A Passive Interoperability Testing Approach Applied to the Constrained Application Protocol (CoAP)

Nanxing Chen, César Viho

IRISA/University of Rennes 1
Campus de Beaulieu, 263 avenue du Général Leclerc
35042, Rennes, France
{nanxing.chen, cesar.viho}@irisa.fr,
http://www.irisa.fr/

Abstract. Constrained Application Protocol (CoAP) is an application protocol designed for the Internet of Things, where smart devices cooperate to provide machine-to-machine Web services. In this context, a high level of interoperability is crucial. This paper addresses the interoperability testing of CoAP applications. It proposes a methodology based on passive testing, which is a technique to test a running system by only observing its behavior without introducing any test input. The methodology (the proposed method and a corresponding testing tool) was put into operation during the CoAP interoperability testing event (Plugtest) organized by ETSI in Paris in March 2012, where a number of CoAP applications were successfully tested, showing the validity and efficiency of this approach.

1 Introduction

Internet of Things (IoT) is an integrated part of future Internet and could be defined as a dynamic global network infrastructure with self configuring capabilities based on standard and interoperable communication protocols where physical and virtual things use intelligent interfaces, and are seamlessly integrated into the information networks. One of the objectives of the IoT is using the captured information by smart objects (e.g. automation systems, mobile personal gadgets, building-automation devices, cellular terminals, the smart grid, etc.) to improve peoples life in a large range of fields: healthcare, environment monitoring, smart energy control, industrial automation and manufacturing, logistics, etc. Promoted by IoT, more and more devices are becoming connected and benefit from interacting with each other to achieve cooperative services. Over the next decade, this could grow to trillions of embedded devices and will greatly increase the Internet's size and scope. On the other hand, the evolution of technologies also brings challenges: devices behind Machine-to-Machine (M2M) applications are generally have limited resources. Typically, they are battery-powered and frequently asleep, limiting them to an average consumption on the order of micro-watts. Power limitations also lead to constraints on available networking. Most devices connect wirelessly as stringing wires are prohibitively expensive and sometimes not applicable. In consequence, packet losses might occur during data transfer.

To deal with these challenging issues, the IETF Constrained RESTful Environments (CoRE) working group [1] has worked out the Constrained Application Protocol (CoAP), an application layer protocol to provide resource constrained devices with low overhead and low power consumption Web service functionalities. Different from traditional Web services protocol, CoAP protocol involves new performance engineering methods, tools, and benchmarking needs. Especially, the ubiquitous nature of CoAP requires interoperability to ensure that smart objects using CoAP work well together in low-power and lossy environment without human intervention, while guaranteeing the services described in the specifications. Testing interoperability of CoAP protocol in this context is challenging as:

- We have to deal with a huge number of different kinds of components; In the context of the IoT, billions of objects will communicate using this CoAP protocol.
- We may not have possibility or the capability to perturb the normal behavior of the components to be tested composing the SUT (System Under Test). As a consequence, the classical active interoperability testing (where stimuli are sent to the SUT) may not be applicable.
- Contrary to usual case of interoperability testing where we make the hypothesis that the communication environment is reliable, the constrained environment of CoAP requires to consider packet losses.

In this paper, an interoperability testing method based on the technique of passive testing is proposed. It involves defining a set of test cases and verifying their validity on the observed behavior (traces) of the CoAP implementations (Clients and Servers) without injecting any test message. In addition, a trace verification algorithm is implemented in a test tool to automate trace analysis, including identifying the occurrence of test cases and assigning appropriate verdicts. The proposed method and the associated trace validation tool were put into operation in CoAP Plugtest - the first formal CoAP interoperability event held in Paris, March 2012, during which a large number of tests were successfully performed.

This paper addresses the interoperability testing of CoAP protocol applications. It is organized as follows: section 2 presents the generality of CoAP as well as the needs of CoAP interoperability testing. Section 3 gives an overview of the proposed method for CoAP interoperability testing. In section 4, the methodology is elaborated. The application of the methodology and the experimental results are presented in section 5. Finally, section 6 gives the conclusions and the perspectives.

2 Background and Motivation

2.1 CoAP Protocol Overview

Most Internet applications today depend on the Web architecture, using HTTP [2] to access information and perform updates. HTTP is based on Representational State Transfer (REST) (Fielding, 2000). It is an architectural style that makes information available as resources are identified by URIs (Uniform Resource Identifier): applications communicate by exchanging representations of these resources by using a limited set of methods. This paradigm is quickly becoming popular, even spreading to Internet of Things (IoT) applications, aiming at extending

1. http://datatracker.ietf.org/wg/core/charter/
2. http://www.w3.org/Protocols/

the Web to constrained nodes and networks. In this context, the IETF Constrained Application Protocol (CoAP) has been designed, which is an application-layer protocol on keeping in mind the various issues of constrained environment to realize interoperations with constrained networks and nodes. CoAP adopts some HTTP patterns such as resource abstraction, URIs, RESTful interaction and extensible header options, but with a lower cost in terms of bandwidth and implementation complexity. Unlike HTTP over TCP, CoAP operates over UDP, with reliable unicast and multicast support (cf. FIG 1). Two layers compose CoAP that have to be considered as a whole when dealing with CoAP interoperability testing:

- the CoAP *transaction layer* is used to deal with UDP and the asynchronous nature of the interactions. Within UDP packets, CoAP uses a four-byte binary header, followed by a sequence of options. Four types of messages are defined, which provide CoAP with a reliability mechanism. Confirmable messages (CON) require acknowledgment while Non-Confirmable messages (NON) do not require acknowledgment. Acknowledgment messages (ACK) are acknowledgments to a CON message and Reset messages (RST) indicate that a CON message was received, but some context is missing to properly process it; Eg. the node has rebooted.
- On top of CoAP's transaction layer, CoAP *Request/Response layer* is responsible for the transmission of requests and responses for resource manipulation and interoperation. The familiar HTTP request methods are supported: *GET* retrieves the resource identified by the request URI. *POST* requests the server to update/create a new resource under the requested URI. *PUT* requests that the resource identified by the request URI to be updated with the enclosed message body. *DELETE* requests that the resource identified by the requested URI to be deleted.

