*Proceedings of the 33rd International Conference on Information Technologies (InfoTech-2019)*
*19-20 September 2019, Bulgaria*

# SIMULATION-BASED SELF-TESTING IN IoT-ENABLED MANUFACTURING

***Digest of paper*[1]**

**Iliya Georgiev, Ivo Georgiev**

*Department of Mathematics and Computer Sciences*
*Metropolitan State University of Denver*
*e-mails: {gueorgil, igeorgi1}@msudenver.edu*
*USA*

**Abstract:** The paper presents self-testing practical approaches for stabilizing a network of Internet of Things (IoT) enabled manufacturing cells. The key technology of such self-testing is simulation of the basic functions.
**Key words:** IoT-enabled manufacturing, simulation-based self-testing, XML stream.

## 1. INTRODUCTION

Dominating trend in manufacturing is continual integration of embedded systems, which are connected in local area networks (LAN) and open to the world by IoT [1]. Such intelligent manufacturing consists of various industrial and computer components (some of them legacy). The incorporated embedded systems have significant computational power to run a whole range of processes, from hard real-time to batch. The paper presents practical methods to self-testing based on simulation. The manufacturing cells are influenced by industrial noises and diversity of adversaries through Internet. The objective of the presented methods is to check dynamically the stability of the provided functionality of manufacturing resources.

## 2. A MODEL OF IoT-ENABLED MANUFACTURING

In this section, we lay down a prototypical model of factory automation and discuss the functionality stack. Every cell (machine, robot, automatic test system, quality assurance instruments, 3D printer, etc.) is driven by an IoT-enabled embedded

[1] The full paper is proposed for including in the IEEE Xplore Digital Library

system that has a variety of peripheral devices (timers, analog-to-digital and digital-to-analog converters, serial and parallel interfaces, analog comparators, power width modulators, USB, networking, etc.), which exchange data from production sensors and control actuators. The embedded operating system supports Internet layer stack and controls hard and soft real-time multithreading.

The embedded systems are connected in a time-sensitive extension of Ethernet. Every IoT controller is an Internet host and supports application specific protocols for messaging. The computer aided design (CAD) center exchanges data with the IoT cells via Internet.

The suggested information-exchanging technology is an XML stream that is circulating through the connected devices [2]. The stream is divided into XML elements that are objects of different encryption and contain only metadata for classified access and references to production instructions and data. The stream elements are oriented to some IoT cell and have references to concrete industrial hierarchy procedures. IoT application software use web services to get the stream but can decrypt only the sections that are oriented to its production role.

The message-passing technology between the IoT embedded systems and other hosts is over web services [3] that are carriers of the XML stream.

## 3. SELF-TESTING SCENARIO

### 3.1. Testing of the stable state of the IoT embedded system

The testing goal is to prove the stable state of the embedded system. Stabilization requires fixing the inputs in some initial state so that the threads can be executed repeatedly with predictable outputs. We run a simulator subroutine that stabilizes all the inputs. Once the system is stabilized and additional tests are activated, there is a guarantee that the changes in the outputs are functions of the new test operation and not due to the input unstable values.

We define the following steps for the stable state. First, the IoT embedded system is initialized by a well-defined procedure: the kernel of the operating system has been initialized; the interrupt handlers are accessible; the peripheral devices are in reset state; no ports are initialized. Second, the ports and the peripheral devices are activated in running state without any processing. The interrupt handler vector addresses are compared and verified. Send and receive functions are activated with dummy parameters. Third, the execution speed of an embedded system is determined by some clock control with the option of programmatic change to the main clock frequency. Part of the stable state analysis is to simulate the clock control by cycling over all possible frequencies. Lastly, the Internet connection is verified by checking the network address translation. The simulator organizes a bouncing mode for IP address translation and checks the possible tunneling between IPv4 and IPv6 formats. Additionally, it verifies the internal path tables and DNS (Domain Name Service) support.

Communication with CAD computers is verified by handshaking messages and web services by exchanging the roles of clients and servers.

### 3.2. Simulator of IoT embedded system and timing testing

The dominating trend is to implement time-sensitive networking, which is intended to address the timing and synchronization with standard Ethernet technology. The accepted testing approach follows the next scenario.

