Abstract—Testing a protocol at runtime in an online way is a complex and challenging work. It requires the ability to handle numerous messages in a short time, and also requires the same offline testing preciseness. Meanwhile, since online testing is a long term continuously process, the tester has to undergo severe conditions when dealing with large amount of nonstop traces. In this paper, we present a novel logic-based online passive testing approach to test, at runtime, the protocol conformance and performance through formally specified properties with new definitions of verdicts. In order to evaluate and assess our methodology, we experimented our approach with several Session Initiation Protocol properties in a real IP Multimedia Subsystem environment. Relevant verdicts and discussions are provided.

I. INTRODUCTION

Testing is a crucial activity in the evaluation process of a system or an implementation under test (IUT). Among the commonly applied approaches, the passive testing techniques (also called monitoring) are today gaining efficiency and reliability [9]. These techniques are divided in two main groups: online and offline testing approaches. Offline testing computes test scenarios before their execution on the IUT and gives verdicts afterwards, while online testing provides continuously testing during the operation phase of the IUT.

With online testing approaches, the collection of traces is avoided and the traces are eventually not finite. Indeed, testing a protocol at runtime may be performed during a normal use of the system without disturbing the process. Several online testing techniques have been studied by the community in order to test systems or protocol implementations [14], [10], [2]. These methods provide interesting studies and have their own advantages, but they also have several drawbacks such as the presence of false negatives, space and time consumption, often related to a needed complete formal model, etc. Although they bring solutions, new results and perspectives to the protocol and system testers, they also raise new challenges and issues. The main ones are the non-collection of traces and their on-the-fly analysis. The traces are observed (through an interface and an eventual sniffer) and analyzed on-the-fly to provide test verdicts and no trace sets should be studied a posteriori to the testing process. In our work, we present a novel formal online passive testing approach applied at runtime to test the conformance and performance of the IUT.

We herein extend our previous proposed methodology [3], [4] that presented a passive testing approach for checking the requirements of communicating protocols. In [3] and [4], a formalism was applied to test in an offline way the conformance and performance of an IUT. In this new paper, we develop our approach to test these two aspects in an online way in considering the above mentioned inherent constraints and challenges. Furthermore, our framework is designed to test them at runtime, with new required verdicts definitions of 'Time-Fail', 'Data-Inc' and 'Inconclusive' representing unobserved message within timeout, untested data portion and uncertain status respectively. Finally, to demonstrate the efficiency of our online approach, we apply it on a real communicating environment for assessing its preciseness and efficiency.

Our paper’s primary contributions are:

– We provide a formal online passive testing approach to avoid stopping the execution of the testing process when monitoring a tested protocol. The analyzed traces are never cut which improves the accuracy of the test verdicts.

– Our approach allows the testing process to be executed in a transparent way without overloading the CPU and memory of the used equipment on which the tester will be run.

– Data portion of the messages is taken into account in our online testing approach, and new definitions of online testing verdicts are introduced.

II. RELATED WORKS

When studying the literature, we note that there are very few papers tackling online passive testing. We can however cite the following ones.

In [16], the authors proposed two online algorithms to detect 802.11 traffic from packet-header data collected passively at a monitoring point. They built a system for online wireless traffic detection using these algorithms. Besides, some researchers presented a tool for exploring online communication and analyzing clarification of requirements over the time in [8]. It supports managers and developers to identify risky requirements. We should also cite the works [11], [15] from which an industrial testing tool has been developed. These works are based on formal timed extended invariant to analyze runtime traces with deep packet inspection techniques. However, while most of the functional properties can be easily designed, complex ones with data causality can not. Though their approach is efficient with an important data flow, the process is still offline. To be complete, we have to mention that studies have also been performed to generate invariant from model-checkers. However, it requires a formal model and it still raises unresolved issues [6].