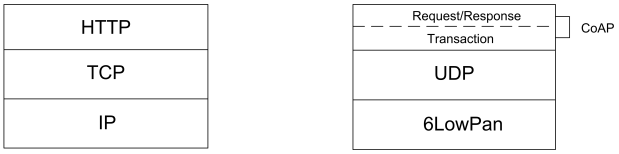


FIG. 1 – *Protocol stacks of HTTP and CoAP.*

CoAP supports built-in resource discovery that allows discovering and advertising the resources offered by a device. A subscription option is provided for client to request a notification whenever a resource changes. This is then accomplished by the device with the resource of interest by sending the response messages with the latest change to the subscribers. CoAP supports block wise transfer. Basic CoAP messages work well for the small payloads such as data from temperature sensors, light switches, etc. Occasionally, applications need to transfer larger payloads – for instance, for firmware updates. Instead of relying on IP fragmentation, CoAP is equipped with *Block* options to support the transmission of large data by splitting the data into blocks. Besides, CoAP also have other features like best-effort multicast, cachability, HTTP mapping, etc. These characteristics of CoAP provide a flexible and versatile application framework. Although CoAP is still a work in progress, many famous embedded operating systems, e.g. Tiny OS [3] and Contiki [4], have already released their CoAP implementations. It

3. http://www.tinyos.net/
4. http://www.sics.se/contiki/

is slated to become one of the most important ubiquitous application protocols for the future Internet of Things. Thus, providing efficient testing methods with associated tools are required to ensure interoperability of billions of objects (implementing CoAP) that will be widely deployed.

2.2 The Needs of CoAP Interoperability Testing

To ensure that protocol applications interoperate correctly and provide expected services, several testing methods such as *conformance testing* (Rayner, 1987) and *interoperability testing* (Lee et al., 1997), (Zaidi et al., 2009), (Viho et al., 2001) etc. have been developed. Conformance testing checks whether an implementation is correct with respect to its relevant specifications. It allows protocol designers to focus on the fundamental problems of their products and it is considered as an important step to assure interoperability. However, it is widely agreed that conformance testing has limitations in ensuring interoperability. In fact, even following the same standard, two different devices might not be interoperable. A variety of reasons account for non-interoperability (see FIG 2): *(i)* In the standards: frequently, standards provide choices to implementers, and different choices can be incompatible so that they fail in interoperation. Moreover, unintentional ambiguities may exist in standards leading also to non-interoperability. *(ii)* During implementations: One may observe programming errors and different interpretations of the standard or different choices of options allowed by the standard. Also, sometimes a new service will enter the market before any standard exists for it, and the protocols involved may become informal. *(iii)* Incompleteness of conformance testing: conformance test suites may not be complete so that they do not guarantee conformance to protocol standards. *(iv)* Besides technical reasons, there are also business reasons that can account for non-interoperability. e.g., businesses sometimes view features that defeat interoperation as a competitive advantage. Nevertheless, on one hand, customer needs are growing and manufac-

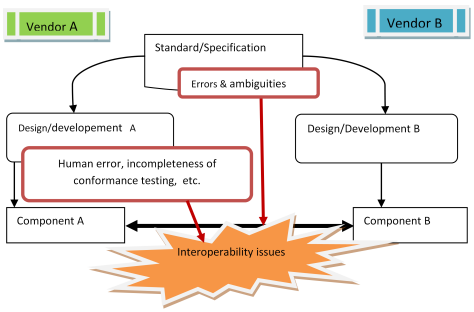


FIG. 2 – *Non interoperability issues.*

turers permanently develop new equipments with improved quality of service. On the other hand, with the rapid widespread commercial adoption of complex and diverse IoT technologies, interoperability is essential for M2M applications to provide cooperative services. To deal with these issues, interoperability testing is holding a strategic position in the design of new technologies. Its role is to determine whether several interconnected products from different product lines interoperate correctly and provide the expected services. In this context,

and among other means to achieve interoperability, the standardization bodies and industry forums arrange regular workshops and interoperability testing events (e.g. CoAP Plugtest [5], IPSO Interoperability events [6], Tahi IPv6 Interoperability event [7], etc.), where vendors can test the interoperability of their equipments with other fellow industry equipments and several test suites provided by different testing experts.

As one of the most important protocol for the future Internet of Things, the number of smart objects using CoAP is expected to grow substantially, concerning M2M applications that deal with manipulation of various resources on constrained networks. For CoAP applications to be widely adopted by the industry, interoperability testing is required to ensure that CoAP implementations from different vendors work well together. Therefore, in this paper we propose a methodology for CoAP interoperability testing, including testing method and an associated testing tool, which were successfully used during the CoAP Plugtest. The contributions and originality of this work are three-fold: *(i)* Contrary to the *active testing* method used in conventional interoperability testing events, which is done by actively stimulating the implementations and verifying the outputs, we apply passive testing. It is a technique based only on observation. Its non-intrusive nature makes it appropriate for interoperability testing, especially in the context of IoT. *(ii)* As IoT implies providing services in lossy networks, we also take into account fundamental CoAP implementations interoperability testing in lossy context. *(iii)* Contrary to manual verification used in conventional interoperability testing events, the verification procedure has been automatized by a test validation tool, which increases the efficiency, reduces time and costs.

3 CoAP Passive Interoperability Testing Method Overview

This section gives a general overview of CoAP interoperability testing methodology (cf. FIG 3). Based on the specifications of CoAP (Shelby et al. (2011); Shelby (2011); Hartke (2012); Bormann and Z.Shelby (2012)), a set of interoperability test purposes are selected. Each test purpose represents a critical property of CoAP to be verified. Once the test purposes are defined, a test case is derived for each test purpose, which describes in detail the expected behavior of the CoAP implementations to be observed. These test cases are then used to process the observed behavior of CoAP applications and help to draw a conclusion of interoperability. In this work, we use the formal model *Input-Output Labeled Transition System* (Verhaard et al., 1992) (IOLTS for short in the sequel) to model the test purposes, test cases, etc. This model is also used to explain the algorithm of trace verification

3.1 Formal Model

Specification languages for reactive systems can often be described in terms of labeled transition systems. In this paper, we use the IOLTS model, which allows differentiating input, output and internal events while indicating the interfaces specified for each message.