The testing unit initializes communication with guaranteed end-to-end latency. It supports a closed loop control that is operating between two IoT embedded systems. The communication on the Ethernet network is separated into fixed length repeating time slices. Within these cycles, it is possible to prioritize those traffic items that can't be preempted.

The testing contains the following procedures:

a. Check of the common time of the embedded system under test. For real time communication with hard restrictions to the transmission delays, all network systems must have a common time reference and synchronize their clocks among each other.

b. Organizing a dialog through the time-sensitive network and checking it by digital signatures. The testing with digital signatures is organized in three levels: testing of the network data streams, testing of the timing characteristics, and testing of the interface with the sensors and actuators. In all levels a signature of this data sequence is calculated and stored as an etalon signature. Periodically, new signature is calculated and compared to the etalon. If they are not equal, the sequence has been violated by a software bug, data degradation, or improper behaviour of the sensors and actuators. The signature is a 64-bit polynomial calculated by hashing [4].

c. Implementing non-intrusive tests that are periodically activated in the tested IoT production cell. Non-intrusiveness of the test allows the IoT system to operate normally in time as if the test did not run. Our nonintrusive test is based on the popular technique of merging testing procedures with critical operations. Such procedures store valid information into an array at run time. Later, the simulator analyses the information in the array in background mode.

### 3.3. XML stream simulation

The XML section originated to a cell starts with a non-encrypted unique identifier of the embedded system. Every embedded system reads the stream and, by comparing the identifiers, can read or not the encrypted body using the crypto key.

The integrity and the authenticity of the section are proved by digital signatures. A testing thread runs in background mode parsing the section and, by taking the role of the author, inserts into the stream section a report, which contains the results of the tests for some period. At the end of the report, there is a general assessment of the stable state of the manufacturing cell. The Internet software stack must respond adequately to the injected XML stream, reading its designated section and responding within the report. If some embedded system does not perform correctly, it is declared in non-stable state. The simulator runs in one of the IoT system and generates the following sequence to every other system under test: initialization by stimuli to invoke

a specific state; validation of the communication; capturing content of some of the packets; verification and concluding assessment of the desired functionality.

### 3.4. Web services as transactions

The execution of the web service test has to be atomic. It cannot be preempted in the middle by some error because the cell can remain in an unpredictable and/or unreliable state. For that reason, we wrap all operations that are invoked by web services as transactions. This assures that all operations are encapsulated entirely in a single logical unit of work and are executed in an all-or-nothing mode. Transactions can commit (fully successful) or abort (transaction failed). Aborted transactions must be rolled back to undo any changes they performed.

### 3.5. IoT device emergency reboot

In our development, we implement rebooting for recovering the initial state. The reboot is partial by deploying some properties of the development environment, applying them only to the interrupt handlers that manipulate the machinery sensors and actuators. When some stuck-at-unknown state conditions occur, some coarse monitoring of each individual interrupt handler takes place, prompting a restart. Handlers that work with sensors and actuators implement a procedure that allows rebooting of the external devices into a well-known state. The procedure implements the policy that a thread is unavailable unless it replies on time. Handlers analyze each interrupt and possible related errors and, depending on the nature of the error, may force the embedded system to ignore them or reboot.

## 4. CONCLUSION

In this paper, we have focused on the following contributions: design of a testing suite based on a combination of functional simulation methods; testing timing characteristics of the time-sensitive network of IoT embedded systems that is organized by digital signatures incorporated into the simulator; implementing web services, as the main technology for Internet messaging, by commit-or-rollback transactions.

## REFERENCES

[1] IoT standard and protocols, (2019). *2019 comparisons of networks, protocols and standards*, (*available at:* https://www.postscapes.com/ internet-of-things-protocols/).

[2] Freeman, E., Gelernter, D., (2012). Document Stream OS, *US Patent App. US6006227A.*

[3] Lemos, A.L., Florian, D., and Boualem, B., (2016). Web service composition: a survey of techniques and tools, *ACM Computing Surveys* (CSUR) 48.3, 33.

[4] Daemen, J., Rijmen, V., (2003). AES Proposal: Rijndael (PDF). *National Institute of Standards and Technology*, Archived (PDF) from the original on 5 March 2013.