

Passive Interoperability Testing for Request-Response Protocols: Method, Tool and Application on CoAP Protocol

Nanxing Chen and César Viho

IRISA/University of Rennes 1
Campus de Beaulieu, Avenue du Général Leclerc, 35042 Rennes Cedex, FRANCE
`nanxing.chen@irisa.fr`, `cesar.viho@irisa.fr`

Abstract. Passive testing is a technique that aims at testing a running system by only observing its behavior without introducing any test input. The non-intrusive nature of passive testing makes it an appropriate technique for interoperability testing, which is an important activity to ensure the correct collaboration of different network components in operational environment. In this paper we propose a passive interoperability testing approach, especially for request-response protocols in the context of client-server communications. According to the interaction pattern of request-response protocols, the observed interactions (trace) between the network components under test can be considered as a set of conversations between client and server. Then, a procedure to map each test case into these conversations is carried out, which intends to verify the occurrence of the generated test cases as well as to determine whether interoperability is achieved. The trace verification procedure has been automated in a passive testing tool, which analyzes the collected traces and deduces appropriate verdicts. The proposed method and the testing tool were put into operation in the first interoperability testing event of Constrained Application Protocol (CoAP) held in Paris, March 2012 in the scope of the Internet of Things. By using this approach, an amount of CoAP applications from different vendors were successfully and efficiently tested, revealing their interoperability degree.

Keywords: Interoperability Testing, Passive Testing, Request-Response Protocol, CoAP

1 Introduction

With the development and increasing use of distributed systems, computer communication mode has changed. There is increasing use of clusters of workstations connected by a high-speed local area network to one or more network servers. In this environment, resource access leads to communications that are strongly request-response oriented. This tendency resulted in a large amount of protocols such as Hypertext Transfer Protocol (HTTP)¹, Session Initiation Protocol

¹ <http://tools.ietf.org/html/rfc2616>

(SIP)², and very recently the Constrained Application Protocol (CoAP) [1], etc. Due to the heterogeneous nature of distributed systems, the interoperability of these protocol applications is becoming a crucial issue. In this context, interoperability testing is required before the commercialization to ensure correct collaboration and guarantee the quality of services.

This paper proposes a methodology for the interoperability testing of request-response protocols. Specifically, we apply the technique of *passive testing*, which aims at testing a running system by only observing its external behavior without disturbing its normal operation. The methodology consists of the following main steps: *(i)* Interoperability test purposes extraction from the protocol specifications. Each test purpose specifies an important property to be verified. *(ii)* For each test purpose, an interoperability test case is generated, in which the detailed events that need to be observed are specified. *(iii)* Behavior analysis. In order to verify whether the test purposes are reached, as well as to detect non-interoperable behavior, traces produced by protocol implementations are processed by keeping only the client-server *conversations* with respect to the interaction model of request-response protocols. These conversations will further be analyzed by a trace verification algorithm to identify the occurrence of the generated test cases and to emit an appropriate verdict for each of them.

The proposed passive interoperability testing method has been implemented in a test tool, which was successfully put into operation during ETSI CoAP Plugtest - the first formal CoAP interoperability testing event held in Paris, March 2012 in the context of the Internet of Things.

This paper is organized as follows: Section 2 introduces the background and motivation. Section 3 proposes the methodology for passive interoperability testing of request-response protocols. Section 4 describes the application of this method on CoAP Plugtest as well as the experimental results. Finally, we conclude the paper and suggest further research directions in Section 5.

2 Background and Motivation

The request-response oriented communication is generally used in conjunction with the client-server paradigm to move the data and to distribute the computations in the system by requesting services from remote servers. The typical sequence of events in requesting a service from a remote server is: a client entity sends a request to a server entity on a remote host, then a computation is performed by the server entity. And, finally a response is sent back to the client.

Request-response communications are now common in the fields of networks. Request-response exchange is typical for database or directory queries and operations, as well as for many signaling protocols, remote procedure calls or middleware infrastructures. A typical example is REST (Representational State Transfer) [10], an architecture for creating Web service. In REST, clients initiate request to servers to manipulate resources identified by standardized Uniform

² <http://www.ietf.org/rfc/rfc3261.txt>

Resource Identifier (URI). E.g., the HTTP methods GET, POST, PUT and DELETE are used to read, create, update and delete the resources. On the other hand, servers process requests and return appropriate responses. REST is nowadays popular, which is applied in almost all of the major Web services on the Internet, and considered to be used in the *Internet of Things*, aiming at extending the Web to even the most constrained nodes and networks. This goes along the lines of recent developments, such as Constrained RESTful Environments (CoRE)³ and CoAP, where smart things are increasingly becoming part of the Internet and the Web, confirming the importance of request-response communication.