5. http://www.etsi.org/plugtests/coap/coap.htm
6. http://www.ipso-alliance.org/category/events
7. http://www.tahi.org/inop/6thinterop.html

Definition 1 An IOLTS is a tuple $M = (Q^M, \Sigma^M, \Delta^M, q_0^M)$ where Q^M is the set of states of the system M with q_0^M its initial state. Σ^M is the set of observable events at the interfaces of M. In IOLTS model, input and output actions are differentiated: We note $p?a$ (resp. $p!a$) for an input (resp. output) a at interface p. $\Gamma(q) =_{def} \{\alpha \in \Sigma^M | \exists q', (q, \alpha, q') \in \Delta^M\}$ is the set of all possible events at the state q. $\Delta^M \subseteq Q^M \times (\Sigma^M \cup \tau) \times Q^M$ is the transition relation, where $\tau \notin \Sigma^M$ stands for an internal action. A transition in M is noted by $(q, \alpha, q') \in \Delta^M$.

3.2 Testing Method Overview

The outlines of CoAP interoperability testing methodology are illustrated in FIG 3, which consists in the following steps:

1. Regarding the specifications of CoAP (Shelby et al. (2011)), a set of interoperability test purposes (ITP) is selected. Each test purpose represents a critical property that needs to be verified and must be validated to guarantee its correctness (Schulz et al., 2007). Formally, an ITP can be represented by a deterministic and complete IOLTS equipped with trap states used to select targeted behavior. ITP = (Q^{ITP}, Σ^{ITP}, $\triangle^{ITP}$, q_0^{ITP}) where:
   - Σ^{ITP} is the set of observable events related to the test purpose.
   - Q^{ITP} is the set of states. An ITP has a set of trap states $Accept^{ITP}$, indicating the targeted behavior. States in $Accept^{ITP}$ imply that the test purpose has been reached and are only directly reachable by the observation of outputs produced by the IUTs.
   - ITP is complete, which means that each state allows all actions. This is done by inserting "$*$" label at each state q of the ITP, where "$*$ " is an abbreviation for the complement set of all other events leaving q . By using "$*$" label, ITP is able to describe a property without taking into account the complete sequence of detailed specifications interaction.
2. For each ITP, an iop test case (ITC) is generated. An ITC is the detailed set of instructions that need to be taken in order to perform the test. Specifically in passive testing, an ITC describes in detail the behavior of IUTs to be observed, which are related to the given ITP. Formally, an iop test case ITC is represented by an extended version of IOLTS called T-IOLTS for Testing-IOLTS. A T-IOLTS ITC can be defined by: ITC = $(Q^{ITC}, \Sigma^{ITC}, \Delta^{ITC}, q_0^{ITC})$, where q_0^{ITC} is the initial state. {*Pass, Fail, Inconclusive*} $\in Q^{ITC}$ are the trap states representing interoperability verdicts. Respectively, verdict *Pass* means the ITP is satisfied ($Accept^{ITP}$ is reached) without any fault detected. *Fail* means at least one fault is detected, while *Inconclusive* means the behavior of IUTs is not faulty, however can not satisfy the ITP. Σ^{ITC} denotes the observation of the messages from the interfaces. Δ^{ITC} is the transition function. In the sequel, any ITC is supposed to be deterministic. The set of test cases is called a *test suite*. An example of ITP and ITC can be found in FIG. 6.
3. Analyze the observed behavior of the IUTs. In this work, we apply the technique of passive testing for the following arguments: First, passive testing does not disturb the normal operation of the protocol implementations, thus is suitable for interoperability testing in operational environment. Also, passive testing does not introduce extra overhead into the networks, hence is appropriate in the context of IoT. During the test, packets exchanged between CoAP implementations (CoAP client and CoAP server) are captured and by a packet sniffer. Captured traces are then filtered to keep only the useful information and

analyzed against the test cases by using a passive testing tool. And a verdict *Pass, Fail,* or *Inconclusive* is issued. Respectively, verdict *Pass* means the test purpose is verified without any fault detected, *Fail* means at least one fault is detected, while *Inconclusive* means the behavior of implementations is not forbidden by the specifications, however does not correspond to the test purpose.

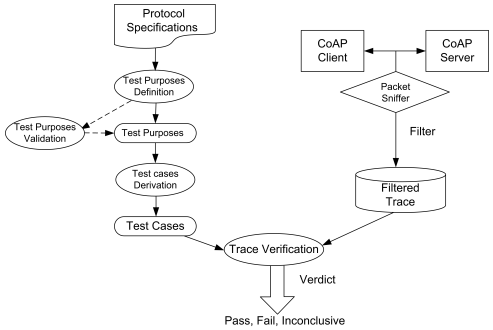


FIG. 3 – *CoAP interoperability testing method.*

4 Passive Interoperability Testing for CoAP Applications

This section describes in detail the testing procedure, including the testing architectures, each testing step and an associated validation tool.

4.1 Testing Architectures

Two test architectures have been defined for different purposes. The basic test architecture is illustrated in FIG. 4-(a). It involves a *Test System (TS)* and a *system under test (SUT)* composed of two *CoAP implementations under test (IUT)*, namely a CoAP client and a CoAP server. Since we apply the technique of passive testing, a packet sniffer is used to capture the packets (traces) exchanged between the IUTs. Moreover, as CoAP is designed for constrained networks, which imply possibility of packet losses, we also need to consider testing the interoperability of CoAP applications in lossy environment. The corresponding architecture is as FIG. 4-(b): A UDP gateway is used in-between the client and server to emulate a lossy medium (c.f. more details in Section 5)

4.2 Selection of CoAP Interoperability Test Purposes

In order to realize interoperability testing for CoAP protocol, first a set of CoAP *interoperability test purposes* (ITP) is defined to focus on the most important properties of CoAP. To ensure that the ITPs are correct w.r.t the specification. In our work, the ITPs were chosen and cross validated by experts from ETSI [8], IRISA [9] and Beijing University of Post and

8. http://www.etsi.org/WebSite/homepage.aspx
9. http://www.irisa.fr/

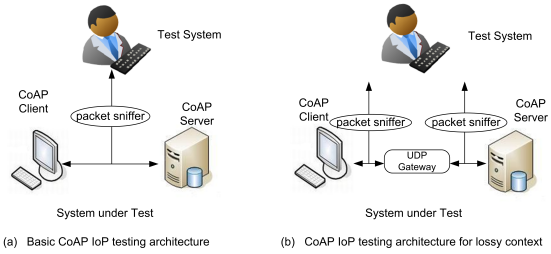


FIG. 4 – *CoAP interoperability testing architectures.*

Telecommunication [10], and reviewed by IPSO Alliance to guarantee the correctness. From the specifications, a total of 27 test purposes were chosen, which cover four most important aspects of the protocol: RESTful methods, Link Format for resource discovery, Blockwise transfer for large resources, and resource observation.

