An Online Passive Testing Approach for Communication Protocols

Jorge Lopez, Xiaoping Che and Stephane Maag
Institut Mines-Telecom/Telecom SudParis CNRS UMR 5157, Evry, France

Keywords: Online Testing, Passive Testing, Formal Methods.

Abstract: Testing a protocol at runtime in an online way is a complex and challenging work. It requires the same preciseness in conformance testing and efficiency in performance testing, where conformance testing is a functional test which verifies whether the behaviors of the protocol satisfy defined requirements, and performance testing is a qualitative and quantitative test which checks whether the performance requirements of the protocol have been satisfied under certain conditions. As a matter of course, it raises an interesting issue of converging these two kinds of testing by using the same formal approach, and applying the approach online. In this paper, we present a novel logic-based online testing approach to test the protocol conformance and performance through formally specified properties. In order to evaluate and assess our methodology, we developed a prototype and experimented it with a set of Session Initiation Protocol properties in a real IP Multimedia Subsystem environment. Finally, the relevant verdicts and discussions are provided.

1 INTRODUCTION

Testing is a crucial activity in the evaluation process of a system or an implementation under test (IUT). Among the well known and commonly applied approaches, the *passive* testing techniques (also called *monitoring* or *run-time verification*) are today gaining efficiency and reliability (Bauer et al., 2011). These techniques are divided in two main groups: *online* and *offline* testing approaches. Offline testing aims at collecting set of protocol traces while running (through interfaces, ports or points of observations (P.O)) and then checking some properties through these traces afterwards. Several model based offline testing techniques have been studied by the community in order to passively test systems or protocol implementations (Veanes et al., 2005; Lee and Miller, 2006; Raimondi et al., 2008; Cao et al., 2010). Nevertheless, though offline testing still raises many interesting issues (Lalanne and Maag, 2013), online testing approaches bring out these same issues plus the challenges that are inherent to online testing. Among these inherent constraints, we shall cite the *non-collection* of traces. Indeed, in passive online testing, the traces are observed (through an eventual sniffer), analyzed on-the-fly to provide test verdicts and no trace sets are studied a posteriori to the testing process. In our work, we focus on the online testing of an IUT using passive testing technique.

Testing an implementation of a protocol is often associated to the checking of its conformance and performance. Conformance testing is a functional test which verifies whether the behaviors of the protocol satisfy defined requirements. Performance testing is a qualitative and quantitative test which checks whether the performance requirements of the protocol have been satisfied under certain conditions. They are mainly applied to validate or verify the scalability and reliability of the system. Many benefits can be brought to the testing process if conformance and performance testing inherit from the same approach and can be applied online.

Our main objective is then to propose a novel passive online testing approach based on formal conformance and performance testing techniques (Che et al., 2012) (Che and Maag, 2013). Although some crucial works have been done in run-time conformance testing area (Bauer et al., 2011), they study run-time verification of properties expressed either in linear-time temporal logic (LTL) or timed linear-time temporal logic (TLTL). Different from their work focusing on testing functional properties based on formal models, our work concentrates on formally testing functional and non-functional properties without formal models, through real protocols in an online manner.

In this work, we firstly extend one of our previous proposed methodologies to present a passive testing approach for checking the conformance and performance requirements of communicating protocols. Furthermore, we design an online testing frame-

Lopez J., Che X. and Maag S..
An Online Passive Testing Approach for Communication Protocols.
DOI: 10.5220/0004885501360143
In *Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2014), pages 136-143
ISBN: 978-989-758-030-7

work to test these requirements in real-time, with new verdicts definitions of 'Pass', 'Fail', 'Time-Fail' and 'Inconclusive'. Finally, since several protocol conformance and performance requirements need to be tested in order to verify the efficiency of our online approach, we perform our approach in a real IP Multimedia Subsystem (IMS) communicating environment for assessing its preciseness and efficiency.

Our paper's primary contributions are:

- A formal approach is proposed for formally expressing the conformance and performance requirements, and data portions are taken into account.
- An online testing framework is designed for testing conformance and performance requirements in real-time, and new definition of testing verdicts are introduced.

The reminder of the paper is organized as follows. In Section 2, a short review of the related works are provided. In Section 3, a brief description of the syntax and semantics used to describe the tested properties is presented. In Section 4, we illustrate our online testing framework and relevant algorithms. Our approach has been implemented and experimented in Section 5. It has been performed through a real IMS framework to test Session Initiation Protocol (SIP) properties. The real-time communications of the IMS allow to evaluate our approach efficiently. Finally, we conclude and provide interesting perspectives in Section 6.