Promoted by the rapid development of computer technology, protocols using the request-response transaction communications are increasing. Normally, protocol specifications are defined in a way that the clients and servers interoperate correctly to provide services. To ensure that they collaborate properly and consequently satisfy customer expectations, protocol testing is an important step to validate protocol implementations before their commercialization. Among them, *conformance testing* [7] verifies whether a protocol application conforms to its specifications. It allows developers to focus on the fundamental problems of their protocol implementations. However, it is a well-known fact that, even following the same standard, clients and servers might not interoperate successfully due to several reasons: poorly specified protocol options, incompleteness of conformance testing, inconsistency of implementation, etc. These aspects may cause the interoperable issues in realizing different services. However, the heterogeneous nature of computer systems requires interoperability issues to be solved before the deployment of the product. Therefore, *interoperability testing* [11] is required to ensure that different protocol applications communicate correctly while providing the expected services.

To perform interoperability testing (*iop* for short in the sequel), the conventional method is the *active testing* approach (e.g. [8, 4]). It requires to deploy a *test system* (TS) that stimulates the *implementations under test* (IUT) and verify their reactions. Although widely used, active testing has limitations: test can be difficult or even impossible to perform if the tester is not provided with a direct interface to stimulate the IUTs, or in operational environment where the normal operation of IUTs cannot be shutdown or interrupted for a long period of time. On the contrary, *passive testing* represents an alternative, which aims at testing a system by passively observing its inputs/outputs without interrupting its normal behavior.

Until now, passive testing has been studied and applied to computing systems to supervise distributed computations, communications networks for fault management [6], protocol testing [3, 5], runtime verification [12], etc. In this paper, we will provide a passive interoperability testing methodology for request-response protocols. We have chosen to use passive testing technique for the following arguments: First, passive testing does not insert arbitrary test messages, thus is suitable for interoperability testing in operational environment as is often con-

³ <http://datatracker.ietf.org/wg/core/charter/>

cerned by request-response services. Also, passive testing does not introduce extra overhead into the networks, hence is appropriate for testing in the context of Internet of Things, where devices are resource limited.

The work presented in this paper is original. It involves using the non-intrusive passive testing technique to verify interoperability, where there exist only few works in the literature. Moreover, the method does not only verify whether the test purposes are reached, but also detects non-interoperable behavior. Last but not least, the procedure of trace verification is automated by implementing a tool, which was successfully put into practice for the test of an important machine-to-machine communication protocol CoAP. To our knowledge, it was the first time that passive automated interoperability testing method was applied in an interoperability testing event, which increased drastically the efficiency, while keeping the capacity of non-interoperability detection.

3 Interoperability Testing for Request-Response Protocols

3.1 Formal Model

Specification languages for reactive systems can often be given a semantics in terms of labeled transition systems. In this paper, we use the IOLTS (Input-Output Labeled Transition System) model [9], which allows differentiating input, output and internal events while precisely indicating the interfaces specified for each event.

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 . $\Gamma(q) =_{def} \{\alpha \in \Sigma^M \mid \exists q', (q, \alpha, q') \in \Delta^M\}$ 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 passive interoperability testing architecture (c.f. Fig.1) for request-response protocols involves a *test system* and a *system under test*, composed of two *implementations under test*, namely a *client* and a *server*. In passive iop testing, the test system has two main roles: (i) Observe and collect the information exchanged (trace) between the client and the server. (ii) Analyze the collected trace to check interoperability. Generally, trace verification can be done *online* to monitor the system and report abnormalities at any time. Elsewise it can be done *offline*, i.e, the traces during the test execution are stored in a file and will be analyzed in a posteriori manner. As passive testing does not apply any stimulus, testing activity is only based on an accurate level of observation, relying

on the set up of sniffer at *point(s) of observation* (PO) to observe the messages exchanged between the client and the server. In this paper we consider *black-box* testing: the test system is not aware of the internal structure of IUTs. Only their external behavior can be verified during their interactions.

