electronics

Article

CLOCIS: Cloud-Based Conformance Testing Framework for IoT Devices in the Future Internet

Jaehoon Yoo [1,†], Jaeyoung Hwang [2,†], Jieun Lee [1], Seongki Yoo [3] and JaeSeung Song [1,*]

[1] Department of Computer Security and Convergence Engineering for Intelligent Drones, Sejong University, Gwangjin-gu, Seoul 05006, Republic of Korea; metoofire@sju.ac.kr (J.Y.); love9ly@sju.ac.kr (J.L.)
[2] AI Trustworthiness Verification Team, Telecommunications Technology Association, Bundang-gu, Seongnam 13591, Republic of Korea; jyhwang@tta.or.kr
[3] Center for Future Transport and Cities, Coventry University, Coventry CV1 5FB, UK; ad3869@coventry.ac.uk
[*] Correspondence: jssong@sejong.ac.kr
[†] These authors contributed equally to this work.

Abstract: In recent years, the Internet of Things (IoT) has not only become ubiquitous in daily life but has also emerged as a pivotal technology across various sectors, including smart factories and smart cities. Consequently, there is a pressing need to ensure the consistent and uninterrupted delivery of IoT services. Conformance testing has thus become an integral aspect of IoT technologies. However, traditional methods of IoT conformance testing fall short of addressing the evolving requirements put forth by both industry and academia. Historically, IoT testing has necessitated a visit to a testing laboratory, implying that both the testing systems and testers must be co-located. Furthermore, there is a notable absence of a comprehensive method for testing an array of IoT standards, especially given their inherent heterogeneity. With a surge in the development of diverse IoT standards, crafting an appropriate testing environment poses challenges. To address these concerns, this article introduces a method for remote IoT conformance testing, underpinned by a novel conceptual architecture termed CLOCIS. This architecture encompasses an extensible approach tailored for a myriad of IoT standards. Moreover, we elucidate the methods and procedures integral to testing IoT devices. CLOCIS, predicated on this conceptual framework, is actualized, and to attest to its viability, we undertake IoT conformance testing and present the results. When leveraging CLOCIS, small and medium-sized enterprises (SMEs) and entities in the throes of IoT service development stand to benefit from a reduced time to market and cost-efficient testing procedures. Additionally, this innovation holds promise for IoT standardization communities, enabling them to champion their standards with renewed vigor.

Keywords: conformance testing; cloud-based testing; Internet of Things; oneM2M standards; IoT testing; TTCN-3

check for updates

Citation: Yoo, J.; Hwang, J.; Lee, J.; Yoo, S.; Song, J. CLOCIS: Cloud-Based Conformance Testing Framework for IoT Devices in the Future Internet. *Electronics* **2023**, *12*, 4980. https://doi.org/10.3390/electronics12244980

Academic Editors: Dimitrios Dechouniotis, Marios Avgeris, Konstantinos Tsitseklis and Vitoropoulou Margarita

Received: 11 November 2023
Revised: 6 December 2023
Accepted: 7 December 2023
Published: 12 December 2023

1. Introduction

In recent times, the Internet of Things (IoT) has emerged as a promising solution to address various societal challenges [1]. By seamlessly integrating a plethora of devices—ranging from sensors and actuators to vehicles—it is anticipated that billions of devices will be incorporated into diverse sectors such as home automation, industrial automation, medical aids, the automotive industry, and several others [2–5]. As a result, IoT services are becoming ubiquitous, affecting every industrial domain. Ensuring a sustainable and high-quality delivery of IoT services is paramount to prevent grave risks stemming from malfunctioning devices or products that fail to interoperate [6].

Given this backdrop, IoT testing has evolved as an indispensable facet of IoT technologies [7,8]. In particular, conformance testing is deemed a critical component of IoT testing to ensure a certain level of quality assurance. It is employed to validate whether systems or devices adhere to standards established by telecommunications bodies [9,10]. Furthermore, the International Telecommunication Union (ITU) acknowledges the significance of conformance testing to ensure that independent implementations based on a uniform standard remain interoperable [11]. However, conventional testing methodologies are ill suited for IoT testing, which presents several challenges:

- Testing heterogeneity (issue #1): Presently, numerous IoT standards organizations are formulating and upholding IoT standards. However, only select entities, such as oneM2M, are crafting specifications for conformance testing. The majority of research on IoT testing does not touch on standard-based IoT testing.
- Testing environment (issue #2): IoT services inherently rely on a diverse environment influenced by varying protocols and standards [6,12]. Consequently, designing a bespoke IoT testing system to accommodate this heterogeneity is both challenging and costly.
- Testing intervention (issue #3): Traditional IoT device testing often requires manual human intervention, which proves inefficient for testing a multitude of devices. Human interference can also inadvertently skew test results. Hence, automation of the testing process is imperative [13].
- Testing cost (issue #4): Prior to product launch and certification by authoritative bodies, face-to-face testing necessitates both testing systems and experts to be co-located, leading to high expenses and resource consumption [11].

In light of these challenges, we introduce CLOCIS, a cloud-based IoT conformance testing framework developed using TTCN-3, a specialized protocol testing language. CLOCIS offers scalability for a diverse array of IoT standards and facilitates remote testing of IoT systems adhering to different standards. Specifically, issues #1 and #2 are tackled by dynamically incorporating IoT testing modules catering to different standards. Addressing issue #3, dedicated components and procedures for device testing have been integrated into the CLOCIS system. Moreover, web interfaces that empower testing professionals to remotely configure and execute conformance tests mitigate the need for physical presence in the testing lab, thereby resolving issue #4. We also present the results of IoT conformance testing utilizing CLOCIS to validate the proposed architecture's feasibility.

In conclusion, leveraging CLOCIS can yield the following benefits:

- Elimination of the necessity for distinct testing tools to evaluate other IoT standards, meaning the versatile CLOCIS system can cater to a spectrum of standard-based IoT systems.
- Automated testing minimizes human intervention, preventing potential discrepancies in results.
- The remote testing infrastructure offers small and medium-sized enterprises (SMEs) and IoT service-developing companies a timely and cost-effective solution, negating the need for physical lab visits. Moreover, IoT standardization bodies can expedite the dissemination of their standards.

The subsequent sections of this paper are structured as follows: Section 2 reviews the existing literature on IoT conformance testing. Section 3 elaborates on the fundamental concept of conformance testing and the pivotal role of TTCN-3. Following a detailed introduction to conformance testing and its associated testing language, Section 4 delves into the CLOCIS architecture and supplementary IoT device testing approaches. Section 5 demonstrates actual IoT conformance testing using CLOCIS, employing oneM2M as the international IoT standard. Section 6 concludes the article and outlines prospective endeavors pertaining to the IoT standard conformance testing tool.

2. Related Work

IoT testing differs markedly from traditional software testing. Since IoT devices comprise an integration of hardware and software and interact not only with humans but also with other machines (i.e., machine-to-machine communication), which includes networking aspects [14,15], the inclusion of a non-human perspective is considered a crucial aspect of IoT testing [16,17].

Consequently, the complexity of IoT testing has spurred research efforts to propose testing methods across a range of domains. The research outlined in [18] briefly introduces a concept for an IoT testing framework that integrates simulation with unit, integration, and end-to-end testing. Additionally, Ref. [19] investigates the compliance of IoT devices with privacy policy agreements and develops models to assess the degree of compliance in terms of privacy criteria. However, these approaches lack a cloud-based testing concept. In other words, both the testing systems and the test experts must be physically co-located to conduct product tests.

The advent of cloud-based testing systems presents a formidable solution to previously encountered challenges. The inherent attributes of cloud computing, such as enhanced service delivery [20], reduction in production costs and time [21], and increased responsiveness to changes in requirements, are leveraged to augment service provision. Consequently, cloud computing serves as a viable platform for conducting IoT testing, with the added functionalities of logging and result viewing accessible remotely. The F-Interop framework [22], underpinned by cloud technology, furnishes testing professionals with a remote environment that supports a diverse range of IoT standards and protocols. However, it primarily focuses on interoperability testing. In parallel, Ref. [23] provides support for remote IoT testing, with a particular focus on identifying compatibility issues at the firmware level of IoT devices.