Basic CoAP CoRE Protocol Methods This group of tests contains 16 test purposes, among them 15 tests aim at testing the basic transaction of the CoAP request/response model with or without options, in both reliable and lossy environment.

The fundamental RESTful Methods interoperability tests involve verifying that both client and server interact correctly according to (Shelby et al., 2011) by using any of the RESTful methods GET, POST, PUT, and DELETE. Specifically, it requires to verify that each time the client sends a request, it contains the correct method code and correct message type code (CON or NON). Upon the reception of a request sent by CoAP client, the server sends piggybacked reply accordingly: *(i)* if the request is *confirmable*, the server must send an acknowledgment *ACK*. *(ii)* If the request is *non-confirmable*, the server also sends a *non-confirmable* reply. An example of CoAP RESTful transaction is illustrated in FIG. 5-(a). The client sends a confirmable request, asking for humidity. Upon the reception, the server acknowledges the message, transferring the payload while echoing the Message ID(0x7af2) generated by the client. Sometimes however, a server cannot obtain immediately the resource requested. In this case, it will first send an acknowledgment with an empty payload, which effectively is a promise that the request will be acted. When the server finally has obtained the resource representation, it sends the response in a confirmable mode to ensure that this message not be lost (cf. FIG. 5-(b)). This asynchronous interaction avoids that the client repeatedly retransmits the request. Based on basic transactions, three important options were also chosen to be verified. Moreover, as CoAP protocol is designed for constrained networks, where packet losses may occur, therefore an important aspect is to show that CoAP application should still interoperate correctly even in lossy context. Especially, they must correctly retransmit the request and response if they are lost.

The CoRE Link Format Resource discovery is important for M2M applications. For CoAP, Shelby (2011) standardizes a resource discovery format to discover the list of resources offered by a device, or for a device to advertise or post its resources to a directory service. In Shelby

10. http://www.bupt.edu.cn/

(2011), path prefix for resource discovery is defined as */.well-known/core*. This description is then accessed with a GET request on that URI. The interoperability of resource discovery testing of this property involves 2 tests, aiming at verifying that: when the client requests */.well-known/core* resource, the server sends a response containing the payload indicating all the available links. Also, if the client is interested in specific resources, it can filter the request using a query string. For example *GET /.well-known/core?rt=Temperature* [11] would request only resources with the name *Temperature*.

Block-wise Transfer CoAP is based on datagram transports, which limits the maximum size of resource representations that can be transferred. In order to handle large payloads, Bormann and Z.Shelby (2012) defines a mechanism *Block-wise transfer*. It supports the transmission of large data by splitting the data into blocks for sending and manages the reassembly on the application layer upon receipt in order to avoid fragmentation on the lower layers. This group of test purposes contains 4 tests that check the main functionalities of CoAP block-wise transfer. FIG. 5-(c) illustrates a block-wise transfer of a large payload humidity requested by the client. Upon the reception, the server divides the response into four blocks and transfer them separately to the client.

CoAP Observe As the representation of a resource on a server may change from time to time, Hartke (2012) defines a mechanism *CoAP Observe*, which is an asynchronous approach to support pushing information from servers to interested clients over a period of time. The interoperability testing of this property contains 5 tests to check the main functionalities of resource observation: If a client is interested in the current state of specific resource, it can register its interest in this resource by issuing a GET request with an *Observe* option to the resource. The server then keeps track of the client and sends a notification whenever the observed resource changes. If the client rejects a notification with a RST message or when it performs a GET request without an *Observe* option for a currently observed resource, the server will remove the client from the list of observers for this resource. And the client will no longer receive any updated information about the resource. If a client wants to receive notifications later, it needs to register again. An example of observation registration and cancellation can be found in FIG. 5-(d).

4.3 Test Case Derivation

For each test purpose, a test case is derived. In passive testing, each test case contains the events to be observed with respect to the given test purpose. It specifies the events that can lead to different verdicts: *(i)* the expected events to be observed that allow reaching the test purpose and consequently lead to *Pass* verdict. *(ii)* Unexpected events that can lead to *Fail* verdict. *(iii)* Other behavior allowed by the specifications but not allows to satisfy the test purpose, thus leads to *Inconclusive* verdict. The assignment of different verdicts is done by studying carefully the specifications, and validated by ETSI, IRISA, BUPT and IPSO Alliance.

An example of test case is given in FIG. 6. The test purpose (FIG. 6-(a)) focuses on the GET method in confirmable transaction mode. i.e., when the client sends a GET request (as explained in the test specification FIG. 6-(c)), the request contains parameters: a Message ID,

11. *rt*: Resource type attribute. It is a noun describing the resource.

