Online ISSN 1848-3380, Print ISSN 0005-1144
ATKAFF 54(4), 448–458(2013)

Nanxing Chen, César Viho, Anthony Baire, Xiaohong Huang, Jiexi Zha

Ensuring Interoperability for the Internet of Things: Experience with CoAP Protocol Testing

DOI 10.7305/automatika.54-4.418
UDK 004.451.642
IFAC 4.0.5; 2.8.2

Original scientific paper

Constrained Application Protocol (CoAP) is a specialized web transfer protocol, designed for realizing interoperation with constrained networks and nodes for machine to machine applications like smart energy, building automation, etc. As an important ubiquitous application protocol for the future Internet of Things, CoAP will be potentially implemented by a wide range of smart devices to achieve cooperative services. Therefore, a high level of interoperability of CoAP implementations is crucial. In this context, CoAP Plugtest – the first formal CoAP interoperability testing event was held in Paris, March 2012 to motivate vendors to verify the interoperability of their equipments. The event turned to be successful due to our contribution, including the test method and tool. This paper presents the testing method and procedure for the CoAP Plugtest event. To carry out the tests, a set of test objectives concerning the most important properties of CoAP have been selected and used to measure the interoperability of CoAP implementations. The process of verification has been automated by implementing a test validation tool based on the technique of passive testing. By using the test tool, a number of devices were successfully tested.

Key words: CoAP, Interoperability testing, IoT, Passive testing

Osiguravanje međuoperabilnosti za Internet stvari: iskustvo s testiranjem CoAP protokola. Constrained Application Protocol (CoAP) je specijalizirani prijenosni protokol, dizajniran za realizaciju međuoperabilnosti uz ograničene mrežame i čvorove za primjene poput pametne energije, automatizacije u zgradarstvu itd. Kao važan i sveprisutan protokol za Internet stvari CoAP bi mogao biti implementiran kod velikog broja pametnih uređaja kako bi se ostvarile kooperativne usluge. Zbog toga je od velike važnosti postići visoku razinu međuoperabilnosti CoAP implementacija. U tom kontekstu, CoAP Pluqtest - prvo formalno testiranje CoAP međuoperabilnosti je održano u Parizu u ožujku 2012. kako bi se motivirali prodavači da provjere međuoperabilnost svoje opreme. Testiranje je bilo uspješno zahvaljujući našem doprinosu koji uključuje metodu i alate za testiranje. U ovom radu prikazana je metoda i procedura testiranja za CoAP Pluqtest. Kako bi se proveli testovi odabran je skup ciljeva koji se odnose na najvažnija svojstva CoAP protokola i oni su korišteni za mjerenje međuoperabilnosti CoAP implementacija. Proces verifikacije je automatiziran implementacijom alata za provjeru testa koji se temelji na tehnici pasivnog testiranja. Korištenjem alata za testiranje uspješno su testirani brojni uređaji.

Ključne riječi: CoAP, testiranje međuoperabilnosti, Internet stvari, pasivno testiranje

1 INTRODUCTION

The Internet of Things (IoT) is a paradigm that is rapidly gaining ground in the field of modern wireless telecommunications. It is a vision towards future Internet, where smart objects such as sensors, Radio-Frequency IDentification (RFID) tags, mobile phones, etc. – are able to interact with each other and cooperate to reach common goals [3]. Promoted by Internet of Things, web services are nowadays ubiquitous. Devices behind these applications are typically battery powered and equipped with slow micro-controllers and small RAMs and ROMs. The data transfer is performed over wireless links with low bandwidth and high packet error rates. Unlike the Internet, there are high scalability requirements with trillions of nodes and the communication is often machine-to-machine (M2M) without human intervention.

To deal with these various challenging issues of IoT, the Constrained Application Protocol (CoAP) has been designed by Constrained RESTful Environments (CoRE) working group[1] to make it possible to provide resource constrained devices with low overhead and low power consumption web service functionalities and consequently to integrate smart objects with the web.