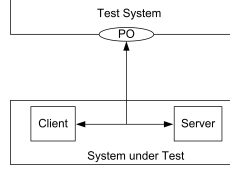


Fig. 1. Passive interoperability testing architecture

The testing procedure is illustrated in Fig.2. It consists of the following main steps:

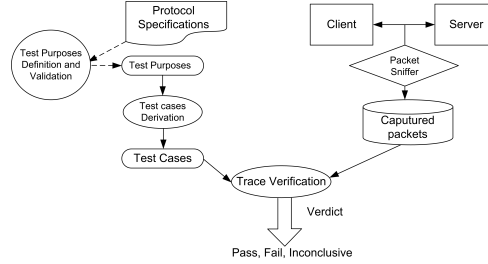


Fig. 2. Passive interoperability testing procedure

1. Interoperability test purposes (ITP) selection from protocol specifications. An ITP is in general informal, in the form of an incomplete sequence of actions representing a critical property to be verified. Generally it can be designed by experts or provided by standards guidelines for test selection. Test purpose is a commonly used method in the field of testing to focus on the most important properties of a protocol, as it is generally impossible to validate all possible behavior described in specifications. Nonetheless, an ITP itself must be correct w.r.t the specification to assure its validity. Formally, an ITP can be represented by a deterministic and complete IOLTS equipped with trap states used to select targeted behavior.

$ITP = (Q^{ITP}, \Sigma^{ITP}, \Delta^{ITP}, q_0^{ITP})$ where:

- $\Sigma^{ITP} \subseteq \Sigma^{S_{client}} \cup \Sigma^{S_{server}}$. where S_{client} and S_{server} are the specifications on which the IUTs are based.
- 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. Once the ITPs chosen, an iop test case (ITC) is generated for each ITP. An ITC is the detailed set of instructions that need to be taken in order to perform the test. The generation of iop test case can be either manual, as usually done in most of the interoperability events, also for “young” protocols whose specifications are not yet stable. ITCs can also be generated automatically by using various formal description techniques existing in the literature such as [11, 4]. Formally, an iop test case ITC is represented by an IOLTS: $ITC = (Q^{ITC}, \Sigma^{ITC}, \Delta^{ITC}, q_0^{ITC})$, 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 reach the ITP. Σ^{ITC} denotes the observation of the messages from the interfaces. Δ^{ITC} is the transition function. In active testing, ITCs are usually controllable. i.e, ITC contains stimuli that allow controlling the IUTs. On the contrary, in passive testing, ITCs are only used to analyze the observed trace produced by the IUTs. The correct behavior of IUTs implies that the trace produced by the IUTs should exhibit the events that lead to *Pass* verdicts described in the test cases. In the sequel, an ITC is supposed to be deterministic. The set of test cases is called a *test suite*. An example of ITP and ITC can be seen on Fig.6 in Section 4.2.
 3. Analyze the observed behavior of the IUTs against each test case and issue a verdict *Pass*, *Fail* or *Inconclusive*. In this paper we choose *offline* testing, where the test cases are pre-computed before they are executed on the trace.

3.3 Request-response Protocol Passive Interoperability Testing

In offline passive interoperability testing for request-response protocols, the packets exchanged between the client and server are captured by a packet sniffer. The collected traces are stored in a file. They are key to conclude whether the protocol implementations interoperate (c.f. Fig.2).

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* [6]. 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 [5], 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 request-response protocols. As the interoperability testing of this kind of protocol 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 the tested request-response 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 case 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. 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 TC .
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*.

Algorithm 1: Trace verification for request-response protocols

```

Input: filtered trace  $\sigma$ , 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
   $\sigma = \alpha.\sigma'$ ;
  if  $\alpha$  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  $verdict.TC_i$ 

```

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 in fact, 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 request-response protocols, each filtered traces are in fact composed of a set of *conversations*. The objective of the algorithm is intended 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 expected as they are described by 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.

3.4 Passive Testing Tool

To realize trace verification, we have developed a passive testing tool [2], which aims to automate the process of trace verification. A description of this tool is given in Fig.3.

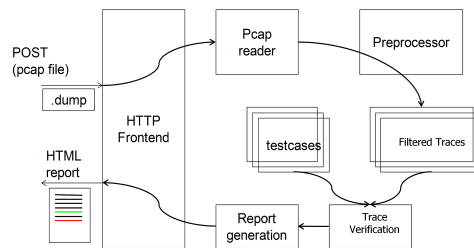


Fig. 3. Passive interoperability testing tool

The tool is implemented in language Python3⁴ mainly for its advantages: easy to understand, rapid prototyping and extensive library. The tool is influenced by TTCN-3⁵. 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 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.3, a web interface (HTTP frontend) was developed. Traces produced by client and server implementations 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 format⁶. Each time a trace is submitted, it is then dealt by a preprocessor to filter only the messages relevant to the tested request-response protocol, i.e., to keep only the conversations made between the client and server.