The practice of continuously testing small changes in software systems has proven to be beneficial and is widely adopted in software development [24]. In this approach, software is tested in environments that closely mimic production settings. However, when applied to IoT systems, this method faces unique challenges due to factors such as the large scale of deployments, the heterogeneity of devices, complex network characteristics, and the intricate integration with their operational environments. IoT test environments offer a potential solution by emulating nodes, networks, and domain environments for executing IoT applications. This approach addresses the testing challenges associated with minor modifications in IoT devices, though it still necessitates that IoT devices provide a specific testing interface.

The article by Chernyshev et al. explores the current state of IoT research, including simulators and testbeds [25]. They identified key research topics and objectives for IoT and conducted a comparative study of nine simulation tools based on their coverage of IoT architecture layers as well as three large-scale IoT hardware testbeds. They pinpointed three major challenges in IoT testing: the lack of support for common IoT communication standards, insufficient end-to-end service simulation across IoT layers, and significant discrepancies between simulator and real-world test results. CLOCIS addresses standardized IoT device testing and cloud-based distributed IoT device testing. However, as noted in our limitations section, CLOCIS currently does not support the concept of test offloading.

Linghuan et al. [26] introduced a practical graph-based combinatorial testing framework, CT-IoT, designed to automatically select testing paths for efficient IoT testing. They detailed the framework's operation and its application in two real-world IoT systems. Additionally, the authors proposed four coverage criteria to aid testers in assessing the comprehensiveness of IoT system testing. CT-IoT was evaluated on eight real-world systems from the industry, demonstrating its practicality and effectiveness. CT-IoT offers a solution for identifying which IoT devices require testing and the network paths to access these devices. However, it is a proprietary testing solution and is still limited to test functions that require initiation by the testing IoT device.

Moysis et al. [27] introduced Fogify, an emulator that simplifies modeling, deploying, and conducting extensive tests on fog and edge computing environments. Fogify allows users to construct complex fog topologies with varied resources and quality of service (QoS) parameters, implement containerized configurations in cloud-based or local infrastructures, and perform dynamic "what-if" scenarios that introduce faults and adjustments in real-time to evaluate potential service limitations prior to public deployment. While Fogify is unconventional as a testing tool, enabling the injection of malicious inputs to verify the robustness of IoT devices in a network, it does not adhere to standardized testing protocols nor does it support testing initiated by the IoT devices themselves, which sets it apart from solutions like CLOCIS.

In this paper, we compare the types of devices, scenarios, and environments supported by the proposed testing system CLOCIS with other IoT testing tools reviewed in this section (refer to Table 1). This comparison will help to clarify the IoT device testing that CLOCIS aims to achieve. As previously described, CLOCIS provides standardized IoT device testing, and in particular, is designed to enable automated remote testing of a large number of IoT devices, especially constrained IoT devices with limited memory and computing power, by using a message-triggering approach delivered by the testing system. This functionality is not offered by previously developed IoT testing tools. Similarly, CLOCIS does not support all the features provided by other testing tools. For instance, it does not support testing packet injection in a distributed edge computing environment or real-time bug detection functions that tools like Fogify offer.

Table 1. Benchmark IoT testing tools compared with CLOCIS to highlight their respective features and capabilities.

Tools	**IoT-TaaS** [13]	**Autocon** [28]	**Fogify** [27]	**CT-IoT** [26]	**CLOCIS**
Standard-based	◯	◯	X	X	◯
Remote testing	◯	◯	X	X	◯
Platform testing	◯	◯	◯	◯	X
Constrained device	X	X	X	X	◯
Code testing	X	X	O	O	X
Environment testing	X	X	O	O	X

In our previous work [13], we designed a service-oriented automated framework specifically for IoT testing, aiming to overcome the typical challenges of coordination, costs, and scalability encountered in traditional software testing, especially for IoT devices developed as per established standards. Our designed framework includes modules for remote distributed interoperability testing, scalable automated conformance testing, and semantic validation testing, all of which are well suited for evaluating IoT devices. This motivated us to implement the designed testing system.

Subsequently, we defined the necessary features for the envisioned testing system as requirements, which led to the creation of Autocon-IoT, a system capable of remotely testing IoT platforms [28]. Autocon-IoT incorporates features for the automated testing of IoT platforms and suggests a method of testing based on standardized messaging. Despite this, there were constraints on testing vast numbers of IoT devices. To address these challenges, this paper introduces a testing solution for functions that must be initiated by IoT devices with substantial memory and processing limitations.

Table 2 exhibits various comparisons among IoT testing tools and shows features that CLOCIS can cover. For this comparison, in addition to the testing tools mentioned in the literature review, we also examined a commercial IoT testing tool, i.e., TTworkbench [29], that utilizes TTCN-3, the formal test description language employed by CLOCIS.

Table 2. A comparative analysis of various IoT testing tools to demonstrate the range of testing coverage and environments. This analysis highlights the specific testing targets of CLOCIS.

Tools	Environment	Testing Target	Scalability	Comparison with CLOCIS
Fogify [27]	Perform testing on modeling, deploying, and conducting extensive tests on fog and edge computing environments.	It focuses on multiple IoT nodes within edge and fog layers, injecting fault packets into the testing environment.	It evaluates IoT devices operating in edge and fog network environments, limiting testing to devices connected to these specific nodes.	It enables testing in distributed edge and fog environments, primarily focusing on IoT devices in these settings. However, it fails to cover cases initiated by the target IoT device.
CT-IoT [26]	It is engineered to autonomously determine paths for effective IoT testing.	Its goal is to test IoT devices with faults within a shared network.	It has the capability to test all IoT devices on a common network.	The tool identifies IoT devices in need of testing, yet it is restricted to testing functions that must be initiated by the device under test.
TTwork-bench [29]	Commonly utilized in testing laboratories, it operates within a pre-established testing environment.	The tool is versatile, supporting a range of testing devices, from IoT gadgets to vehicles.	Utilizing a standardized formal testing description language (such as TTCN-3), it enables scalable testing.	While this tool also employs TTCN-3, mirroring CLOCIS's fundamental testing approach, it lacks an upper tester feature and thus is unable to accommodate constrained IoT devices.
Continuous testing [24]	It establishes test environments conducive to continuous testing.	This method primarily targets IoT devices operating in the field that have undergone minor modifications due to patch applications.	With the recommended testing environments in place, this approach can conduct scalable tests.	It concentrates on testing IoT devices with minor changes; however, it cannot assess the side effects of these changes, nor can it test scenarios initiated by the devices themselves.

In summary, there has been significant research dedicated to supporting the testing of IoT within various specialized domains. However, upon analysis, it is apparent that aspects such as remote testing capabilities and comprehensive testing across various IoT standards have not been thoroughly addressed.

3. Background Technologies of Conformance Testing and Motivation

The subsequent subsections delineate the procedure and salient features of conformance testing and introduce Testing and Test Control Notation version 3 (TTCN-3) as an instrumental methodology for conducting conformance assessments.

3.1. Overview of Conformance Testing

Conformance testing enables the identification of both expected and anomalous behaviors within specific protocols. However, this form of testing does not encompass the entire system; rather, it focuses on verifying system compliance with particular scopes or requirements as delineated in the testing specifications. Moreover, conformance testing does not evaluate the system's interoperability with other systems. Specifications for conformance testing in the realms of Global System for Mobile Communications (GSM), Universal Mobile Telecommunications System (UMTS), and Voice over Internet Protocol (VoIP) have been formulated by the European Telecommunications Standards Institute (ETSI). These test specifications are crafted in accordance with the ISO/IEC 9646 conformance testing methodology, which is well established and documented [30]. References to critical elements of the ISO/IEC are made within these conformance testing specifications, and the widely adopted procedures are illustrated in Figure 1. The following items elucidate the framework and components integral to conformance testing, as well as the role of TTCN-3 in facilitating these assessments:

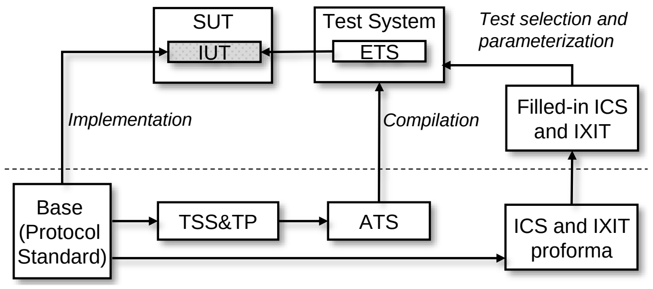


Figure 1. Generalized procedure of conformance testing [30].