[1] http://datatracker.ietf.org/wg/core/charter/

As the number of smart objects using CoAP is expected to grow substantially, the heterogeneous nature of CoAP implementations requires interoperability. Although CoAP drafts [1, 4, 6, 7] have emerged as common specifications, they are not enough to guarantee the interoperability. It is a known fact that, even if following the same standard, two different devices might not be interoperable. Nevertheless, in the context of the IoT, interoperability is particularly important to guarantee the successful collaboration of smart objects.

In order that the CoAP interoperability issues be solved before the deployment of the implementations, ETSI[2] (the European Telecommunication Standard Institute), together with Probe-IT[3] (the European project in the context of Internet of things) organized the first formal CoAP plugtest in Paris, France, March 2012[4]. The objective of this paper is to present our testing procedure and experience for the CoAP Plugtest, which turned to be a successful event due to our testing method and tool. The paper is organized as follows: In section 2, the general overview CoAP protocol is presented. Section 3 introduces briefly the CoAP Plugtest event. Section 4 presents the CoAP interoperability testing, including the testing method and an associated automatic analysis test validation tool based on the technique of passive testing [9, 10]. The application of the testing approach and the experimental results are exhibited in section 5. Finally, section 6 gives the conclusions and the perspectives.

2 COAP PROTOCOL OVERVIEW

Most Internet applications today depend on the Web architecture, using HTTP (Hypertext Transfer Protocol) [8] to access information and perform updates. HTTP is based on Representational State Transfer (REST) [14], 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 applications, aiming at extending 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.

[2] www.etsi.org/

[3] http://www.probe-it.eu/

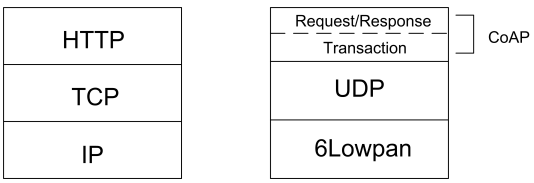
[4] http://www.etsi.org/plugtests/coap/coap.htm

The interaction model of CoAP is similar to the client/server model of HTTP. A CoAP request is sent by a client to request an action on a resource identified by a URI (Uniform Resource Identifier) on a server. The server then sends a response, which may include a resource representation. While HTTP targets for traditional IP networks, CoAP targets for resource constrained networks such as wireless sensor networks.

The architecture of CoAP is illustrated in Fig.1. Unlike HTTP over TCP, CoAP operates over UDP in order to lower the header overhead and parsing complexity as required by resource constrained devices. Within UDP packets, CoAP uses a short 4 bytes compact binary header followed by a sequence of options. Besides, the use of UDP also enables best-effort multicast.

CoAP uses a two-layer approach to support asynchronous transaction: *(i)* 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 (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) 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 request URI to be deleted.



Slika 1. Protocol stack of HTTP and CoAP

CoAP supports built-in resource discovery, which 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.

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 cacheability, 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[5] and Contiki[6], have already released their CoAP implementations. It is slated to become one of the most important ubiquitous application protocols for the future Internet of Things.

3 COAP PLUGTEST

To ensure that protocol implementations work properly and satisfy customer expectations, several testing methods such as *conformance testing* and *interoperability testing* [15](*iop* for short in the sequel) have been deployed to validate communication protocols before commercialization. Conformance testing [11] verifies whether a network component conforms to its specification. It allows developers to focus on the fundamental problems of their protocol implementations. However, although conformance testing increases the probability of interoperability, it cannot guarantee interoperability. Conforming products may not interoperate due to several reasons: poorly specified protocol options, incompleteness of conformance testing, etc. Nevertheless, with the rapid widespread commercial adoption of complex and diverse IoT technologies, interoperability is essential for M2M applications to provide cooperative services. Moreover, the large variety of protocols, carrying different features, need different interoperability testing events to verify their interoperability. To efficiently address these problems, the standards bodies and industry forums arrange regular workshops and interoperability testing events (e.g. IPSO Interoperability events[7], Tahi IPv6 Interoperability event[8], etc.), where vendors may test the interoperability of their equipments with other fellow industry participants.

As one of the most important protocol for the future Internet of Things, the application of CoAP is 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. In this context, ETSI and the European project Probe-IT, together with IPSO Alliance organized the first formal CoAP plugtest in Paris, France in March 2012. The objectives of this events are to get-together industry people to share their experiences, test their equipments in order to make their products successful in multi-vendor environment by achieving interoperability.