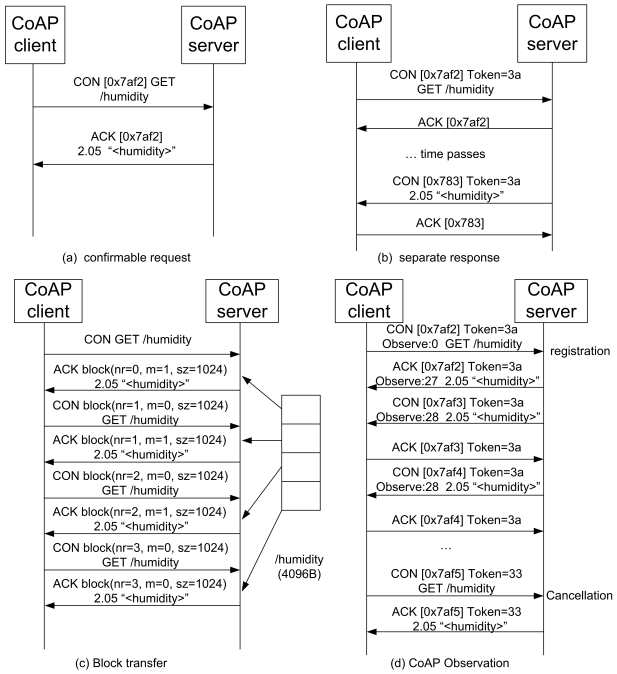
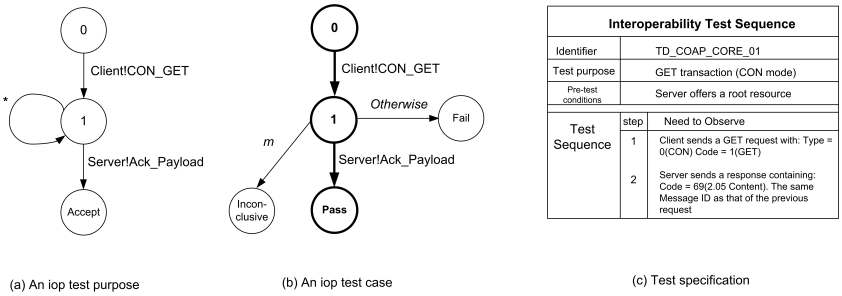


FIG. 5 – *CoAP Transaction Examples.*

Type=0 for confirmable transaction mode, Code=1 for GET method), the server's response contains an acknowledgment, echoing the same Message ID, as well as the resource presentation (Code=69(2.05 Content)). The corresponding test case is illustrated in FIG. 6-(b) The bold part of the test case represents the expected behavior that leads to *Pass* verdict. Behavior that is not forbidden by the specifications leads to *Inconclusive* verdict (for example, response contains code other than 69. These events are noted by m in the figure). However other behavior leads to *Fail* verdict (for example non-match of Message ID. These events are labeled by *otherwise*).



Interoperability Test Sequence		
Identifier	TD_COAP_CORE_01	
Test purpose	GET transaction (CON mode)	
Pre-test conditions	Server offers a root resource	
Test Sequence	step	Need to Observe
	1	Client sends a GET request with: Type = 0(CON) Code = 1(GET)
	2	Server sends a response containing: Code = 69(2.05 Content). The same Message ID as that of the previous request

(c) Test specification

FIG. 6 – *CoAP test case example.*

4.4 Trace Verification

Trace Verification Algorithm Derived test cases are used to analyze the behavior of CoAP applications. Specifically in passive testing, The packets exchanged between the CoAP client and server are captured and stored in a file. They are are key to conclude whether CoAP devices interoperate (cf. FIG. 3).

In passive testing, one issue is that the test system has no knowledge of the global state where the system under test SUT can be in w.r.t a test case at the beginning of the trace. In order to realize the trace analysis, a straight way is *trace mapping* (Lee et al., 1997). This approach compares each event in the trace produced by the SUT strictly with that in the specification. SUT specification is modeled as a Finite State Machine (FSM). Recorded trace is mapped into the FSM by backtracking. Initially, all states in the specification are the possible states that the SUT can be in. Then, the events in the trace are studied one after the other: the states which can be led to other states in the FSM by the currently checked event are replaced by their destination states of the corresponding transitions. Other states are redundant states and removed. After a number of iterations, if the set of possible states becomes empty, SUT is determined faulty. i.e., it contains a behavior which contradicts its specification as trace mapping procedure fails. This approach however, has some limitations. First, to model a complex network by a single FSM maybe complex. Moreover, this approach does not suit interoperability testing: as the SUT concerned in interoperability testing involves several IUTs, therefore to calculate their global behavior encounters state explosion.

In (Zaidi et al., 2009), another method called *invariant approach* was introduced. Each invariant represents an important property of the SUT extracted from the specification. It is composed of a preamble and a test part, which are cause-effect events respectively w.r.t the property. The invariant is then used to process the trace: The correct behavior of the SUT requires that the trace exhibit the whole invariant.

In this paper, we propose another solution to perform passive trace verification. The idea is to make use of the special interaction model of CoAP: As the interoperability testing of CoAP essentially involves verifying the correct transactions between the client and the server, therefore each test case consists of the dialogues (requests and responses) made between them, and generally starts with a request from the client. A strategy is as follows: *(i)* the recorded trace is filtered to keep only the messages that belong to CoAP protocol. In this way, the trace only contains the conversations made between the client and the server. *(ii)* Each event in the filtered trace will be checked one after another according to the following rules, which correspond to the algorithm of trace verification (c.f. Algorithm 1). This algorithm aims at mapping the test cases into the trace. i.e., to match a test case with the corresponding conversation(s) in the trace. Recall that in our work, each test case specify the events that lead to verdicts *Pass*, *Fail* or *Inconclusive* assigned on its trap states. Therefore, if a test case is identified on the trace, we can check whether it is respected by comparing each message of the test case with that in its corresponding conversation(s), and emitting a verdict once an associated verdict is reached.

1. If the currently checked message is a request sent by the client, we verify whether it corresponds to the first message of (at least one of) the test cases (noted TC_i) in the test suite TS. If it is the case, we keep track of these test cases TC_i, as the matching of messages implies that TC_i might be exhibited on the trace. We call these TC_i *candidate test cases*. The set of candidate test cases is noted TC. Specifically, the currently checked state in each candidate test case is kept in memory (noted $Current_i$).