Inspired from these above cited works, we propose an online formal passive testing approach by defining functional properties of IUT, without modeling the complete system, and by considering eventual false negatives. For this latter, we introduce a new verdict 'Time-Fail' for distinguishing the real faults and the faults caused by timeouts. In addition, since online protocol testing is a long-term continuously testing process, we provide a temporary storage for remaining the integrity of incoming traces. Further, our approach provides the ability to test both the data portion and control portion, accompanying with another new verdict 'Data-Inc'.
III. ONLINE TESTING APPROACH

In this section, we describe the architecture and testing process of our online testing approach. We also provide the new definitions of online testing verdicts.

A. Architecture of the approach

In our approach, the Horn logic [7] is used for formally expressing properties as formulas. This logic has the benefit of allowing the re-usability of clauses. And it provides better expressibility and flexibility when analyzing protocols. A syntax tree generated from the formulas will be used for filtering incoming traces and optimizing evaluation processes. For the evaluation part, we use the SLD-resolution algorithm for evaluating formulas. The architecture of our online testing approach is illustrated in Figure 1.

![Architecture of our online testing approach](image)

B. Testing Process

The testing process consists of 8 parts (Figure 1): Formalization, Construction, Capturing, Generating Filters/Setup, Filtering, Transfer/Buffering, Load Notification and Evaluation.

**Formalization:** Initially, informal protocol requirements are formalized using Horn-logic based syntax. Due to the space limitation, we will not go into details. The interested readers may have a look at the works [3] and [9]. Then the verdicts {"Pass", "Fail", "Time-Fail", "Inconclusive", "Data-Inc"} are provided to the interpretation of obtained formulas on real protocol execution traces.

**Construction:** From formalized formulas, a syntax tree is constructed for further testing processes. In this process, each formula representing a requirement will be transformed to an Abstract Syntax Tree (AST) using the TREEGEN algorithm [12]. The standard BNF representation of each formula is the input to construct an AST. All the generated ASTs are finally combined to a syntax tree using a fast merging algorithm [1]. The syntax tree will be transferred to the tester as requirements and will be used to filter the captured traces.

**Capturing:** The monitor consecutively captures traces of the protocol to be tested from points of observations (P.Os) of the IUT, until the testing process finishes. When messages are captured, they are tagged with a time-stamp \( t_m \) in order to test the properties with time constraints and to provide verdicts on the performance requirements of the IUT.

**Generating Filters and Setup:** Once the syntax tree is constructed, it will be applied to captured traces for playing the role of a filter. Meanwhile, the tree will also be sent to the tester with the definition of verdicts. According to different conditions, verdicts are defined as below:

a) **PASS:** The message or trace satisfies the requirements.

b) **FAIL:** The message or trace does not satisfy the requirements.

c) **TIME-FAIL:** The target message or trace cannot be observed within the maximum time limitation. Since we are working on online testing, a timeout is used to stop searching target message in order to provide the real-time status. The timeout value should be the maximum response time written in the protocol standard. If we cannot observe the target message within the timeout time, then a Time-Fail verdict will be assigned to this property. It has to be noticed that this verdict is only provided when no time constraint is required in the requirement. If any time constraint is required, the violation of this requirement will be concluded as Fail.

d) **INCONCLUSIVE:** Uncertain status of the properties. Different from offline testing, this verdict will not appear in the final results. It only exists at the beginning of the test or when the test is paused, in order to describe the indeterminate state of the properties (e.g. a property that requires a special occurrence on the protocol that did not occur yet).

e) **DATA-InC** (Data Inconclusive): In the testing process, some properties may be evaluated through traces containing only control portion (there is no data portion or the latter case mentioned in Step ‘Transferring’). If any property requires for testing the data portion, Data-Inc verdicts will be assigned to the property, due to the fact that no data portion can be tested. However, these Data-Inc verdicts will be eventually updated to Pass or Fail based on the data (coming from complete traces) analyzed on the tested properties. Currently we are using worst-case solution (all concluded as Fail verdicts). It won’t affect the overall results, since Data-Inc verdicts only represent a tiny proportion (less than 0.1%) of the whole traces in our experiments. However, expecting eventual contingencies, we plan to apply a support vector machine (SVM) approach [5] in the future.