The next step is trace verification, which takes into two files as input: the set of test cases 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.

⁴ <http://www.python.org/getit/releases/3.0/>

⁵ <http://www.ttcn-3.org/>

⁶ <http://www.tcpdump.org/>

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 (c.f. a use case in Section 4.3).

4 Experimentation

The proposed passive interoperability testing method for request-response protocols has been put into operation in the CoAP Plugtest - the first formal CoAP interoperability testing event in the context of the Internet of Things.

4.1 CoAP Protocol Overview

The Internet of Things (IoT) is a novel paradigm that is rapidly gaining ground in the field of modern wireless telecommunications. It combines the general meaning of the term ‘Internet’ with smart objects, such as sensors, Radio-Frequency Identification (RFID) tags, mobile phones, etc. which are able to interact with each other and cooperate to reach common goals. However, applications in the context of IoT are typically resource limited: they are often battery powered and equipped with slow micro-controllers and small RAMs and ROMs. The data transfer is performed over low bandwidth and high packet error rates, and the communication is often machine-to-machine. To deal with the various challenging issues of constrained environment, the Constrained Application Protocol (CoAP) has been designed by Constrained RESTful Environments (CoRE) working group⁷ to make it possible to provide resource constrained devices with Web service functionalities.

CoAP protocol is a request-response style protocol. A CoAP request is sent by a client to request an action on a resource identified by a URI on a server. The server then sends a response, which may include a resource representation. CoAP is consist of two-layers (c.f. Fig.4): (i) CoAP transaction layer deals with UDP and the asynchronous interactions. Four types of message are defined at this layer: Confirmable (CON, messages require acknowledgment), Non-Confirmable (NON, messages do not require acknowledgment), Acknowledgment (ACK, an acknowledgment to a CON message), and Reset (RST, messages indicate that a Confirmable message was received, but some context is missing to properly process it. eg. the node has rebooted). (ii) CoAP Request/Response layer is responsible for the transmission of requests and responses for resource manipulation and interoperation. CoAP supports four request methods: *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 request URI to be deleted.

⁷ <http://datatracker.ietf.org/wg/core/charter/>

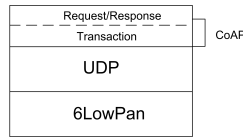


Fig. 4. Protocol stack of CoAP

4.2 Test Purposes and Test Cases

As one of the most important protocol for the future Internet of Things, the application of CoAP is potentially wide, especially concerning energy, building automation and other M2M applications that deal with manipulation of various resources on constrained networks. For that CoAP applications be widely adopted by the industry, hardware and software implementations from different vendors need to interoperate and perform well together. Regarding the specifications of CoAP [1], a set of 27 interoperability test purposes are selected. To ensure that the ITPs are correct w.r.t the specifications, the ITPs were chosen and cross-validated by experts from ETSI⁸, IRISA⁹ and BUPT¹⁰, and reviewed by IPSO alliance. The test purposes concern the following properties:

- Basic CoAP methods GET, PUT, POST and DELETE. This group of tests involves in verifying that both CoAP client and server interoperate correctly w.r.t different methods as specified in [1], even in lossy context as often encountered by M2M communication. (c.f. an example in Fig.5-(a)).
- Resource discovery¹¹. As CoAP applications are considered to be M2M, they must be able to discover each other and their resources. Thus, CoAP standardizes a resource discovery format defining a path prefix for resource as */.well-known/core*. The interoperability testing of resource discovery requires verifying that: when the client requests */.well-known/core* resource, the server sends a response containing the payload indicating all the available links.
- Block-wise transfer¹²: CoAP is based on datagram transports such as UDP, which limits the maximum size of resource representations (64 KB) that can be transferred. In order to handle large payloads, CoAP defines an option *Block*, in order that large sized resource representation can be divided in several blocks and transferred in multiple request-response pairs. The interoperability testing of this property therefore involves in verifying that: when the client requests or creates large payload on the server, the server should react correctly to the requests (c.f. an example in Fig.5-(b)).
- Resource observation¹³ is an important property of CoAP applications, which provides a built-in push model where a subscription interface is provided for client to request a response whenever a resource changes. This push is accomplished by the