2. If the currently checked message is a response sent by the server, we check if this response corresponds to an event of each candidate test cases TC_i at its currently checked state (memorized by $Current_i$). If it is the case, we further check if this response leads to a verdict *Pass, Fail* or *Inconclusive*. If it is the case, the corresponding verdict is emitted to the related test case. Otherwise we move to the next state of the currently checked state of TC_i, which can be reached by the transition label - the currently checked message. Then we take another message in the trace and restart from the beginning. On the contrary, if the response does not correspond to any event at the currently checked state in a candidate test case TC_i, we remove this TC_i from the set of the candidate test cases.
3. Besides, we need a counter for each test case. This is because in passive testing, a test case can be met several times during the interactions between the client and the server due to the non-controllable nature of passive testing. The counter $Counter_i$ for each test case TC_i is initially set to zero. Each time a verdict is emitted for TC_i, the counter increments by 1. Also, a verdict emitted for a candidate test case TC_i each time when it is met is recorded, noted $verdict.TC_i.Counter_i$. For example, $verdict.TC_1.1=Pass$ represents a sub-verdict attributed to test case TC_1 when it is encountered the first time in the trace. All the obtained sub-verdicts are recorded in a set $verdict.TC_i$. It helps further assign a global verdict for this test case.
4. The global verdict for each test case is emitted by taking into account all its sub-verdicts recorded in $verdict.TC_i$. Finally, a global verdict for TC_i is *Pass* if all its sub-verdicts are *Pass* *Inconclusive* if at least one sub-verdict is *Inconclusive,* but no sub-verdict is *Fail. Fail*, if at least one sub-verdict is *Fail*.

The complexity of the algorithm is $O(M \times N)$, where M is the size of the trace, N the number of candidate test cases. The trace verification procedure aims at looking for the possible test cases that might be exhibited in the trace by checking each event taken in order from the trace. Regarding the transaction mode of CoAP, each filtered traces are composed of a set of *conversations*. The objective of the algorithm is to match the test cases with the conversations, so that the occurrence of the test cases in the trace is identified. By comparing each message of the test case with that of its corresponding conversation(s), we can determine whether IUTs interactions are as described in the test cases. Moreover, the possibility that a test case can appear several times in the trace is also taken into account. Therefore the global verdict for a given test case is based on the set of subverdicts, increasing the reliability of interoperability testing. Not only we can verify whether the test purposes are reached, but also non-interoperable behavior can be detected due to the difference between obtained subverdicts.

Trace Validation Tool To realize trace verification, we have developed a tool, which aims to automate the process of verifying the captured traces.

The tool is implemented in language Python3 [12] mainly for its advantages: easy to understand, rapid prototyping and extensive library. The tool is influenced by TTCN-3 [13], it implements basic TTCN-3 snapshots, behavior trees, ports, timers, messages types, templates, etc. However it provides several improvements, for example object-oriented message types definitions, automatic computation of message values, interfaces for supporting multiple input and

12. http://www.python.org/getit/releases/3.0/
13. http://www.ttcn-3.org/

Algorithm 1: Trace Verification Algorithm.

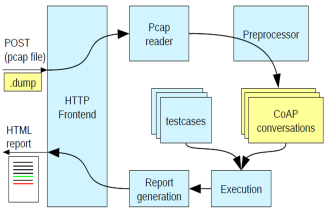
Input: filtered trace σ, test suite TS
Output: $verdict.TC_i$
Initialization: $TC = \emptyset, Counter_i = 0, Current_i = q_0^{TC_i}, verdict.TC_i = \emptyset$;
while $\sigma \neq \emptyset$ **do**
  σ=$\alpha.\sigma'$;
  if α *is a request* **then**
    for $TC_i \in TS$ **do**
      if $\alpha \in \Gamma(Current_i)$ **then**
        $TC = TC \cup TC_i$ /*Candidate test cases are added into the candidate test case set*/;
        $Current_i$=$Next_i$ where $(Current_i, \alpha, Next_i) \in \Delta^{TC_i}$
      end
    end
  end
  else
    for $TC_i \in TC$ **do**
      if $\alpha \in \Gamma(Current_i)$ **then**
        $Current_i$=$Next_i$ where $(Current_i, \alpha, Next_i) \in \Delta^{TC_i}$;
        if $Next_i \in \{Pass, Fail, Inconclusive\}$ **then**
          $Counter_i$=$Counter_i$+1 ;
          $verdict.TC_i.Counter_i$=$Next_i$ /* Emit the corresponding verdict to the test case*/;
          $verdict.TC_i$= $verdict.TC_i \cup verdict.TC_i.Counter_i$
        end
      end
      else
        TC=$TC \setminus TC_i$
      end
    end
  end
end
return $verdict.TC_i$

presentation format, implementing generic codecs to support a wide range of protocols, etc. These features makes the tool flexible, allowing to realize passive testing.

As illustrated in FIG. 7-(a), a web interface (HTTP frontend) was developed. Traces produced by a client and a server implementation of a request-response protocol, captured by the packet sniffer are submitted via the interface. Specifically in our work, the traces should be submitted in pcap format [14] by using tool Wireshark [15]. Each time a trace is submitted, it is then dealt by a prepossesor to filter only the messages relevant to the tested request-response protocol. In this way, the trace contains only the conversations between the client and server.

The next step is trace verification, which is carried out by taking into two files as input: the set of test cases (also programmed in Python, c.f. an example in FIG. 7-(b)) and the filtered trace. The trace is analyzed according to Algorithm 1, where test cases are verified on the trace to check their occurrence and validity. Finally, unrelated test cases are filtered out, while other test cases are associated with a verdict Pass, Fail or Inconclusive. The results are then reported from the HTTP frontend: Not only the verdict is reported, also the reasons in case of Fail or Inconclusive verdicts are explicitly given, so that users can understand the blocking issues of interoperability (cf. a use case in Section 5.2).

14. http://www.tcpdump.org/
15. http://www.wireshark.org/



(a) Trace Validation Tool

Interoperability Test Description			
Identifier:	TD_COAP_CORE_12		
Test Purpose:	Handle request containing several URI-Path options		
References:	CoAP Specification, clause 5.4.5, 5.10.2,6.5		
Pre-test conditions:	• Server offers a /seg1/seg2/seg3 resource		
Test Case:	**Step**	**Type**	**Description**
	1	Check	Client sends a confirmable GET request to server's resource. Request must contain: • Type = 0 (CON) • Code = 1 (GET) • Option type = URI-Path (one for each path segment)
	2	Check	Server sends response containing: • Code = 69 (2.05 content) • Payload = Content of the requested resource • Content type option

```
class TD_COAP_CORE_12 (CoAP Testcase)
        def run (self):
                self.match_coap ("client", CoAP (code = "get",
                                                 opt = Opt(CoAPOptionUriPath(), superset=True)))
                opts = list (filter ((lambda o: isinstance (o, CoAPOptionUriPath)), self.frame.coap["opt"]))
                        if len (opts) > 1:
                                self.setverdict ("pass", "multiple UriPath options")
                        else:
                                self.setverdict ("inconc", "only one UriPath option")

                        for o in opts:
                                if "/" in str (o["val"]):
                                        self.setverdict ("fail", "option %s contains a '/'" % repr (o))
                self.next_skip_ack()
                self.match_coap ("server", CoAP (       code = 2.05,
                                                        pl = Not (b""),
                                                        opt = Opt (CoAPOptionContentType()), ))
```