The work presented in this paper involves our main contribution to the plugtest event, including testing method and tool, which were successfully put into operation during the CoAP plugtest. To the best of our knowledge, there does not exist research works concerning CoAP interoperability testing. Moreover, The originality of the work consists in:

- 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 the method of passive testing. Passive testing is a testing technique based only on observation. Its non-intrusive nature makes it appropriate for interoperability testing in the context of IoT.
- As IoT implies providing services in lossy networks, we take into account interoperability testing of CoAP implementations in lossy context.
- Contrary to manual verification used in conventional interoperability testing events, the verification procedure has been automated by a test validation tool, which increases considerably the efficiency.

4 COAP INTEROPERABILITY TESTING – METHOD AND TOOL

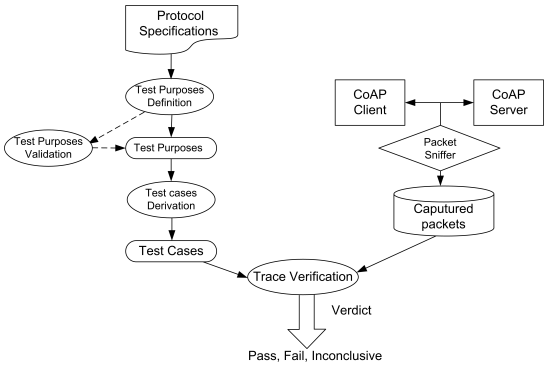
4.1 Testing method overview

In order to carry out CoAP interoperability testing, we have defined a methodology (c.f. Fig.2): Regarding the specifications of CoAP ([1,4,6,7]), a set of interoperability test purposes have been selected. Each test purpose represents a critical property of CoAP that needs 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.

Our testing method is based on 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

[5] http://www.tinyos.net/
[6] http://www.sics.se/contiki/
[7] http://www.ipso-alliance.org/category/events
[8] http://www.tahi.org/inop/6thinterop.html

resource-limited context of IoT. During the test, packets exchanged between CoAP implementations (CoAP client and CoAP server) are captured by a packet sniffer. Captured traces are analyzed against the test cases by using a passive testing tool to evaluate the behavior of the implementations w.r.t the given test purpose. 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 in the specifications, however does not correspond to the test purpose.



Slika 2. CoAP interoperability testing procedure

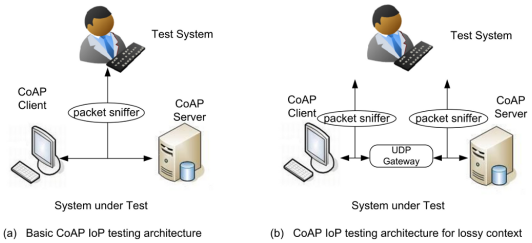
4.2 Testing architectures

Two test configurations are defined and adopted in the IoT CoAP Plugtest for different purposes. The basic test architecture is illustrated in Fig.3-(a). It involves a *Test System (TS)* and a *system under test (SUT)* composed of 2 *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 possible packet loss, we also need to consider testing the interoperability of CoAP applications in lossy environment. The corresponding architecture is shown in Fig.3-(b): A UDP gateway is used in-between the client and server to emulate a lossy medium.

4.3 CoAP interoperability test purposes

The first step to perform CoAP interoperability testing is to define a set of CoAP *interoperability test purposes* (ITP). An ITP is in general a statement, representing a critical property to be tested [5]. It allows focuing on the most important properties of a protocol, especially the specifications of CoAP are still under work with continuous extension. To ensure that the ITPs are correct w.r.t the specification, the ITPs were chosen and cross-validated by the experts from ETSI[9], IRISA[10] and Beijing University of Post and Telecommunication[11], and reviewed by IPSO alliance to guarantee the correctness.



Slika 3. CoAP interoperability testing architectures

Regarding the specifications of CoAP protocol, the following aspects are considered, which cover the most important properties of CoAP:

- The basic CoAP protocol methods [1].
- CoRE Link Format for resource discovery [7].
- CoAP Block Transfer for transferring large resources [6].
- CoAP Observation for pushing resource representations from servers to interested clients [4].