⁸ <http://www.etsi.org/WebSite/homepage.aspx>

⁹ <http://www.irisa.fr/>

¹⁰ <http://www.bupt.edu.cn/>

¹¹ <http://tools.ietf.org/id/draft-shelby-core-link-format-00.txt>

¹² <http://tools.ietf.org/html/draft-ietf-core-block-08>

¹³ <http://tools.ietf.org/html/draft-ietf-core-observe-04>

device with the resource of interest by sending the response message with the latest change to the subscriber. The interoperability testing of this property consists of: upon different requests sent by the client to register or cancel its interest for a specific resource, the server should react correctly. i.e., it adds the client to the list of observers for the resource in the former case, while remove it from the list in the latter case (c.f. an example in Fig.5-(c)).

The following figure demonstrates some typical examples of CoAP transactions. Fig.5-(a) illustrates a confirmable request sent by the client, asking for the resource of humidity. Upon the reception of the request, the server acknowledges the message, transferring the payload while echoing the Message ID generated by the client. Fig.5-(b) illustrate a block-wise transfer of a large payload (humidity) requested by the client. Upon the reception of the request, the server divides the resource into 4 blocks and transfers them separately to the client. Each response indicates the block number and size, as well as whether there are further blocks (indicated by value m). Fig 5-(c) illustrates an example of resource observation, including registration and cancellation. At first, the client registers its interest in humidity resource by indicating Observe option. After a while, it cancels its intention by sending another GET request on the resource without Observe option.

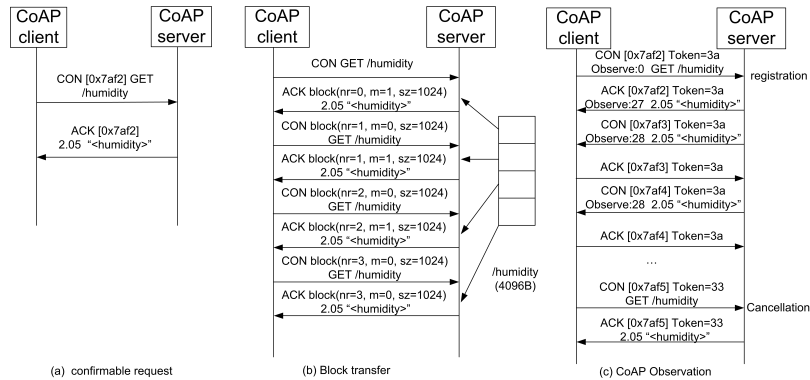


Fig. 5. CoAP transaction examples

Once the set of test purposes are defined, a test case is derived for each test purpose. The following figure shows an example. The test purpose focuses on the GET method in confirmable transaction mode. i.e., when the client sends a GET request (It implies parameters: a Message ID, Type=0 for confirmable transaction mode, Code=1 for GET method. The parameters are omitted in the figure due to the limitation of space), 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 a code other than 69. These events are noted by *m* in the figure for the sake of simplicity). However other unexpected behavior leads to *Fail* verdict (labeled by *Otherwise*). The test cases are derived, validated by the experts of IRISA, BUPT, ETSI and IPSO Alliance w.r.t the specifications of CoAP. They are implemented in the testing tool, taking into account all the verdicts. For simplicity, during the test event, only expected behavior to be observed is provided to the users as test specification document (Fig.6-(c)). Nevertheless, in case of *Inconclusive* or *Fail* verdicts, an explication will be provided to the users.