**Filtering:** The incoming captured traces will go through the filtering module, and messages in the traces are filtered into different sets. The unnecessary messages irrelevant to any of the requirements are filtered into the “Unknown” set, and they will not go through the testing process. Finally, traces will be filtered to multiple optimized streams. This step will obviously reduce the processing time, since futile comparisons with irrelevant messages are omitted.

**Transferring:** The filtered traces are transferred (6a) to the tester when the tester is capable for testing. If the tester priority has to be decreased (e.g. the CPU and RAM must be used for another task on the user computer), a “load notification” (7) is provided to the monitor to transfer/store incoming traces.
Based on the message format of the protocols to be tested, different buffering methods will be applied.

- If in the message format, the size of its header is larger than its body. Then the whole message will be buffered in the temporary storage.

- On the contrary, if the size of its header is equal or less than its body, then only the control portion of the packets are buffered (6b) in the temporary storage. Since not all the protocol requirements have specific needs on the data portion, only buffering the control portion will save a lot of memory space when buffering millions of messages.

When the tester is available (notification obtained), the stored traces are retransferred (6c) to the tester. In the latter case mentioned above, only the control portion of packets are provided. In both cases, the continuity of traces is ensured, since no packet will be dropped in any condition. If the protocol requirement has specific needs on the data portion, then the new verdict Data-Inc can be given and will be eventually updated to final verdicts by future analysis with the entire traces (the tester is indeed available again).

**Load Notification:** When the tester reaches its limit regarding the amount of data processable or is given a lower priority (e.g. to discharge the CPU / RAM), it sends a "Load Notification Y" to pause incoming filtered traces and store them in the temporary storage. When the tester is available back, a "Load Notification X" to release stored traces and to pursue incoming packets is sent. When captured traces from the IUT are transferred to the tester buffer, a checking overflow function will be called. If the buffer already reached to its maximum capacity, it will notify the IUT to redirect incoming traces to temporary storage in order to avoid the overflow. On the contrary, if the buffer is in a stable condition, it will send the available notification N to the temporary storage for releasing stored messages and to the IUT for returning back to normal transport process.

**Evaluation:** The tester checks whether the incoming traces satisfy the formalized requirements, and provides the final verdicts Pass, Fail or Time-Fail and temporary verdicts Inconclusive or Data-Inc.

**IV. Experiments**

**A. Environment**

The IP Multimedia Subsystem (IMS) is a standardized framework for delivering IP multimedia services to users in mobility. It aims at facilitating the access to voice or multimedia services in an access independent way, in order to develop the fixed-mobile convergence. Most communication with its core network and between the services is done using the Session Initiation Protocol (SIP) [13]. For our experiments, communication traces were obtained through four ZOIPER1 clients which are VoIP soft clients, meant to work with any IP-based communication systems and infrastructure. On the other side, the server is provided by Fonality2. The tests are performed in the virtual machines by opening a live capture on the client local interface.

**B. Test Results**

For better understanding how our approach works, we illustrate a simple use case tested on one of the clients. As shown in Figure 2, we have a SIP requirement to be tested: “Every 2xx response for INVITE request must be responded with an ACK within 2s”, which can be formalized to a formula as Step 1 shows.

This formula will be transformed to a syntax tree. When the syntax tree is generated and transferred to the IUT monitor, it will start to capture the trace and apply the syntax tree as a filter (Step 3 and 4) for captured messages. Meanwhile, the syntax tree will be applied in the tester as requirement. Once the captured trace is filtered into different sets (Step 5), it will check the Load Notification value first. Currently, the Load Notification value equals to N, which makes the tester available to test incoming traces. Then all incoming traces will be sent to the tester directly (Step 6a). As soon as the tester receives the trace, it tests the trace through the formalized property. When the tester is almost reaching to its maximum capacity, it will send a load notification value Y back to the monitor (Step 7 and 8). In this case, all incoming traces will be stored in the temporary storage (Step 6b) until the tester recovers to