Concerning these aspects, a set of 27 test purposes were defined. For each test purpose, a test scenario (test case) is derived, describing in detail the behavior of the IUTs to be verified.

4.3.1 Basic CoAP methods

This group of tests contains 16 test purposes that aim at testing the basic transaction of the CoAP request/response model in both reliable and lossy environment.

The fundamental CoAP methods interoperability tests involve verifying that both client and server interact correctly according to [1] by using any of the methods GET, POST, PUT, and DELETE. Specifically, it requires to verify that each request sent by the client contains the correct method code and message type code (CON or NON). Upon the reception of a request, 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.

[9] http://www.etsi.org/WebSite/homepage.aspx
[10] http://www.irisa.fr/
[11] http://www.bupt.edu.cn/

Sometimes however, a server cannot obtain immediately the resource requested by the client. In this case, the server will first send an acknowledgment with 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.4-(b)).

Based on basic transactions, we have also chosen three important options: *(i) Token* option: is used for request/response matching in asynchronous communication. Specifically, every request that has a client-generated token must be echoed in any response. *(ii)* CoAP uses URI schemes for identifying CoAP resources and locating resources. In this work we choose to test that: for a client's request containing *URI-Path* or *URI-Query* options, the server must be able to send a response message with the correct message type and code in corresponding to the previous request, as well as the requested resource.

4.3.2 Link format

Link format is standardized in [7] for CoAP applications to realize resource discovery. The path prefix for resource discovery is defined as */.well-known/core*, which can be accessed with a GET request on that URI. 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*[12] would request only resources with the name *Temperature*.

4.3.3 Block-wise transfer

CoAP is based on datagram transports, which limits the maximum size of resource representations (64 KB) that can be transferred. In order to handle large payloads, [6] 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.

This group of test purposes contains 4 tests that check the main functionalities of CoAP block-wise transfer: *(i)* If the client knows the large resource that it requires, it sends a GET request containing *Block* option, indicating block number and desired block size. In return, the server sends a response containing the requested block number and size. The transaction repeats until the client obtains the whole resource. *(ii)* If a response generated by a resource handler exceeds the client's requested block size, the server automatically divides the response and transfer it in several blocks. *(iii)* Similarly, block options also make it possible for the client to update or create a large size resource on the server, by using PUT and POST request respectively. (cf. an example in Fig.4-(c))

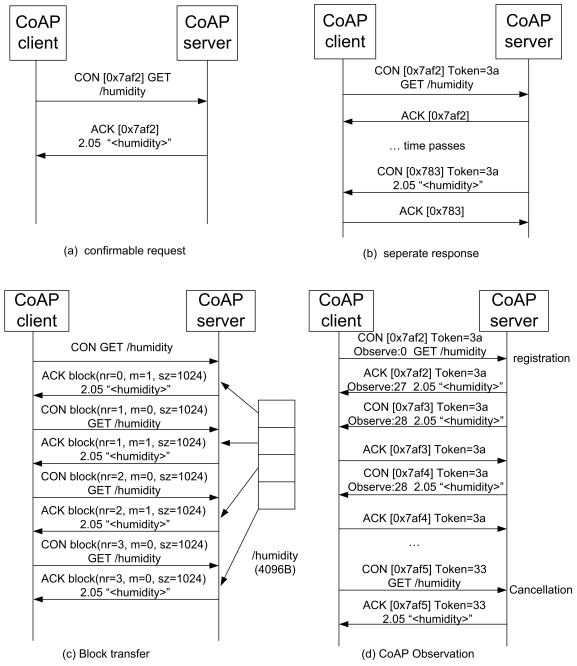
4.3.4 CoAP Observe

As the representation of a resource on a server may change from time to time, [4] defines a mechanism *CoAP Observe*, an asynchronous approach to push 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 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.4-(d).

4.3.5 Examples

The following figure demonstrates some typical examples of CoAP transactions. Fig.4-(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.4-(b) is an example of a separate response: When a CoAP server receives a request which it is not able to handle immediately, it first acknowledges the reception of the message by an acknowledgment with empty payload, and later sends back a confirmable response in a separate manner. Fig.4-(c) illustrate a block-wise transfer of a large payload (humidity) requested by the client. Upon the reception, the server divides the resource into 4 blocks and transfer them separately to the client. Fig.4-(d) illustrates an example of resource observation, including registration and cancellation. At first, the client register 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.