- Test Suite Structure (TSS) and Test Purpose (TP): Derived from the pertinent base standards, these documents provide a clear, informal narrative of each test's objectives, emphasizing the rationale behind the test rather than the specifics of its implementation.
- Abstract Test Suite (ATS): This is the comprehensive set of test cases. Each test case delineates the precise encoding of the Test Purposes, typically formulated in a standardized test specification language, such as TTCN.
- Implementation Conformance Statement (ICS) and Implementation Extra Information for Testing (IXIT): These documents include supplementary details necessary for the execution of tests, such as specific addresses and timer values.
- Executable Test Suite (ETS): Derived from the ATS, this can be swiftly converted into an operational format using TTCN compilers, which are compatible with most contemporary test tool platforms (e.g., C++, Java).
- System Under Test (SUT) and Implementation Under Test (IUT): The SUT refers to the actual system exposed to testing protocols [31], within which resides the IUT, representing the realized versions of applications, services, or protocols.

3.2. Role of the TTCN-3

The TTCN-3, a language expressly crafted for protocol testing, has been developed and is currently maintained by ETSI. This language, which stands as a redesigned variant based on the Tree and Tabular Combined Notation (TTCN) standard (ITU-T Rec. X.292), is adept at specifying tests for reactive systems. Its applications are extensive, encompassing protocol testing, service testing, module testing, and providing application programming interfaces (APIs) for diverse platforms [32].

TTCN-3 delineates a standardized conceptual model that functions independently from the SUT, the processing platform, and the language implementation. This model features well-defined operations for each entity and interfaces to facilitate communication, management, external data exchange, and logging. These operations are enacted through the implementation and interpretation into intermediate languages.

The conceptual framework of the TTCN-3-based testing system is illustrated in Figure 2. This system executes conformance testing through the interactions among distinct entities, each equipped with specialized functions. These include facilitating communication with the SUT, managing timers, handling types, and invoking external functions. The framework also integrates interfaces such as the TTCN-3 Control Interface (TCI) and the TTCN-3 Runtime Interface (TRI). The Test Executable (TE), which is charged with running the TTCN-3 module, interacts with these interfaces to activate the actual TTCN-3 modules. The TTCN-3 conceptual testing system is composed of five key entities, with the following detailed descriptions of their functions:

- Test Management (TM): This entity serves as the managerial component of a testing system, where the test developer orchestrates the sequence of test suite executions. TTCN-3 facilitates the creation of test campaigns and allows for the customization of log formats and management.

- Component Handling (CH): As the TE can operate in a centralized or decentralized manner, the role of CH is to synchronize entities across different nodes and manage their interactions.
- Codec (CD): This entity is responsible for the encoding and decoding of messages during bidirectional exchanges with the SUT. To facilitate communication, the TTCN-3's abstract data representation must be transposed into the actual format mandated by the standards-based IUT.
- Test Logging (TL): TL manages all logging events produced by the testing system. It is imperative that the system provides a robust logging mechanism for debugging purposes.

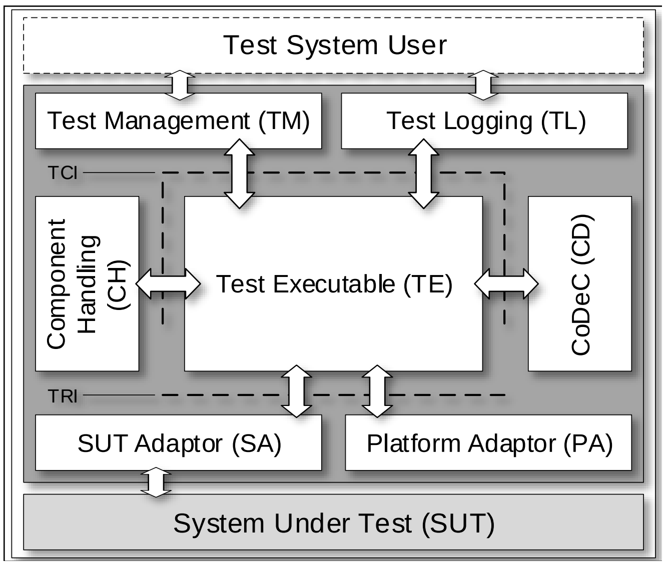


Figure 2. Conceptual testing framework using TTCN-3 as its test description language.

The following types of entities communicate with the TRI interface:

- SUT Adapter (SA): The SA handles the actual communication with the SUT, including transmitting messages according to specified procedures and receiving messages from the SUT to relay to the TE. The core functions of the SA involve bidirectional message exchange and procedural operations with the SUT. Given its role in sending and receiving messages and procedures, the SA typically exhibits concurrent and synchronous operations, necessitating individual thread management for processing inbound and outbound data.
- Platform Adapter (PA): Characterized by its support for timing functions and external functions, the PA uses timers to manage the execution flow of TTCN-3 code. External functions enhance the capabilities of TTCN-3 by employing libraries written in native programming languages like C++ or Java, thereby enabling functionalities beyond the native scope of TTCN-3.

Building on the conceptual framework of TTCN-3, we will present in the following section a testing system known as CLOCIS. This system incorporates several enhancements to address the previously mentioned challenges.

3.3. Motivation

Most software testing is conducted by sending testing inputs from an external system to the target software and observing the software's response. In the case of IoT device testing, it involves dispatching test requests from a system to the IoT device and analyzing the device's responses. However, some functionalities require the IoT device to initiate an action, and the testing system must then verify the accuracy of the messages sent by the IoT device.

For example, when an IoT device registers with a cloud server, it should transmit a registration message, which the server processes and responds to. The test system must be configured to anticipate such registration requests from the IoT device. Subsequently, the developer activates the IoT device's registration function to send the request to the test system. This process necessitates precise coordination between the testing system and the IoT device for such test cases. The IoT device often requires a separate user interface or command-line interface to enable the necessary functions, presenting challenges when testing large volumes of IoT devices.

To facilitate the testing of functions initiated by the target device, the testing system should be capable of remotely instructing the IoT device to automatically activate and execute the function, transmitting the outcome to the testing server. CLOCIS is engineered to facilitate such remote testing of IoT devices.

4. Design of Cloud-Based IoT Testing Tool

As previously discussed, traditional IoT conformance testing faces challenges with respect to scalability, cost, and configuration. Thus, there is a pressing need for more effective methods of IoT conformance testing. In this chapter, we will elucidate the architecture of CLOCIS and demonstrate how it addresses the aforementioned challenges in traditional IoT testing.

4.1. The CLOCIS Architecture

As shown in Figure 3, CLOCIS is compartmentalized into two principal logical structures: IoT testing module repository and IoT testing execution management. Conformance testing is executed through systematic interactions between components within these structures. Moreover, CLOCIS features web interfaces that facilitate remote IoT conformance testing, allowing testing professionals to conduct and review tests from afar. Below, we delineate the logical structures of CLOCIS and the components within each.