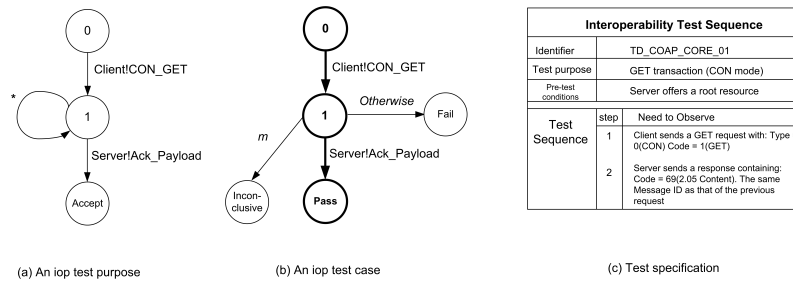


Fig. 6. Example of test purpose and test case

4.3 CoAP Plugtest

CoAP Plugtest¹⁴ is the first formal interoperability event, held in Paris, March 2012 during two days for CoAP protocol in the scope of Internet of Things. It was co-organized by the Probe-IT¹⁵ (the European project in the context of Internet of things), the IPSO Alliance and ETSI¹⁶ (the European Telecommunication Standard Institute). The objective of the CoAP plugtest is to enable CoAP implementation vendors to test end to end interoperability with each other. Also, it is an opportunity for standards development organization to review the ambiguities in the protocol specifications. 15 main developers and vendors of CoAP implementations, such as Sensinode¹⁷, Watteco¹⁸, Actility¹⁹, etc. participated in the event. Test sessions are scheduled by ETSI so that each participant can test their products with all the other partners.

The testing method is based on the technique of passive testing as described in Section 3. During the test, the participants start launching their equipments.

¹⁴ <http://www.etsi.org/plugtests/coap/coap.htm>

¹⁵ <http://www.probe-it.eu/>

¹⁶ <http://www.etsi.org/>

¹⁷ <http://www.sensinode.com/>

¹⁸ <http://www.watteco.com/>

¹⁹ <http://www.actility.com/>


Packets exchanged between CoAP implementations (CoAP client and CoAP server) were captured by using *Wireshark*²⁰. Captured traces were analyzed against the test cases by using the passive testing tool presented in Section 3.4. For CoAP Plugtest, the tool was developed to support the message formats of the CoAP drafts. It checks the basic message type code as well as parameters such as token or message ID. CoAP test suite is implemented. During the plugtest, 410 traces produced by the CoAP devices were captured and then submitted and processed by the passive validation tool. Received traces are filtered, parsed and analyzed against the test cases. And an appropriate verdict *Pass*, *Fail*, or *Inconclusive* is issued for each test purpose. A use case of the tool is as follows:

CoAP validation tool

Version: 20120325_43

Submit your traces (pcap format)
 Aucun f... choisi

I agree to leave a copy of this file on the server (for debugging purpose)



Summary

ip6-localhost (::1) vs ip6-localhost (::1)		
TD_COAP_CORE_01	7 occurrence(s)	inconc
TD_COAP_CORE_02	2 occurrence(s)	fail
TD_COAP_CORE_03	2 occurrence(s)	fail
TD_COAP_CORE_04	0 occurrence(s)	none
TD_COAP_CORE_05	0 occurrence(s)	none
TD_COAP_CORE_06	0 occurrence(s)	none
TD_COAP_CORE_07	0 occurrence(s)	none
TD_COAP_CORE_08	0 occurrence(s)	none
TD_COAP_CORE_09	7 occurrence(s)	inconc

Testcase TD_COAP_CORE_01

Conversation 1 -> **inconc**

```

<Frame 1: [::1] -> [::1] ] CoAP [CON 0xaeca] GET />
  | pass | match: CoAP (type=0, code=1)
<Frame 2: [::1] -> [::1] ] CoAP [ACK 0xaeca] 2.16 Success >
  | inconc | mismatch: Coap (code=69, mid=0xaeca)
              CoAP.code: ValueMismatch
                  got: 80
                  expected: 69

```

Fig. 7. Trace verification tool use case

The top left image is the user interface of the tool. Users can submit their traces in pcap format. Then, the tool will execute the trace verification algorithm and return back the results as 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 *Inconclusive* is given (For a test case which does not appear in the trace, it is marked as “none” and will not be verified on the trace). Moreover, users can view the details about the verdict for each test case. 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.7). 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

²⁰ <http://www.wireshark.org/>

should send a response containing Code value 69(2.05 Content). However in the obtained trace, the server's response contains Code value 80, indicating that the request is successfully received without further information. This response is not forbidden in the specification, however does not allow to satisfy the test case. In fact, the same situation exists in all the other conversations that correspond to this test case. Therefore, the global verdict for this test purpose is *Inconclusive*.