[12] *rt*: Resource type attribute. It is a noun describing the resource.



Slika 4. CoAP transaction examples

4.3.6 CoAP interoperability testing in lossy context

As CoAP protocol is designed for constrained networks and nodes for M2M applications, where devices are generally low power, in consequence many packet losses will occur. Therefore, an important aspect is to show that CoAP application should still interoperate correctly even in lossy context.

In CoAP Plugtest, we have chosen to test basic CoAP confirmable GET method in lossy context: If a client sends a request for a resource on a server, the server sends a piggybacked response, or a separate response if it needs more time to obtain the resource. As it involves lossy medium, both request and response may lost. The client and server must correctly retransmit the request and response if they are lost.

4.4 CoAP interoperability test cases derivation

For each test purpose, a test case is derived. In our work, each test case specifies the events that can lead to different verdicts: *(i)* the expected events to be observed that allow satisfying the test purpose and consequently lead to *Pass* verdict. *(ii)* Unexpected events that lead to *Fail* verdict. *(iii)* Other behavior allowed by the specifications but cannot satisfy the test purpose, thus leads to *Inconclusive* verdict. The assignment of different verdicts is done by studying carefully the specifications, and cross validated by ETSI, IRISA, BUPT and IPSO Alliance. The test purposes and test cases are described in natural language. An example is given in Fig.5. Notice that due to space limitation, in the description of test case we only list the expected events that lead to *Pass* verdict. But behavior that leads to all kinds of verdicts *Pass*, *Fail* and *Inconclusive* are implemented (c.f. the buttom of Fig.5) for the sake of trace verification, which will be described in the next section.