---

1http://www.zopier.com/softphone/
2http://www.fonality.com
<table>
<thead>
<tr>
<th>Properties</th>
<th>Total Messages</th>
<th>Filtered out Messages</th>
<th>Filtered out Rate</th>
<th>Pass</th>
<th>Fail</th>
<th>Time-Fail</th>
<th>Incon</th>
<th>Data-Inc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prop.1</td>
<td>2324506</td>
<td>1631797</td>
<td>70.19%</td>
<td>631271</td>
<td>0</td>
<td>61432</td>
<td>32</td>
<td>2164</td>
</tr>
<tr>
<td>Prop.2</td>
<td>2324506</td>
<td>1631797</td>
<td>70.19%</td>
<td>498124</td>
<td>194579</td>
<td>0</td>
<td>52</td>
<td>2164</td>
</tr>
<tr>
<td>Prop.3</td>
<td>2324506</td>
<td>1979904</td>
<td>85.17%</td>
<td>314923</td>
<td>0</td>
<td>29673</td>
<td>14</td>
<td>1086</td>
</tr>
<tr>
<td>Prop.4</td>
<td>2324506</td>
<td>1979904</td>
<td>85.17%</td>
<td>247257</td>
<td>97339</td>
<td>0</td>
<td>14</td>
<td>1086</td>
</tr>
<tr>
<td>Prop.5</td>
<td>2324506</td>
<td>2259032</td>
<td>97.18%</td>
<td>61550</td>
<td>0</td>
<td>3924</td>
<td>6</td>
<td>371</td>
</tr>
</tbody>
</table>

TABLE I. ONLINE TESTING RESULT FOR PROPERTIES

an available state (Step 6c). Finally, after our 2 hours testing process, we got 18864 ‘Pass’ verdicts, 5 ‘Fail’ verdicts caused by violation of the time constraint and no Time-Fail verdicts.

Secondly, we test our approach in a more complex environment. It has been performed to concurrently test five properties on a huge set of messages: “Prop.1: Every request must be responded”, “Prop.2: Every request must be responded within 8s”, “Prop.3: Every INVITE request must be responded”, “Prop.4: Every INVITE request must be responded within 4s” and “Prop.5: Every REGISTER request must be responded”.

The table I shows a snapshot of temporary testing verdicts after 3 hours online continuously testing. Benefited from the filtering function, more than 70% irrelevant messages are filtered out before testing process, which apparently reduce the cost of computing resources. Besides, numbers of Fail and Time-Fail verdicts can be observed. Time-Fail verdicts in Prop.1, Prop.3 and Prop.5 indicate that there are 61432, 29673 and 3924 messages respectively that cannot be observed within the timeout, in other words, they are lost during the communication between the client and the server. Besides, the ‘0’ Fail verdict indicates there is no error observed in the data portion for these three properties currently. On the other side, Fail verdicts reported in Prop.2 and Prop.4 indicate that there are 194579 and 97339 messages that cannot satisfy the time requirement. These Fail verdicts include the Time-Fail verdicts reported in Prop.1 and Prop.3, since lost messages also violate the time requirement.

Moreover, several ‘Inconclusive’ verdicts indicating the numbers of pending procedures for each property can be observed. We also used the control-portion-only buffering mechanism to test the usage of ‘Data-Inc’. All the buffered messages without data portion are successfully reported as ‘Data-Inc’ shown in Table I. Since they take a tiny proportion of whole traces (between 0.015% and 0.09%), we conclude them as Fail in the worst-case. During the whole testing process, our approach successfully handled this huge set of messages and did not suspend.

V. CONCLUSION

This paper introduces a novel online passive testing approach to test conformance and performance of network protocol implementation. It allows to formally define relations between messages and message data, and then to use such relations in order to define the conformance and performance properties. The evaluation of the property returns a Pass, Fail, Time-Fail, Inconclusive or Data-Inc result, derived from run-time traces. To verify and test the approach, several SIP properties are designed to be evaluated. Our methodology has been implemented into a real-time IMS communications environment, and results from testing several properties online have been obtained successfully. Consequently, applying our approach under billions of messages and extending more testers in a distributed environment is part of our future works. In that case, the efficiency and processing capacity of the approach will be scalably tested.

REFERENCES