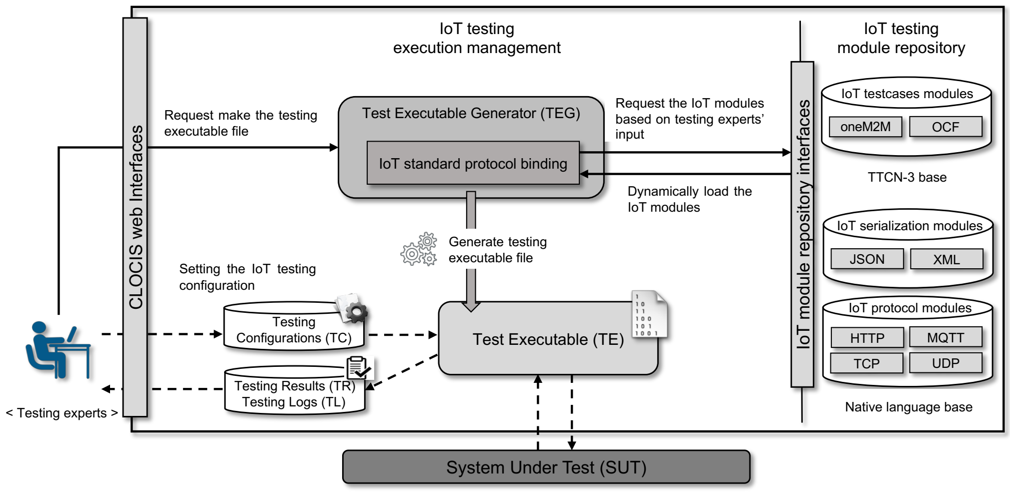

IoT testing execution management
IoT testing module repository
CLOCIS web Interfaces
Request make the testing executable file
Test Executable Generator (TEG)
IoT standard protocol binding
Request the IoT modules based on testing experts' input
Dynamically load the IoT modules
IoT module repository interfaces
IoT testcases modules
oneM2M
OCF
TTCN-3 base
IoT serialization modules
JSON
XML
IoT protocol modules
HTTP
MQTT
TCP
UDP
Native language base
Generate testing executable file
Setting the IoT testing configuration
Testing Configurations (TC)
Test Executable (TE)
Testing Results (TR) Testing Logs (TL)
< Testing experts >
System Under Test (SUT)

Figure 3. The CLOCIS conceptual architecture.

4.1.1. IoT Testing Modules Repository

This repository houses the IoT test cases and protocol modules necessary to compile the Test Executable (TE): the operational element for conformance testing, akin to the Executable Test Suite (ETS) within the testing methodology framework. TTCN-3 serves as a testing notation language, devoid of inherent communication functions. To address this, the defined TTCN-3 constructs must be amalgamated with programming languages—such as C, Java, or C++—that provide the requisite functionality for actual testing, as outlined in the Platform Adapter section (refer to Section 3.2). This integration facilitates communication with the SUT while accommodating various protocols and standards.

In CLOCIS, the testing environment is dynamically assembled via a modular approach to accommodate the functions necessary for SUT interaction. IoT standard certification authorities offering certification programs can thus readily furnish a diverse array of IoT standards to testers through new testing modules. However, utilizing these modules necessitates an additional step known as IoT protocol binding—the process of amalgamating IoT modules—which will be further explained below.

4.1.2. IoT Testing Execution Management

This component manages testing configurations and outcomes, and generates the conformance testing execution file based on input from testing professionals. The testing configuration (TC) retains files specifying test parameters defined by the testers, such as the IP addresses of devices or platforms, IoT protocols, and serialization formats.

The Testing Result (TR) and Testing Log (TL) components store the outcomes and logs of conformance tests. Test case verdicts are categorized as follows: “Pass” when the SUT behaves as expected, “Fail” when it does not, “Inconclusive” when a definitive result cannot be ascertained, and “Error” when there are faults within the testing system [33]. The logs capture message exchanges between the testing system and the SUT, providing testers with insight into the compliance of IoT platforms or devices with standards. Should the results be unsatisfactory, testers can consult the logs to pinpoint and rectify the issues.

To alleviate the labor-intensive nature of drafting new test configurations, TC offers an environment for saving and retrieving previous settings, enabling testers to conduct tests more efficiently without repeatedly reconstructing the testing environment. Both test results and logs can be archived within CLOCIS, allowing testers to reference past tests via the TR and TL.

In the pivotal process facilitated by IoT testing execution management, testers can submit TE generation data to CLOCIS for the creation of the TE. The Test Executable Generator (TEG) evaluates the IoT standards to be tested, retrieves the relevant modules—including test cases, serialization modules, and protocol modules—and integrates them. CLOCIS can also invoke native language libraries from the IoT module repository during the TE compilation process, resulting in the creation of an integrated TE through the inclusion of pertinent IoT modules.

4.2. Procedures for IoT Conformance Testing Using CLOCIS

The IoT conformance testing methodology utilizing CLOCIS is delineated into two distinct phases. Initially, the process entails generating an executable file, termed Test Executable (TE), leveraging the modules within the IoT module repository. Subsequently, testing specialists compose IoT testing configuration files and transmit these to the TE to execute the conformance testing.

During the phase of executable file generation, the paramount procedure is IoT protocol binding. While TTCN-3 inherently supports encoding methods such as Abstract Syntax Notation One (ASN.1), Basic Encoding Rules (BER), and Packed Encoding Rules (PER), it is necessary to accommodate various protocols. Thus, plugin modules are provisioned to facilitate common IoT protocols such as HTTP, MQTT, and CoAP. Furthermore, standard IoT test cases, which elucidate the requisite behaviors for assessment, must be conjoined with corresponding IoT protocols to facilitate the exchange of intelligible messages between CLOCIS and the SUT.

The port type within TTCN-3, designated to manage communication with the SUT, delineates permissible message types during testing. This port type is charged with the transmission and reception of messages. Nonetheless, if a protocol that has not undergone binding is dispatched to the SUT, the message will remain indecipherable. Therefore, messages should be conveyed subsequent to the IoT protocol binding procedures. Since standard port behavior is typically developed by IoT standards organizations and disseminated as certified testing cases, a novel methodology for IoT protocol binding is imperative, particularly as testing system developers are not authorized to alter testing cases.

As shown in Figure 4, the "Port with Translation Capability", an augmented TTCN-3 construct, enables the conversion of one message form to another during the process of sending or receiving messages. Despite the definition of testing behavior upon a single structured data type set, it is applicable in scenarios wherein the actual communication involves disparate structured data types, for instance, in cases where the SUT operates on a different layer of the protocol stack compared to the testing system. Concretely, oneM2M standard (service layer) primitives can be transmitted to HTTP-based (application layer) oneM2M IoT platforms or devices following the oneM2M HTTP protocol binding encoding scheme [34].

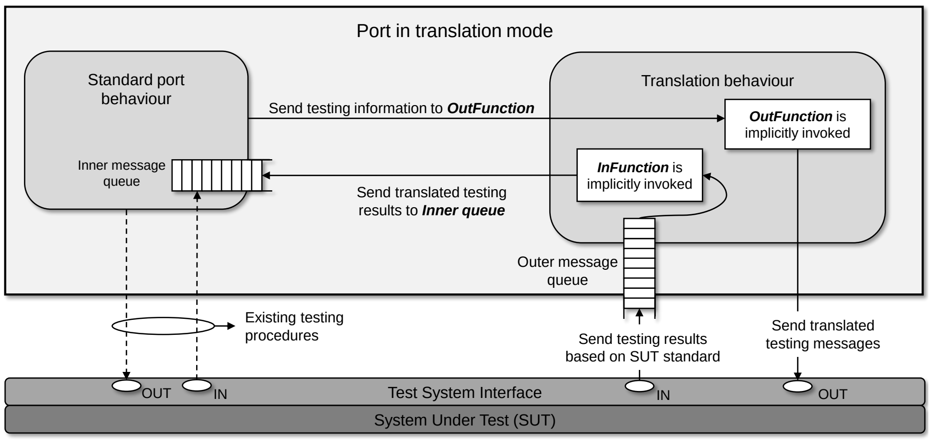

Port in translation mode
Standard port behaviour
Inner message queue
Send testing information to *OutFunction*
Translation behaviour
OutFunction is implicitly invoked
InFunction is implicitly invoked
Send translated testing results to *Inner queue*
Outer message queue
Existing testing procedures
Send testing results based on SUT standard
Send translated testing messages
OUT
IN
Test System Interface
IN
OUT
System Under Test (SUT)

Figure 4. Port with translation capability [35].