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()), ))
```

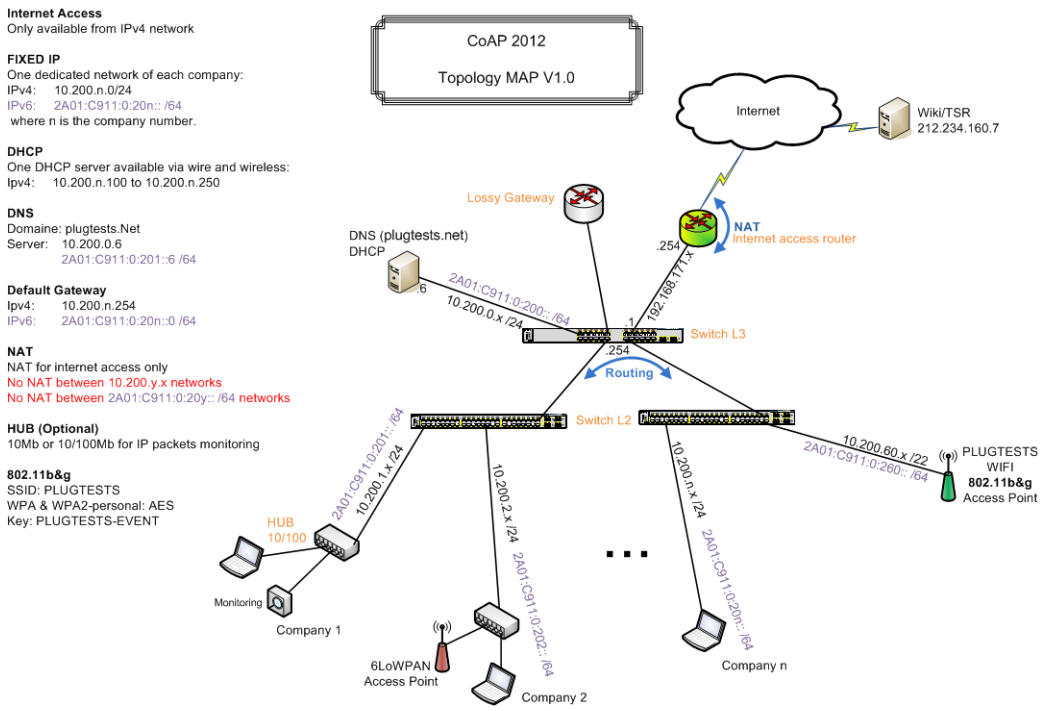
Slika 5. Test case example

4.5 Interoperability test execution: trace verification

In order to carry out CoAP interoperability testing, we have proposed a testing method based on the technique of passive testing. The packets exchanged (traces) between the CoAP client and server are captured by a packet sniffer and stored in a file. They are key to conclude whether CoAP devices interoperate.

In passive testing, one issue is that the test system has no knowledge of the state where the system under test can be in w.r.t a test case at the beginning of the trace due to the uncontrollable nature of passive testing.

In order to handle this problem while realizing the trace analysis, we propose a solution. In fact, each CoAP test case contains the requests and responses made between CoAP client and server, and always starts with a request from the client. Therefore, a strategy is to filter the trace, keeping only the *request-response conversations* made between CoAP applications under test. Then, the next step is try to map the test cases into the conversations to identify whether they occur in the trace. If it is the case, an appropriate verdict will be emitted by comparing the events specified in the test case and those in its corresponding conversation(s) produced during the test. In detail, it involves checking each event taken in order from the filtered trace according to the following rules until the end of the trace.



Slika 6. CoAP Plugtest test bed

1. If the currently checked message is a *CoAP request*, we check whether it corresponds to the first message of (at least one of) the test cases in the test suite. If it is the case, we call these test cases *candidate test cases* and keep track of them. Moreover, the currently checked state in each candidate test case is kept in memory (noted *Currently_checked_state*).

2. If the currently checked message is a *CoAP response*, we check if this response corresponds to an event of each candidate test case at its currently checked state (stored in *Currently_checked_state*). If it is the case, we further check if this response leads to a verdict *Pass*, *Fail* or *Inconclusive* specified in the test case. 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 in the corresponding candidate test case, which can be reached by the currently checked message in the trace. On the contrary, if the response does not correspond to any event at the currently checked state in a candidate test case, we remove this test case from the set of the candidate test cases.

In order to emit an appropriate verdict for each test case, during the trace examination procedure described above, verdict assignment management is also performed and described below:

1. Each test case is associated with a counter. 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 for each test case is initially set to zero. Each time a verdict is emitted for a test case, the counter increments by 1. Also, a verdict emitted for a test case each time when it is met is recorded. All these obtained sub-verdicts will help further assign a global verdict for this test case.

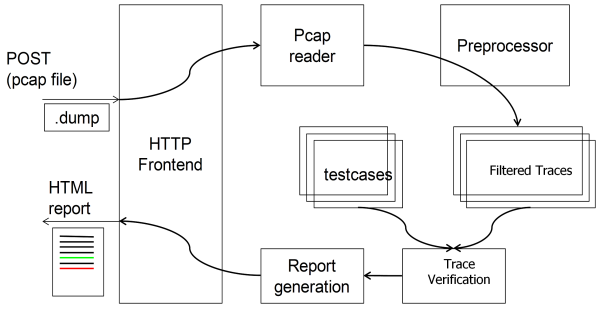
2. The global verdict for a test case is emitted by taking into account all its sub-verdicts. A global verdict is *Pass* if all its sub-verdicts are *Pass*. *Inconclusive* if at least one sub-verdict is *Inconclusive*, but no sub-verdict is *Fail*. *Fail*, if at least one sub-verdict is *Fail*.

The objective of this algorithm is to map each test case with its corresponding conversation(s) made between CoAP applications. And, if the occurrence of a test case is verified, a verdict is emitted, helping determining interoperability.

To realize trace verification, we have developed a passive testing tool, which aims to automate the process of trace verification. A description of this tool is given in Fig.6.

The tool is implemented in language Python3[13] mainly

[13] http://www.python.org/getit/releases/3.0/



Slika 7. Passive testing trace verification tool

