

The State Machine of OPNET network layer is slightly modified and simplified for excluding the meaningless interrupt in point of test case generation, such as “default”. The final state machine is as follow. The possible test cases for network layer can be generated from this state machine diagram by using appropriate test case generation tool.

The experiments for test case generation were executed by ModelJUnit, and each experiment result has averaged among 10 times executions of test.

The comparison experiment has been performed by using Random Walk & Greedy Random Walk test case generation algorithm [6]. Random Walk algorithm simply tests a system by making random walks through a State Machine model, and Greedy Random walk gives priority to transition never taken before. In addition to generation algorithm, test case coverage is also important factor and the ModelJUnit supports three kinds of coverage metrics such as state metric, event metric, and transition metric [7]. The state metric shows how many states are traveled at least once. The event metric shows how many events are triggered at least once. Transition metric represents how many transitions are exercised at least one was chosen in this study because it is important to cover every state transition to ensure correct operation of the network layer [7]. The details of experiments for comparison are as follows.

- **Experiment 1: The number of state is same as original OPNET network model**

When the state machine model had 16 states and the number of interrupt (event) was 25 (Fig. 5), test length (the number of test suite) for 100% transition coverage metric was as follows.

- State Machine Model: 16 states, 25 event
- Random Walk: 210
- Greedy Random Walk: 160

When the relatively small number of State Machine states were randomly added for test, the number of test length (Random Walk) was exponentially increased.

- **Experiment 2: The number of state is reduced by simplifying original OPNET network model**

The similar group of states are merged as follows for the simplification of state machine mode as follows.

- (ACK SEND, ACK CONF) => ACK SEND

- (PEND SEND, PEND CONF) => PEND SEND
- (RXDN, ACK RCV) => RXDN
- (BRxRXON, BRxRX) => BRxRXON
- (BTxTXON, BTxTX, BTxCONF) => BTxTXON

When the Network had 10 states and the number of interrupt (event) was 19 (Fig. 6), test length was as follows.

- State Machine Model: 10 states, 19 event
- Random Walk: 170
- Greedy Random Walk: 50

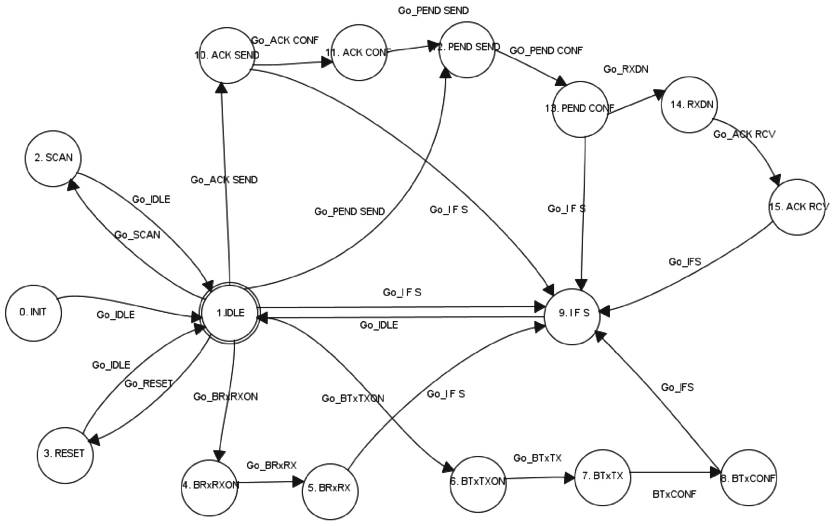

Fig. 5. The State Machine of network layer for ModelJUnit

The test length for the network testing can be very critical in such resource constrained IoT environment. The experiment 2 shows that simplified by state merging can dramatically reduce the number of test case especially in case of greedy random walk. It should be considered that how we can reduce the number of states by simplifying the state machine from OPNET model. The simplified version of state machine model can be also verified by putting & executing it in OPNET modeler. The effect of merged states can be monitored by the simulation result of OPNET. This can be a reciprocal way for an efficient test case generation and network simulation.