In this framework, the port type adhering to IoT standards conveys the requisite testing information to the OutFunction, which is then employed to transform standard behaviors into formats comprehensible to the SUT. Subsequently, the IoT protocol binding process is executed within the OutFunction as per the specific protocol binding specification of an IoT standard. Upon completion of the IoT protocol binding procedures, the reformatted messages are dispatched to the SUT. Post-testing, the results derived from the SUT standards are communicated to the InFunction, which is tasked with retranslating SUT-understandable behaviors into standardized behaviors; the messages are then converted and conveyed in a manner intelligible to the standard port behavior within that function.

At the conformance testing juncture, CLOCIS supports application programming interfaces (APIs) to interface with users. Utilizing these APIs, testing experts can submit the testing configuration files containing the parameters for the conformance assessment. Thereafter, these configurations are integrated into the TE, and the test cases are executed. The outcomes and logs are cataloged within the TR and TL and are made accessible to the testing professionals via the web interface. These results afford the testing experts the means to ascertain if the device or platform has been meticulously developed in accordance with IoT standards.

By employing the aforementioned functionalities, testing experts can remotely assess their services, whilst testing laboratories are empowered to augment their IoT testing repertoire through the expansion of the IoT standard module. Nonetheless, given that only the platform can currently be tested within this architecture, an ancillary structure is required to facilitate the testing of IoT devices.

4.3. Methodology for Testing IoT Devices

The methodology delineated herein primarily addresses the conformance testing of IoT platforms and, as such, is not inherently designed for the direct testing of IoT devices. To elucidate, within this framework, CLOCIS functions as a client, transmitting messages based on test cases to the IoT platform operating as a server, thereby determining conformance. Consequently, the testing paradigm for IoT devices must be structured such that CLOCIS assumes the role of a server and the IoT devices act as clients.

Typically, IoT device testing commences subsequent to the activation of the testing system. A testing expert may then directly initiate a test case and manually input specific messages into the testing system to ascertain device suitability. The primary challenge, however, is that testing experts are often required to manually execute an extensive series of test cases pertaining to IoT device functions. During this process, the potential for human error is heightened, particularly when inadvertent test cases are conducted. To mitigate such issues, the triggering message (TM) and upper tester (UT) have been devised to automate the process, obviating the need for human intervention.

- Triggering message (TM): This is a communicative protocol between the testing system and the upper tester (UT), essential for the execution of specific tests by the testing system. It encapsulates critical testing information, such as the test case name, protocol details, serialization information, and the requisite data that the IoT device's body should encompass before transmission to the testing system. Upon receipt of the TM by the UT, it ascertains the necessary testing parameters based on the provided data and accordingly instructs the device.
- Upper tester (UT): The UT is responsible for receiving the TM from the testing system and subsequently relaying commands to the device. While TMs are standardized for the exchange of information between the testing system and the UT, the UT's analysis of the TM and the execution of device-specific functions can be uniquely implemented by the device manufacturer.

Employing the aforementioned functionalities facilitates automated conformance testing for IoT devices, which proceeds as follows (see Figure 5): (1) The testing expert inputs the test specifics into the test generator (TG) of CLOCIS and initiates a test case for a designated device. (2) The testing system dispatches a TM, which contains pertinent testing details, such as the test case name, serialization data, protocol type, and the testing system's address and port information. If the test cases relate to the POST or PUT methods, the TM also includes data detailing the content to be sent in the body format. (3) Upon receiving the TM, the UT analyzes the content, determines the required testing actions, and executes the appropriate function on the Implementation Under Test (IUT), subsequently conveying the results back to the testing system. (4) Ultimately, the device's response to the function under verification allows the testing system to render a verdict—such as pass or fail—and thereby evaluate the device's compliance with the standard.

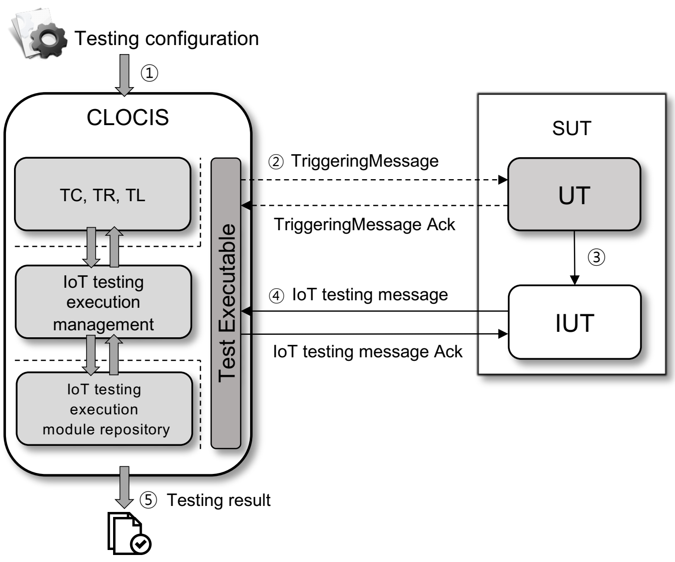


Figure 5. Procedure of the IoT device conformance testing.

Therefore, the CLOCIS architecture not only enables remote conformance testing for various protocols and standards but also supports the automated conformance testing of IoT devices. The ensuing section will explore the implementation of the CLOCIS architecture.

4.4. Triggering Message Handling

The dissemination of the triggering message, as depicted in the second step of Figure 5, along with its corresponding acknowledgment are pivotal to the conformance testing of IoT devices facilitated by CLOCIS. This necessitates a precise definition of both the triggering message and acknowledgment process. Consequently, this section meticulously explores the composition of the triggering message and the methodology for interpreting and generating an acknowledgment.

The upper tester (UT) utilizes the triggering message for the conveyance of control commands between the test system and the upper tester application. Each control command incorporates vital parameters specific to a given test case. The message type for triggering the upper tester is aligned with distinct formats for data exchange, and these formats adhere to the specifications of the TTCN-3 primitive.

CLOCIS, as the testing apparatus, employs the UtTrigger primitive to dispatch a control command. Conversely, the UT leverages the UtTriggerAck primitive to signal the receipt and successful processing of the command back to CLOCIS. CLOCIS initializes the UtTrigger primitive to direct the triggering message to the target Implementation Under Test (IUT). Upon successful receipt, the IUT uses the UtTriggerAck primitive to acknowledge CLOCIS. Subsequently, the IUT engages with the test system utilizing oneM2M request and response primitives aligned with the predefined test cases that correspond to the issued control command.

Algorithm 1 presents a method for parsing a trigger message received via an HTTP POST request. This process is central to the functioning of CLOCIS. The procedure parseTriggerMessagetakes two inputs: a request (req) and a response object (res). The algorithm begins by checking the validity of the incoming request. If the request is not valid, the system sends a “BAD_Request” response. If the request is valid, the algorithm proceeds to extract several key pieces of information from the request:

- to: the target resource in the testing system (TS) that the operation should be directed to.
- op: the type of operation to be performed.
- ty: the type of resource, if specified in the request.
- pc: the content of the primitive operation, if specified in the request.

If the UT and the SUT reside in the same application entity (AE), then the AE is instructed to generate an operation based on the extracted information (to, op, ty, pc). If the UT and SUT do not reside in the same AE, the system establishes a Secure Shell (SSH) connection to execute the necessary commands remotely. After the command execution, the system sends an “OK_Request” response, indicating the successful processing of the trigger message. This algorithm is designed to allow remote IoT device testing by triggering devices to perform specific operations autonomously. It is a key component in enabling automated testing processes within an IoT framework.

Consider a scenario where the control command’s objective is to assess “Test System prompts IUT to execute a test case for the creation of an `<AE>` with a ’labels’ attribute within a CSEBase resource”; subsequent to the exchange of the relevant triggering message and acknowledgment, the test system dispatches a message to create the <AE> resource.