for its advantages: easy to understand, rapid prototyping and extensive library. The tool is influenced by the widely used testing tool TTCN-3 [16]. 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.6, a web interface (HTTP frontend) was developed. Traces produced by client and server implementations of a protocol, captured by the packet sniffer are submitted via the interface. Specifically in our work, the traces should be submitted in pcap format[14]. Each time a trace is submitted, it is then dealt by a preprossesor to filter only the messages relevant to the CoAP 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 the trace analysis algorithm, 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.

5 EXPERIMENTATION IN COAP PLUGTEST

The interoperability approach proposed for CoAP protocol was applied in the CoAP Plugtest event held in Paris. It was the first formal 2 day event held for CoAP protocol in the scope of IoT, co-organized by ETSI, with technical testing support of our team IRISA and BUPT on behalf of the European project Probe-IT[15], together with IPSO Alliance. The objective of the CoAP Plugtest is to enable CoAP implementation developers to test end to end interoperability with each other. Also, it is an opportunity for standards development organization to revise the ambiguities in the protocol specifications. 15 developers and vendors of CoAP implementations, such as Sensinode[16], Watteco[17], Actility[18], etc. participated in the event. Test sessions are scheduled by ETSI so that each participant can test their products with all the other partners.

During the test event, CoAP implementations from different vendors are interconnected in pair-wise combinations. Figure 7 shows the test bed architecture provided by ETSI for this event. Each company was given with a hub to connect their implementations in the test bed. Communication were routed using layer 2 and layer 3 switches as shown in the figure 7.

A total of 27 test cases, concerning the basic CoAP methods, Link format, Observation and Blockwise transfer of CoAP protocol were served as test reference. Interoperability testing in lossy context was also realized by implementing the UDP gateway (by BUPT) as shown in Fig. 3-(b). In this configuration, a UDP gateway is used in-between client and server to emulate a lossy medium. The gateway does not implement the CoAP protocol itself (It is not a CoAP proxy), but works at the transport layer. The gateway plays the following roles:

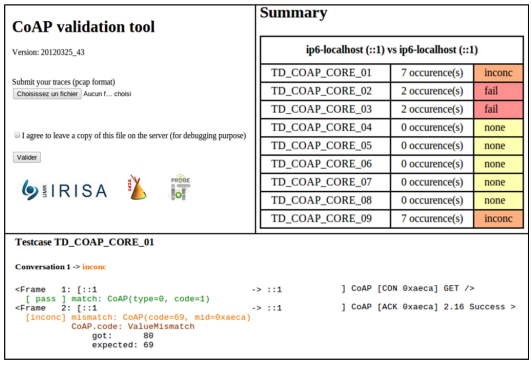
- It performs NAT-style UDP port redirection towards the server (thus the client contacts the gateway and is transparently redirected towards the server).
- It randomly drops packets that are forwarded between the client and the server. In Plugtest, the gateway drops the packet randomly between client and server which goes more than 50 % packet loss, corresponding to the high loss rate in IoT environment.

5.1 Passive trace verification

During the test, the tool Wireshark[19] 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. A use case of the tool is as follows:

[14] http://www.tcpdump.org/

[15] http://www.probe-it.eu/
[16] http://www.sensinode.com/
[17] http://www.watteco.com/
[18] http://www.actility.com/
[19] http://www.wireshark.org/



Slika 8. Passive testing trace verification tool

Figure.8 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.8). 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*.

5.2 Results

The CoAP plugtest was a success with regards to the number of executed tests (3143) and the test results (shown in the sequel), as announced by ETSI. 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, 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[20], 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 interoperability 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. the following table): 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.

At present, we have used this tool to analyze traces against the fondamental requirements of CoAP protocol.

[20] http://www.etsi.org/plugtests/ims2/About_IMS2.htm

	Executed Tests	Non-interoperable
CoAP 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%)

Tablica 1. CoAP Plugtest Results

Works that deal with other test objectives are on progress with the modification of CoAP standard. Future work will also consider improve the test tool. e.g. online trace verification to monitor the network for a long time and report abnormalities at any time. And graphical test case presentation instead of imperative code will be considered.

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.

Literatura

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

[2] W. Colitti, K. Steenhaut, and N. De Caro. Integrating Wireless Sensor Networks with the Web, in Extending the Internet to Low power and Lossy Networks (IP+SN 2011), 2011.