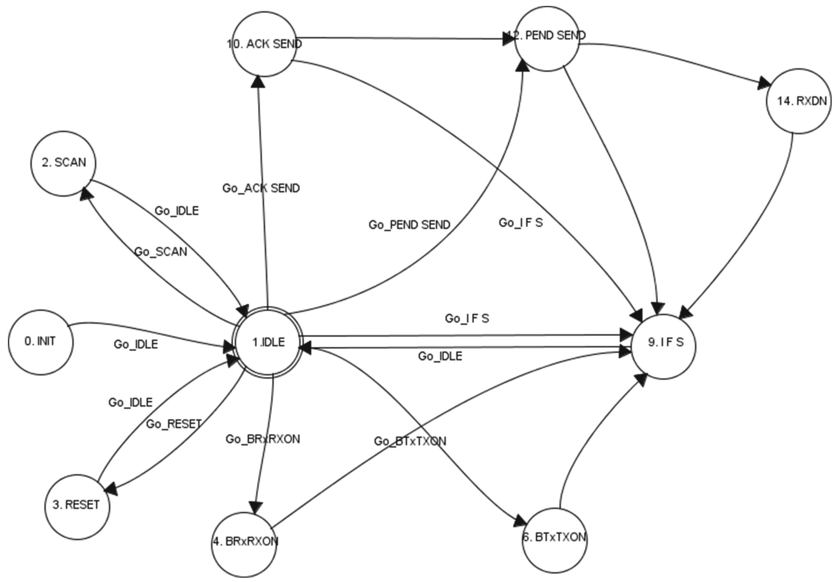

Fig. 6. The Simplified State Machine of network layer for ModelJUnit

4 Conclusions

The IoT normally has hundreds or thousands of sensor nodes and battery constraints in case of outdoor field test. Therefore, it is necessary to efficient testing method for the IoT. In addition, it is necessary to consider a network layer operation which is useful for dynamics caused by mobility, failures, and overlay modes of IoTs.

For the State Machine based testing of IoT protocol, ModelJUnit tool and OPNET ZigBee model are used. The tool generated the test cases by using OPNET network layer state machine model. By the result of these experiments, we realized that the test cases can be generated by using the state machine model of OPNET. Because the number of test length could be rapidly increased in proportion to the number of state machine, a simplification of FSM is necessary. The effect of merged states can be checked & monitored by simulating the simplified model in OPNET.

The more various test case generation experiment is also necessary for verification of the network layer design. The ModelJUnit has many benefits as a tool of State Machine based test case generation. We have used transition-tour test generation algorithm by the tool, but it doesn't support other test sequence generation methods [8, 9]. The study about overcoming above weaknesses also should be done in the future.

References

1. Gansner, E., North, S.: An open graph visualization system and its Appl. Soft. Pract. Experience **30**, 1203–1233 (1999)
2. Javed, A., Strooper, P., Watson, G.: Automated generation of test cases using model-driven architecture. In Proceedings of the 2nd International Workshop on Automation of Software Test (AST 2007), p. 3 (2007)
3. Link, J., Fröhlich, P.: Unit Testing in Java: How Tests Drive the Code. Morgan Kaufmann Publishers Inc., Burlington (2003)
4. IEEE Standards Association: IEEE standard for local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) (2011). http://standards.ieee.org/about/get/802/802.15.html
5. Xiaolong, L., Peng, M.: OPNET-based modeling and simulation of mobile Zigbee sensor networks. Peer-to-Peer Netw. Appl. (2015). doi:10.1007/s12083-015-0349-8
6. http://www.cs.waikato.ac.nz/~marku/mbt/modeljunit/ (2016)
7. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach, pp. 157–162. Morgan Kaufmann Publishers Inc., San Francisco (2007)
8. Lelis, L., Pedrosa, C.: A new method for incremental testing of finite state machines. In: NFM2010
9. Ural, H.: Formal methods for test sequence generation. Comput. Commun. **15**(5), 311–325 (1992)