Algorithm 1 Parsing Algorithm of TriggerMessage

```
Require: req is the HTTP POST request
Require: res is an object to send response
 1: AE is an application entity Object
 2: ssh for SSH connection
 3: procedure PARSETRIGGERMESSAGE(req, res)
 4:     if req is not valid then
 5:         res.sendResponse(BAD_Request)
 6:     else
 7:         to ← req.get_To()                          ▷ Target Resource in TS
 8:         op ← req.get_Operation()                   ▷ Operation Type
 9:         if ty exist in req then
10:             ty ← req.get_Ty()                      ▷ Resource Type
11:         end if
12:         if pc exist in req then
13:             pc ← req.get_Pc()                      ▷ PrimitiveContent
14:         end if
15:         if UT-SUT reside in AE then
16:             AE.generateOperation(to, op, ty, pc)
17:         else
18:             SSH ← ssh.connect()
19:             command ← createCommand()
20:             SSH.execCommand(command)
21:         end if
22:         res.sendResponse(OK_Request)
23:     end if
24: end procedure
```

The following outlines the criteria for defining UtTrigger and UtTriggerAck primitives:

1. The UtTrigger primitive is articulated as a requestPrimitive serialized in JSON format.
2. The UtTrigger includes parameters such as the following:
    - operation: (mandatory) the operation type that the IUT is incited to execute.
    - resourceType: (optional) the resource type of the target resource against which the IUT is prompted.
    - to: (mandatory) the target resource against which the operation is to be performed.
    - primitiveContent: (optional) encapsulates the resource attributes to be included in the requestPrimitive.

Additionally, we provide the response status code (RSC) definitions for the UtTriggerAck primitive as follows. A response status code of OK (value 2000) indicates that the System Under Test (SUT) has successfully received the triggering message from the test system. A response status code of BAD_REQUEST (value 4000) signifies that the SUT has misinterpreted the UtTrigger primitive. Given that the triggering message is integral to controlling IoT devices, only the aforementioned response status codes are permissible within the UtTriggerAck primitive.

For the exchange of triggering messages, the Hypertext Transfer Protocol (HTTP) is designated as the protocol for ongoing communication between the TS and upper tester application due to its extensive support among IoT devices and advantageous attributes such as persistent connection, programming simplicity, and versatility.

Control commands embedded within the HTTP request body must be serialized into JavaScript Object Notation (JSON), as it is exceptionally lightweight and facilitates effortless parsing and generation by machines.

In alignment with these specifications, Table 3 exemplifies the exchange of triggering messages between CLOCIS and the IUT. It delineates a scenario wherein the test aim is to evaluate "Test System prompts the IUT to execute a test case for the deletion of an <AE>

resource." The triggering message and its acknowledgment are documented in the first and second rows, respectively. As previously articulated, any response to a triggering command solely comprises a response status code, which, for triggering operations, is strictly confined to either 2000 (OK) or 4000 (BAD_REQUEST), in accordance with operational protocols.

Table 3. UtTrigger and UtTriggerAck primitives.

Primitive	Triggering Message

Ut Trigger	`Request{`
	 `"m2m:rqp":{`
	 `"op": 4, //indicate DELETE operation`
	 `"to": {TARGET_AE_RESOURCE_ADDRESS}`
	 `//indicate Target AE resource addr.`
	 `}`
	 `}`

Ut Trigger Ack	`Response{`
	 `"m2m:rsp": {`
	 `"rsc": 2000`
	 `}`
	 `}`

5. Implementation and Evaluation

This section outlines the implementation of the CLOCIS and provides practical examples through the testing of IoT device conformance. Specifically, we apply the oneM2M IoT standard, which is recognized as a widely utilized benchmark in the IoT domain.

As previously discussed, TTCN-3 serves as a language designed to describe the desired testing behaviors. A testing framework that can compile TTCN-3 code into an executable format and facilitate communication with the System Under Test (SUT) is indispensable. Eclipse Titan is an open-source conformance testing framework that adheres to the TTCN-3 conceptual model, offering compilation and execution capabilities within the TTCN-3 environment. Within Eclipse Titan, constructs known as "Test Ports (TPs)" and "Protocol Modules" are defined. TPs are developed to handle the transmission and reception of actual messages, supporting essential IoT protocols such as HTTP, MQTT, and CoAP through C++ plugins. The protocol modules facilitate the straightforward encoding and decoding of messages in formats like JSON or XML. Given that Eclipse Titan embodies the functionality required by CLOCIS, it has been selected as the foundation for CLOCIS's development. However, Eclipse Titan only supports the basic IP protocol stack and lacks cloud-based testing functionality. Therefore, IoT standard binding procedures specific to CLOCIS must be comprehended, and capabilities for cloud-based IoT conformance testing need to be integrated into Eclipse Titan.

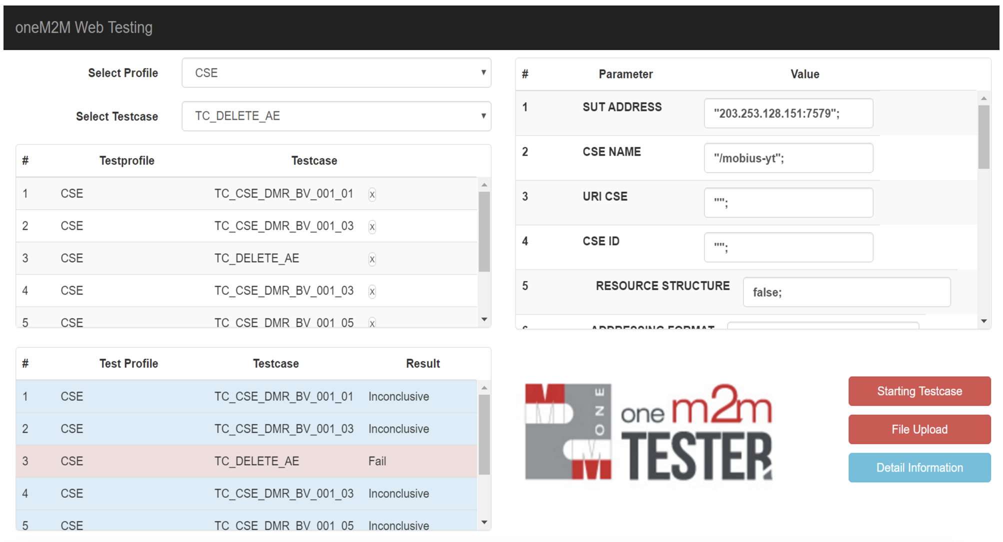
For our evaluation, we conducted conformance testing with an open-source IoT service layer platform (i.e., Mobius). Mobius provides a standardized framework for building and integrating various IoT services. It implements the oneM2M standard, which ensures interoperability between devices and services across different IoT ecosystems. The platform supports data collection, device control, and communication, enabling developers and organizations to deploy scalable and cross-domain IoT solutions efficiently. Mobius also comes with standardized IoT devices called nCUBE-thyme. nCUBE-Thyme is part of the Mobius IoT platform's ecosystem, ensuring compatibility and adherence to oneM2M standards and allowing for efficient and standardized IoT device and data management.

The nCUBE-thyme was employed as the SUT for this purpose. The nCUBE-Thyme, predicated on the oneM2M standard, is available in Node.js, Java, and Android versions.

For this evaluation, the Node.js version was utilized. oneM2M is an international IoT standardization consortium whose objective is to reduce fragmentation within the IoT service layer standards, involving eight ICT standard development organizations and approximately 200 member entities (http://onem2m.org/, accessed on 1 December 2023). Moreover, oneM2M has been actively developing testing specifications, including those for conformance and interoperability testing, with a oneM2M ATS based on TTCN-3 being developed and made accessible (https://git.onem2m.org/TST/ATS) [36,37] (accessed on 1 December 2023). Further, the oneM2M product profile specification encompasses an array of feature sets mandated for certification by the testing authority [38].