(b) A Test Case Example

FIG. 7 – *Trace Validation Tool.*

5 Experimentation

5.1 CoAP Plugtest

The proposed passive interoperability testing method as well as the tool were put into operation in CoAP Plugtest – the first formal CoAP interoperability test organized by the European project Probe-IT [16], IPSO Alliance, together with ETSI, held in Paris, France in March 2012. The objectives of this event are to get-together industry people to share their experiences, test their equipments in order to verify the interoperability, identify the issues and improve the applications. It also provides useful feedback to enhance the ongoing ETSI and IETF standardization. 15 developers and vendors of CoAP implementations, such as Sensinode [17], Watteco [18], Actility [19], etc. participated in the event.

During the test event, CoAP implementations from different manufactures are interconnected in pair-wise combinations. Test sessions are scheduled by ETSI so that each participant can test their products with all the other partners (1 hour per session). The following figure FIG. 8 shows the test bed architecture provided by ETSI for this event. Each company was given with a switch to connect their implementations in the test bed. Communication were routed using layer 2 and layer 3 routers.

5.2 CoAP Passive Interoperability Testing

The test suite is composed of 27 test cases (cf. Section 4.2), concerning the basic RESTful methods, Link format, Observation and Blockwise transfer of CoAP were served as test reference. Interoperability testing in lossy context was also realized by implementing a UDP gateway between client and server to emulate a lossy medium (c.f. FIG 4-(b)). The gateway

16. http://www.probe-it.eu/
17. http://www.sensinode.com/
18. http://www.watteco.com/
19. http://www.actility.com/

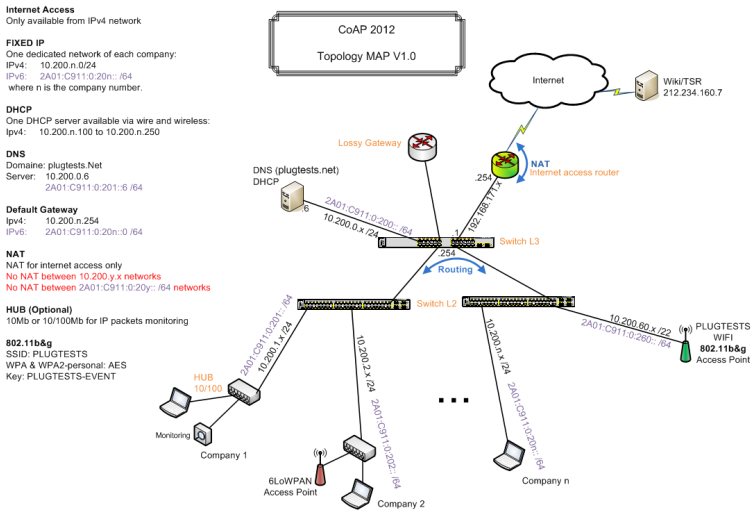


FIG. 8 – *CoAP Plugtest Test Bed.*

does not implement the CoAP protocol itself (It is not a CoAP proxy). It plays the following roles: *(i)* It performs NAT-style UDP port redirection towards the server (thus the client contacts the gateway and is transparently redirected towards the server). *(ii)* It randomly drops packets that are forwarded between the client and the server. In Plugtest, the gateway drops the packet randomly between Client and the server which goes more than 50% packet loss, which corresponds to the unreliable environment of the Internet of Things.

During the test, the tool Wireshark [20] was used to capture the packets changed by the CoAP applications. It produces pcaps file which contain the traces. Participants then submit the traces to the trace validation tool. Once a pcap file is submitted, a CoAP filtering is made using source IP address and destination IP address to filter only the conversations made between the client and the server. When the conversations are isolated, then trace verification is executed.

FIG. 9 shows the web interface where pcap files should be uploaded, as well as the test results found after uploading the pcap files and information on the results obtained. The results for each test case are shown at the top right corner in the summary table. In this table, the number of occurrence of each test case in the trace is counted, as well as a verdict *Pass*, *Fail* or *Inconc(lusive)* is given (or a test case which does not appear in the trace, it is marked as "*none*" and will not be verified on the trace). In this example, test case TD_COAP_CORE_1 (GET method in CON mode) is met 7 times in the trace. The verdict is *Inconclusive*, as explained by the tool: *CoAP.code ValueMismatch* (cf. the bottom of FIG. 9). In fact, according to the test case, after that the client sends a request (with Type value 0 and Code value 1 for a confirmable GET message), the server should send a response containing Code value 69(2.05 Content). However in the obtained trace, the server's response contains Code value 80(2.16), indicating that the request is successfully received without further information. This response is allowed in the specification, however does not satisfy the test case. In fact, the same situation exists in all the other conversations that correspond to this test case. The global verdict is *Inconclusive*.

20. http://www.wireshark.org/

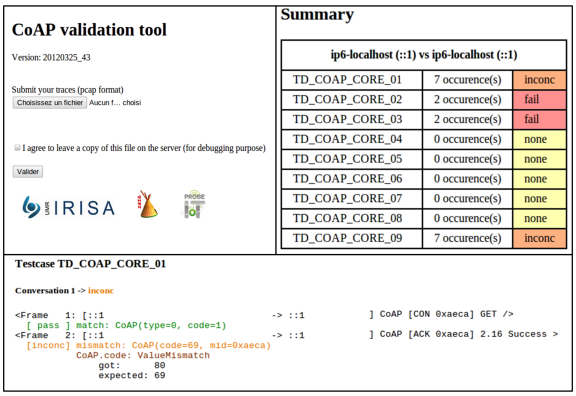


FIG. 9 – *An example of trace validation tool use case.*

5.3 Results

A total of 3081 tests were executed during this two days event within 234 test sessions. The feedback from participants on the testing method and passive validation tool is positive, due to the following results:

– To our knowledge, it is the first time that an interoperability event is conducted by passive testing. Conventional interoperability events rely on *active testing*, which is done by actively stimulating the implementations and verifying the corresponding outputs. However, most of stimulation of these IUTs is manual, which need the intervention of experts for installation, synchronization, etc., Besides, according to our experience (Sabiguero et al., 2007), active testing cause many false negative verdicts: most of *Fail* verdicts are in fact due to the inappropriate network configuration, synchronization and inappropriate IUTs configuration. Also, the non-intrusive property of passive testing allows discover interoperability issues in operational environment, where the normal operations of the IUTs are not disturbed.