[3] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey, Comput. Netw., vol. 54, pp. 2787–2805, 2010.

[4] K. Hartke. Observing Resources in CoAP, draft-ietf-core-observe-04, 2012.

[5] S.Schulz, A.Wiles, and S.Randall. TPLan-A notation for expressing test purposes. ETSI, TestCom/FATES, LNCS 4581, pp.292-304, 2007.

[6] C. Bormann and Z. Shelby. Blockwise transfers in CoAP. draft-ietfcore- block-05, 2012.

[7] Z. Shelby. CoRE Link Format. draft-ietf-core-linkformat-09, 2011.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1999.

[9] F.Zaidi, A.Cavalli, and E.Bayse. Network Protocol Interoperability Testing based on Contextual Signatures. The 24th Annual ACM Symposium on Applied Computing SAC'09, Hawaii, USA, 2009.

[10] D.Lee, A.N.Netravali, K.K.Sabnani, B.Sugla, and A.John. Passive testing and applications to network management. In International Conference on Network Protocols, ICNP'97, pages 113-122. IEEE Computer Society Press, 1997.

[11] ISO. Information Technology - Open System Interconnection Conformance Testing -Methodology and Framework, Parts 1-7. International Standard ISO/IEC 9646/1-7,1994.

[12] A.Sabiguero, A.Baire, A.Boutet and C.Viho. Virtualized Interoperability Testing: Application to IPv6 Network Mobility. 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2007, Proceedings; 01/2007, 2007.

[13] R.M.Fuhrer. Sequential Optimization of Asynchronous and Synchronous Finite-State Machines, Ph.D. thesis, Department of Computer Science, Columbia University, 1999.

[14] R.T.Fielding. Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, Irvine, 2000.

[15] A.Desmoulin, C.Viho. Automatic Interoperability Test Case Generation Based on Formal Definitions. Lecture Notes in Computer Science, vol 4916, pp. 234-250, 2008.

[16] A.Baire, C.Viho, and N.Chen. 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.

Nanxing Chen is currently a PHD student in the research center IRISA, Rennes, France. Her research interests focus on protocol interoperability testing, especially passive testing.

César Viho received his PhD diploma in computer sciences from University of Bordeaux (France) in 1991. Since 1992, he joined the computer sciences department ISTIC of University of Rennes 1 (France), where he obtained his HdR (*Habilitation à diriger les Recherches*) and a Professor position in 2006. His research activity is done in IRISA laboratory. Currently, he is head of the Network, Telecommunication and Services research department. His interests include interoperability testing, resource management in wireless networks and real-time multimedia delivery over IP-based networks. He participated in several European and international projects and he published several papers on those topics.

Anthony Baire is a R&D engineer in the IRISA research laboratory at the University of Rennes since 2004. He has been involved in the development of testing tools and in the organisation of interoperability events.

Xiaohong Huang received her B.E. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2000 and Ph.D. degree from the school of Electrical and Electronic Engineering (EEE), Nanyang Technological University, Singapore in 2005. Since 2005, Prof. Huang has joined Network and Information Center at Beijing University of Posts and Telecommunications. She is now an active research staff in Network and Information Center in Beijing University of Posts and Telecommunications. She has involved in several research projects, including National 973 project, NSFC, Eu FP6-GO4IT, CNGI-CERNET2, CNGI-Access Grid, and so on. Her current research interests are Future Internet architecture, Network Traffic Identification, Network Measurement, Protocol testing and so on.

Jiexi Zha is currently the master student at Beijing University of Posts and Telecommunications, China. His research interests include protocol engineering, conformance testing and interoperability testing.

AUTHORS' ADDRESSES

Nanxing Chen,
Prof. César Viho, Ph.D.
Anthony Baire,
IRISA (Institute for research in computer science and random systems),
University of Rennes,
Campus de Beaulieu - 263 avenue du Général Leclerc, 35042 Rennes cedex
email: Cesar.Viho@irisa.fr

Xiaohong Huang, Ph.D.
Jiexi Zha,
Network and Information Center,
Beijing University of Posts and Telecommunications,
No 10, Xitucheng Road, Haidian District, Beijing, PRC, 100876

Received: 2012-11-10
Accepted: 2013-03-22