To demonstrate practical IoT device conformance testing and performance evaluation, we performed conformance testing using the CLOCIS and upper tester we developed. The testing was conducted on a computer equipped with Ubuntu 16.04 LTS OS, an Intel Core i7-7800X CPU @ 3.50GHz X2 Processor, and 8.0GB of RAM. We established a suite of 30 test cases, encompassing general capability (GEN), registration (REG), and data management and repository function (DMR) as a single test set, and executed these test sets 30 times to evaluate IoT device conformance. The execution of the IoT device conformance testing averaged 2.5 s per test set, with a standard deviation of 0.165. This demonstrates the feasibility of remotely reviewing test results via CLOCIS and provides the testing expert with an opportunity to address issues identified through the IoT device's functional test outcomes during or subsequent to the development process.

Building upon this concept, CLOCIS supports a web-based front conformance testing tool, as shown in Figure 6. This tool allows for the remote testing of IoT devices, enabling testers to conduct assessments without the need to physically visit IoT testing laboratories. Moreover, the framework's automated testing capabilities permit developers to debug their products autonomously, both throughout and subsequent to the development process. Figure 6 presents a screenshot of the web-based oneM2MTester, illustrating a historical overview of the conformance testing results for an Implementation Under Test (IUT) based on the oneM2M standard.



Select Profile: CSE

Select Testcase: TC_DELETE_AE

#	Testprofile	Testcase
1	CSE	TC_CSE_DMR_BV_001_01 x
2	CSE	TC_CSE_DMR_BV_001_03 x
3	CSE	TC_DELETE_AE x
4	CSE	TC_CSE_DMR_BV_001_03 x
5	CSE	TC_CSE_DMR_BV_001_05 x

#	Parameter	Value
1	SUT ADDRESS	"203.253.128.151:7579";
2	CSE NAME	"/mobius-yt";
3	URI CSE	"";
4	CSE ID	"";
5	RESOURCE STRUCTURE	false;

#	Test Profile	Testcase	Result
1	CSE	TC_CSE_DMR_BV_001_01	Inconclusive
2	CSE	TC_CSE_DMR_BV_001_03	Inconclusive
3	CSE	TC_DELETE_AE	Fail
4	CSE	TC_CSE_DMR_BV_001_03	Inconclusive
5	CSE	TC_CSE_DMR_BV_001_05	Inconclusive

Figure 6. CLOCIS conformance testing result screenshot capture.

6. Lessons Learned

The CLOCIS testing system is designed to assess IoT devices with limited memory and processing capabilities, which are called constrained IoT devices. Specifically, CLOCIS can prompt the devices being tested to initiate predefined actions, such as sending a registration request to the testing system. Without CLOCIS, developers would need to create an additional emulator to initiate such requests and implement extra code for testing and debugging, which consume additional memory and processing resources. This supplementary code must be removed before devices are dispatched to end users as it can present security vulnerabilities that attackers might exploit.

Due to the deployment of cloud infrastructure, CLOCIS is susceptible to network latency and reliability issues, which could affect the accuracy and timeliness of testing results. The performance may also be influenced by various environmental factors, such as internet connectivity, cloud availability, and server capacity. These challenges are common when using cloud-based testing tools. To address these issues, two strategies can be considered. The first involves uploading the emulator of the device under test to the cloud, conducting all tests there, and then allowing the results and certification reports to be downloaded. The second strategy proposes creating a lightweight version of the cloud-based testing system, offloading it as an instance to an edge node on the same network as the device under test, and performing edge-computing-based testing. Both approaches show promise for commercial testing service applications and could potentially be implemented in CLOCIS for future use as a commercial testing tool.

Currently, CLOCIS has been developed to perform testing of functionalities before the deployment of IoT devices. Once deployed in the field, IoT devices often require additions or modifications to their features, as well as patches for bugs. For CLOCIS to operate effectively in real-time systems, solutions addressing these aspects need further development. Additionally, real-time testing should be conducted without disrupting the services provided by the IoT device, which necessitates the operation of a parallel testing process within the IoT device itself. However, this requirement for an additional process poses a challenge for constrained IoT devices, which have significant limitations. Therefore, further research is needed to enable the real-time testing of IoT devices on constrained devices.

7. Future Work and Conclusions

In the contemporary landscape, the Internet of Things (IoT), in conjunction with technologies like artificial intelligence and 5G, finds applications in numerous industries and facets of daily life. The deployment of IoT services without thorough verification during development can lead to malfunctions in devices or platforms, potentially causing grave repercussions for various industries and the public. Hence, rigorous verification is imperative prior to the operation of IoT services. IoT conformance testing is one testing methodology undertaken to detect and pre-emptively resolve such issues, identifying potential operational failures at an early stage. However, traditional IoT conformance testing necessitates the physical presence of test experts in a laboratory setting, which is not always feasible. Moreover, the majority of research on IoT conformance testing does not account for the methods pertaining to IoT standards conformance, and establishing a testing environment that accommodates the heterogeneity inherent in the myriad of standards and protocols presents a significant challenge in the realm of IoT.

In this article, we address these issues by introducing the architecture and implementation methodology of CLOCIS, an adaptable tool for IoT conformance testing, and demonstrate the remote execution of IoT conformance testing using oneM2M, an international IoT standard. CLOCIS not only enables the testing of the oneM2M standard but also facilitates the incorporation and examination of various other IoT standards. This capability presents small and medium-sized enterprises (SMEs) with time and cost efficiency while also aiding standards organizations in accelerating the widespread adoption of IoT standards. As part of our future endeavors, we plan to extend support to future releases of oneM2M and other IoT standards.

Author Contributions: Writing—original draft, J.Y. and J.H.; Writing—review & editing, J.L. and S.Y.; Supervision, J.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (NRF-2023R1A2C1004453).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest

Abbreviations

The following abbreviations are used in this manuscript:

TTCN-3	Testing and Test Control Notation version 3
SUT	System Under Test
IUT	Implementation Under Test
SME	Small and medium-sized enterprise
GSM	Global System for Mobile Communications
UMTS	Universal Mobile Telecommunications System
ETSI	European Telecommunications Standards Institute
TSS	Test Suite Structure
TP	Test Purpose
ATS	Abstract Test Suite
ICS	Implementation Conformance Statement
ETS	Executable Test Suite
TCI	TTCN-3 Control Interface
TRI	TTCN-3 Runtime Interface
TEG	Test Executable Generator
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
PER	Packed Encoding Rules
TM	Triggering message
TC	Testing configuration
CH	Component Handling
TL	Test Logging
TR	Testing Result
SA	SUT Adapter
PA	Platform Adapter
UT	Upper tester

References

1. Ahlgren, B.; Hidell, M.; Ngai, E.C.H. Internet of Things for Smart Cities: Interoperability and Open Data. *IEEE Internet Comput.* **2016**, *20*, 52–56. [CrossRef]
2. Zanella, A.; Bui, N.; Castellani, A.; Vangelista, L.; Zorzi, M. Internet of things for smart cities. *IEEE Internet Things J.* **2014**, *1*, 22–32. [CrossRef]
3. Ziegler, S.; Crettaz, C.; Ladid, L.; Krco, S.; Pokric, B.; Skarmeta, A.F.; Jara, A.; Kastner, W.; Jung, M. Iot6–moving to an ipv6-based future iot. In Proceedings of the Future Internet Assembly, Dublin, Ireland, 7–9 May 2013 ; Springer: Berlin/Heidelberg, Germany, 2013; pp. 161–172.
4. Rajab, H.; Cinkelr, T. IoT based Smart Cities. In Proceedings of the 2018 International Symposium on Networks, Computers and Communications (ISNCC), Rome, Italy, 19–21 June 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–4.
5. More than 50 Billion Connected Devices. White Paper, (ERICSSON, 2013) . Available online: https://api.semanticscholar.org/CorpusID:16270177 (accessed on 6 December 2023).
6. Brady, S.; Hava, A.; Perry, P.; Murphy, J.; Magoni, D.; Portillo-Dominguez, A.O. Towards an emulated IoT test environment for anomaly detection using NEMU. In Proceedings of the 2017 Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6.
7. Ahmed, B.S.; Bures, M.; Frajtak, K.; Cerny, T. Aspects of Quality in Internet of Things (IoT) Solutions: A Systematic Mapping Study. *IEEE Access* **2019**, *7*, 13758–13780. [CrossRef]
8. Sand, B. IoT Testing-The Big Challenge Why, What and How. In Proceedings of the International Internet of Things Summit, Rome, Italy, 27–29 October 2015; Springer: Cham, Switzerland, 2015; pp. 70–76.
9. Koné, O.; Castanet, R. Test generation for interworking systems. *Comput. Commun.* **2000**, *23*, 642–652. [CrossRef]
10. Zhang, Y.; Li, Z. IPv6 conformance testing: Theory and practice. In Proceedings of the 2004 International Conference on Test, Charlotte, NC, USA, 26–28 October 2004; IEEE: Piscataway, NJ, USA, 2004; pp. 719–727.
11. Kim, E.E.; Ziegler, S. Towards an open framework of online interoperability and performance tests for the internet of things. In Proceedings of the 2017 Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6.