4.4 Results

The CoAP plugtest was a success with regards to the number of executed tests (3081) and the test results (shown in the sequel). The feedback from participants on the testing method and passive validation tool is positive mainly due to the following aspects:

- To our knowledge, it is the first time that an interoperability event is conducted by using automatic passive testing approach. In fact, conventional interoperability methods that rely on *active testing* are often complicated and error-prone. According to our previous experience [8], active testing requires usually experts for installation, configuration, and cannot be run reliably by the vendors. Also, test cases are not flexible, as they involve the ordering of tests, needs to re-run a test, etc. Moreover, inappropriate test configuration cause often false verdicts. By using passive testing, complicated test configuration is avoided. Bug fixes in the tool do not require re-running the test. Moreover, it provides the ability to test products in operational environment.
- Also, the passive testing tool shows its various advantages: By using passive testing tool, the participants only need to submit their traces via a web interface. The human readable test reports provided by the validation tool makes the reason of non-interoperable behavior be clear at a glance. Besides, another advantage of the validation tool is that it can be used outside of an interoperability event. 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. Also, passive automated trace analysis allows to considerably increase the efficiency. During the CoAP plugtest, 3081 tests were executed within two days, which are considerable. Compared with past conventional plugtest event, e.g. IMS InterOp Plugtest²¹, 900 tests in 3 days, the number of test execution and validation benefited a drastic increase.
- Moreover, the passive testing tool not only validates test purposes, but also shows its capability of non-interoperability detection: Among all the traces, 5.9% reveal non-interoperability w.r.t basic RESTful methods; 7.8% for Link Format, 13.4% for Blockwise transfer and 4.3% for resource observation. The results help the vendors discover the blocking issues and to achieve higher quality implementations.

5 Conclusion and Future Work

In this paper, we have proposed a passive interoperability testing methodology for request-response protocols. According to their interaction mode, the traces

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

collected during the test were analyzed to verify the occurrence of the test cases. Also, interoperability is determined by comparing each event in the test case with that of its related conversation(s) in the trace. The trace verification procedure has been automated by implementing a testing tool, which was successfully put into operation in the first interoperability testing event of CoAP protocol, where an amount of protocol applications were tested, and non-interoperable behavior was detected. Future work intends to improve the passive validation tool. E.g online trace verification and solutions to solve message overlapping will be considered. Also, the tool is considered to be extended to a wider range of protocols and more complex test configurations.

References

- [1] Shelby, Z., Hartke, K., Frank, B.: Constrained application protocol (CoAP), draft-ietf-core-coap-08, (2011)
- [2] Baire, A., Viho, C., Chen, N.: Long-term challenges in TTCN-3 a prototype to explore new features and concepts, ETSI TTCN-3 User Conference and Model Based Testing Workshop Conference, (2012)
- [3] Arnedo, J.A., Cavalli, A., Núñez, M.: Fast Testing of Critical Properties through Passive Testing. Lecture Notes on Computer Science, Volume 2644/2003, pp. 295-310, (2003)
- [4] Seol, S., Kim, M., Kang, S., Chanson, S.T.: Interoperability test generation and minimization for communication protocols based on the multiple stimuli principle. IEEE Journal on selected areas in Communications, 22 (10), pp. 2062-2074, (2004)
- [5] Zaidi, F., Cavalli, A., Bayse, E.: Network Protocol Interoperability Testing based on Contextual Signatures. The 24th Annual ACM Symposium on Applied Computing SAC'09, (2009)
- [6] Lee, D., Netravali, A.N., Sabnani, K.K., Sugla, B., John, A.: Passive testing and applications to network management. In Int Conference on Network Protocols, ICNP'97, pp. 113-122, (1997)
- [7] ISO. Information Technology - Open System Interconnection Conformance Testing, Methodology and Framework, Parts 1-7. International Standard ISO/IEC 9646/1-7, (1994)
- [8] Sabiguero, A., Baire, A., Boutet, A., Viho, C.: Virtualized Interoperability Testing: Application to IPv6 Network Mobility. 18th IFIP/IEEE Int Workshop on Distributed Syst: Operations and Management. (2007)
- [9] Verhaard, L., Tretmans, J., Kars, P. Brinksma, Ed.: On asynchronous testing. In Protocol Test Systems, vol C-11 of IFIP Transactions, pp. 55-66. (1992)
- [10] Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, (2000)
- [11] Desmoulin, A., Viho, C.: Automatic Interoperability Test Case Generation Based on Formal Definitions. Lecture Notes in Computer Science, vol 4916, pp. 234-250, (2008)
- [12] Falcone, Y., Fernandez, J.C., Mounier, L.: Runtime Verification of Safety-Progress Properties. In Runtime Verification, Lecture Notes in Computer Science, vol 5779, pp. 40-59, (2009)