– The automation of trace verification increases the efficiency. According to ETSI, most of the time (about 60%) of interoperability testing is spent on trace validation, including verifying the results and looking at the problems of unsuccessful tests with the help of experts. Passive automated trace analysis allows to considerably reduce the time. In consequence, within the same time interval, the number of executed tests are drastically increased. During the CoAP plugtest, 3081 tests were executed within two days. Compared with past conventional plugtest event, e.g. IMS InterOp Plugtest [21], 900 tests in 3 days, the number of test execution and validation benefited a drastic increase. The re-usability of the test cases implemented by the tool also, will contribute to increasing efficiency for future CoAP interoperability tests.

– The passive testing tool is easy to use. In fact, the participants only need to submit their traces via a web interface. Complicated test configuration is avoided. The test reports provided by the validation tool makes the reason of non-interoperable behavior be clear. Besides, another advantage of the validation tool is that it can be used outside of an in-

21. http://www.etsi.org/plugtests/ims2/About_IMS2.htm

teroperability event. (It is hosted at http://senslab2.irisa.fr/coap/). In fact, the participants started trying the tool one week before the event by submitting more than 200 traces via Internet. This allows the participants to prepare in advance the test event and to revise, if necessary, their implementations.

– Moreover, the passive testing tool shows its capability of non-interoperability detection (cf. TAB. 1): 5.9% show non-interoperability detected w.r.t basic RESTful methods; 7.8% for Link Format, 13.4% for Block transfer and 4.3% for Observe. The results help the vendors in uncovering the blocking issues and achieving higher quality.

	Executed Tests	Non-interoperable
RESTful Methods	2798	166 (5.9%)
Link Format	77	6 (7.8%)
Block transfer	112	15 (13.4%)
Observe	94	4 (4.3%)
Total	3081	191 (6.2%)

TAB. 1 – *CoAP Plugtest Results.*

6 Conclusions and Future Work

In this paper, we have introduced an approach for the interoperability testing of CoAP protocol, including the methodology, test architecture, tool implementation as well as experimental results. The most important properties of CoAP protocols were defined as test purposes, and verified by using the technique of passive testing. Trace verification was automated by implementing a trace validation tool. The method and tool were put into operation during the ETSI CoAP Plugtest event of March 2012 in Paris, where an amount of tests were successfully performed.

Future work will consider how to improve the test method in two main directions: (i) Due to the uncontrollable nature of passive testing, Inconclusive verdicts are emitted leading to rerunning the test or a post-analysis. Solutions to reduce Inconclusive verdicts are to be studied. (ii) In this work, we have chosen to use offline trace verification, i.e., traces are at first recorded and then processed. Future work will consider online trace verification.

References

Bormann, c. and Z.Shelby (2012). Blockwise transfers in coap. *draft-ietf-core-block-05*.

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Phd thesis, University of California.

Hartke, K. (2012). Observing resources in coap. *draft-ietf-core-observe-04*.

Lee, D., A.N.Netravali, K.K.Sabnani, B.Sugla, and A.John (1997). Passive testing and applications to network management. *International conferences on network protocols, ICNP'97*, 113–122.

Rayner, D. (1987). Osi conformance testing. *Computer Networks and ISDN Systems - Special Issue: Protocol Specification and Testing archive 14*, 79–98.

Sabiguero, A., A. Baire, A. Boutet, and C.Viho (2007). Network protocol interoperability testing based on contextual signatures. *18th IFIP/IEEE International Workshop on Distributed Systems: Oprations and Management, DSOM 01*.

Schulz, S., A.Wiles, and S.Randall (2007). Tplan-a notation for expressing test purposes. *ETSI, TestCom/FATES LNCS 4581*, 292–304.

Shelby, Z. (2011). Core link format. *draft-ietf-core-linkformat-09*.

Shelby, Z., K.Hartke, and B.Frank (2011). Constrained application protocol (coap). *draft-ietf-core-coap-08*.

Verhaard, L., J.Tretmans, P. Kars, and E. Brinksma (1992). On asynchronous testing. *Protocol Test Systems of IFIP Transactions*, 55–66.

Viho, C., S. Barbin, and L. Tanguy (2001). Towards a formal framework for interoperability testing. In *Proceedings of the IFIP TC6/WG6.1 - 21st International Conference on Formal Techniques for Networked and Distributed Systems*, FORTE '01, pp. 53–68.

Zaidi, F., A. Cavalli, and E. Bayse (2009). Network protocol interoperability testing based on contextual signatures. *SAC '09 Proceedings of the 2009 ACM symposium on Applied Computing*, 2–7.

Résumé

Le protocole CoAP (Constrained Application Protocol) est un protocole applicatif défini pour l'Internet des objets (IdO). Dans ce contexte de l'IdO, l'interaction entre les objets communicants est de type machine-to-machine sans intervention humaine. Cela accroit les exigences d'interopérabilité et oblige à définir des méthodes non intrusives pour les tester. Nous proposons une méthode de test passif basé uniquement sur les observations des interactions entre les composants à tester. Cette méthode et l'outil correspondant ont été utilisés lors de l'événément du test d'interopérabilité (Plugtest) du protocole CoAP organisé par l'ETSI à Paris en Mars 2012. Cela a permis de tester avec succès un grand nombre d'implémentations CoAP. Dans cet article, la méthode de test passif ainsi que l'outil associé sont présentés ainsi que les résultats lors de leur utilisation au cours du Plugtest, montrant la validité et la pertinence de notre approche.