12. Reetz, E.S.; Kuemper, D.; Moessner, K.; Tönjes, R. How to test IoT-based services before deploying them into real world. In Proceedings of the European Wireless 2013; 19th European Wireless Conference, Guildford, UK, 16–18 April 2013; VDE-Verlag: Berlin, Germany, 2013; pp. 1–6.
13. Kim, H.; Ahmad, A.; Hwang, J.; Baqa, H.; Le Gall, F.; Ortega, M.A.R.; Song, J. IoT-TaaS: Towards a prospective IoT testing framework. *IEEE Access* **2018**, *6*, 15480–15493. [CrossRef]
14. Kanstrén, T.; Mäkelä, J.; Karhula, P. Architectures and Experiences in Testing IoT Communications. In Proceedings of the 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Västerås, Sweden, 9–13 April 2018; IEEE: Piscataway, NJ, USA, 2018.
15. Hagar, J.D. Software Test Architectures and Advanced Support Environments for IoT. In Proceedings of the 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Västerås, Sweden, 9–13 April 2018; IEEE: Piscataway, NJ, USA, 2018.
16. Taivalsaari, A.; Mikkonen, T. A Roadmap to the Programmable World: Software Challenges in the IoT Era. *IEEE Softw.* **2017**, *34*, 62–80. [CrossRef]
17. Abdallah, M.; Jaber, T.; Alabwani, N.; Alnabi, A.A. A Proposed Quality Model for the Internet of Things Systems. In Proceedings of the 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), Amman, Jordan, 9–11 April 2019; IEEE: Piscataway, NJ, USA, 2019.
18. Bures, M. Framework for Integration Testing of IoT Solutions. In Proceedings of the 2017 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 14–16 December 2017; pp. 1838–1839. [CrossRef]
19. Subahi, A.; Theodorakopoulos, G. Ensuring Compliance of IoT Devices with Their Privacy Policy Agreement. In Proceedings of the 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), Barcelona, Spain, 6–8 August 2018; IEEE: Piscataway, NJ, USA, 2018.
20. Leah Riungu-Kalliosaari, L.; Taipale, O.; Smolander, K. Testing in the Cloud: Exploring the Practice. *IEEE Softw.* **2012**, *29*, 46–51. [CrossRef]
21. Gao, J.; Bai, X.; Tsai, W.T.; Uehara, T. Testing as a Service (TaaS) on Clouds. In Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, San Francisco, CA, USA, 25–28 March 2013; pp. 212–223. [CrossRef]
22. Palattella, R.M.; Sismondi, F.; Chang, T.; Baron, L.; Vučinić, M.; Modernell, P.; Vilajosana, T.W. F-Interop Platform and Tools: Validating IoT Implementations Faster. In Proceedings of the 17th International Conference on Ad Hoc Networks and Wireless, ADHOC-NOW 2018 , Saint-Malo, France, 5–7 September 2018; IEEE: Piscataway, NJ, USA, 2018; Volume 6, pp. 15480–15493.
23. Chen, W.K.; Liu, C.H.; Liang, W.W.Y.; Tsai, M.Y. ICAT: An IoT Device Compatibility Testing Tool. In Proceedings of the 2018 25th Asia-Pacific Software Engineering Conference (APSEC), Nara, Japan, 4–7 December 2018; IEEE: Piscataway, NJ, USA, 2018.
24. Beilharz, J.; Wiesner, P.; Boockmeyer, A.; Pirl, L.; Friedenberger, D.; Brokhausen, F.; Behnke, I.; Polze, A.; Thamsen, L. Continuously Testing Distributed IoT Systems: An Overview of the State of the Art. In Proceedings of the Service-Oriented Computing—ICSOC 2021 Workshops, Dubai, United Arab Emirates, 22–25 November 2021; Springer International Publishing: Cham, Switzerland, 2022; pp. 336–350.
25. Chernyshev, M.; Baig, Z.; Bello, O.; Zeadally, S. Internet of Things (IoT): Research, Simulators, and Testbeds. *IEEE Internet Things J.* **2018**, *5*, 1637–1647. [CrossRef]
26. Hu, L.; Wong, W.; Kuhn, D.; Kacker, R.; Li, S. CT-IoT: A combinatorial testing-based path selection framework for effective IoT testing. *Empir. Softw. Eng.* **2022**, *27*, 32 . [CrossRef]
27. Symeonides, M.; Georgiou, Z.; Trihinas, D.; Pallis, G.; Dikaiakos, M.D. Fogify: A Fog Computing Emulation Framework. In Proceedings of the 2020 IEEE/ACM Symposium on Edge Computing (SEC), San Jose, CA, USA, 12–14 November 2020; pp. 42–54. [CrossRef]
28. Hwang, J.; Aziz, A.; Sung, N.; Ahmad, A.; Le Gall, F.; Song, J. AUTOCON-IoT: Automated and Scalable Online Conformance Testing for IoT Applications. *IEEE Access* **2020**, *8*, 43111–43121. [CrossRef]
29. TTworkbench Test Automation Platform . Available online: https://www.spirent.com/products/test-automation-platform-ttworkbench (accessed on 5 December 2023).
30. Moseley, S.; Randall, S.; Wiles, A. Experience within ETSI of the combined roles of conformance testing and interoperability testing. In Proceedings of the 33rd European Solid-State Device Research—ESSDERC ’03 (IEEE Cat. No. 03EX704), Delft, The Netherlands, 22–24 October 2003; IEEE: Piscataway, NJ, USA, 2003; pp. 177–189.
31. European Telecommunications Standards Institute (ETSI). *Methods for Testing and Specification (MTS): Deployment of Model-Based Automated Testing Infrastructure in a Cloud*; European Telecommunications Standards Institute (ETSI): Sophia Antipolis, France, 2016.
32. Grabowski, J.; Hogrefe, D.; Réthy, G.; Schieferdecker, I.; Wiles, A.; Willcock, C. An introduction to the testing and test control notation (TTCN-3). *Comput. Netw.* **2003**, *42*, 375–403. [CrossRef]
33. Muhammad, F. *An Introduction to Umts Technology: Testing, Specifications and Standard Bodies for Engineers and Managers*; Universal-Publishers: Irvine, CA, USA, 2008.
34. oneM2M. *oneM2M-TS-0009: HTTP Protocol Binding*; Release 4, V4.5.0; 2023.
35. Etsi, E. *ETSI ES 201 873-1: Methods for Testing and Specification (MTS); the Testing and Test Control Notation Version 3 Part 1: TTCN-3 Core Language*; V4.15.1; 2023.
36. Swetina, J.; Lu, G.; Jacobs, P.; Ennesser, F.; Song, J. Toward a standardized common M2M service layer platform: Introduction to oneM2M. *IEEE Wirel. Commun.* **2014**, *21*, 20–26. [CrossRef]

37. Hwang, J.; An, J.; Aziz, A.; Kim, J.; Jeong, S.; Song, J. Interworking Models of Smart City with Heterogeneous Internet of Things Standards. *IEEE Commun. Mag.* **2019**, *57*, 74–79. [CrossRef]
38. oneM2M. *oneM2M-TS-0025: Product Profiles*; Release 3, V3.1.0; 2019.