2 RELATED WORKS

While a huge number of papers are dedicated to online testing. In this section we present the prior works in these fields.

Model Based Online Testing. Model based testing is a crucial technique in the testing research domain. In (Larsen et al., 2004), the authors present T-UPPAAL– a tool for online black-box testing of real-time embedded systems from non-deterministic timed automata specifications. They describe a sound and complete randomized online testing algorithm and implement it by using symbolic state representation and manipulation techniques. They propose the notion of relativized timed input/output conformance as the formal implementation relation. Likewise, some other researchers describe a practical online testing algorithm that is implemented in the model-based testing tool called Spec Explorer (Veanes et al., 2005), which is being used daily by several Microsoft product groups. They formalize the model programs as interface automata, and use the interface automata for conformance testing. The conformance relation between a model and an implementation under test is formalized in terms of refinement between interface automata. Besides, in (Raimondi et al., 2008), the authors describe how timed automata can be used as a formalism to support efficient online monitoring of timeliness, reliability and throughput constraints expressed in web service SLAs. And they present an implementation to derive on-line monitors for web services automatically form SLAs using an Eclipse plugin and Apache AXIS handlers. The readers would notice that all these works are based on modeling the system by automata or timed automata, due to the convenience that the constructed models can be handled by existing verification tools. However, when the system can not be accessed or modeled, our work will be a complementary to these techniques since we do not need to formalize the system by any automata.

Online Conformance Testing. In the online testing area, there are lots of work focus on conformance testing. In (Cao et al., 2010), the authors present a framework that automatically generates and executes tests "online" for conformance testing of a composite of Web services described in BPEL. The proposed framework considers unit testing and it is based on a timed modeling of BPEL specification, and an online testing algorithm that generates, executes and assigns verdicts to every generated state in the test case. Nevertheless, in (Nguyen et al., 2012), the authors defined a formal model based on Symbolic Transition Graph with Assignment (STGA) for both peers and choreography with supporting complex data types. The local and global conformance properties are formalized by the Chor language in their works. The local properties are used to test behaviors of one isolated peer with respect to its specification model, while the global properties test the collaboration of a set of peers with respect to its choreography model. Inspired from all these works, our work does not require to model the IUT and tackles not only conformance requirements, but also the performance requirements.

indent Moreover, another similar work is provided by the authors of (Hallé and Villemaire, 2012). They presented an algorithm for the runtime monitoring of data-aware workflow constraints. Sample properties taken from runtime monitoring scenarios in existing literature were expressed using LTL-FO^{+}, an extension of Linear Temporal Logic that includes first-order quantification over message contents. Similarly to our work, data are a more central part of the definition of formulas, and formulas are defined with quantifiers specific to the labels. Although the syntax of the logic they used is flexible, it can quickly lose

clarity as the number of variables required increases. Our work improves on this by allowing to group constraints with clause definitions.

Online Performance Testing. Many studies have investigated the performance of online systems. A method for analyzing the functional behavior and the performance of programs in distributed systems is presented in (Hofmann et al., 1994). In the paper, the authors discuss event-driven monitoring and event-based modeling. However, no evaluation of the methodology has been performed. In (Dumitrescu et al., 2004), the authors present a distributed performance-testing framework, which aimed at simplifying and automating service performance testing. They applied Diperf to two GT3.2 job submission services, and several metrics are tested, such as *Service response time*, *Service throughput*, *Offered load*, *Service utilization* and *Service fairness*.

Besides, in (Wei et al., 2009), the authors propose two online algorithms to detect 802.11 traffic from packet-header data collected passively at a monitoring point. The algorithms have a number of applications in real-time wireless LAN management, they differ in that one requires training sets while the other does not. Moreover, in (Yuen and Chan, 2012), the authors present a monitoring algorithm SMon, which continuously reduces network diameter in real time in a distributed manner. Nevertheless, most of these approaches are based on monitoring techniques, they do not provide a formalism to test a specific performance requirement. Our approach allows to formally specified protocol performance requirements in order to check whether the real-time performance of the protocol remains as expected in its standard.

Although lots of works have been done in the online testing area. Inspired from and based on their works, our work is different from focusing on using model-driven techniques, evaluating the performance of the system. We concentrate on how to formally and passively test the conformance and performance requirements written in the standard. And also we are trying to converge the online conformance and performance testing by using the same formal approach.

3 FORMAL APPROACH

We will provide in this section basic definitions, syntax and semantics of our formalism which are necessary for the understanding of our approach.

3.1 Basics

A communication protocol message is a collection of data fields of multiple domains. Data domains are defined either as *atomic* or *compound* (Che et al., 2012). An *atomic* domain is defined as a set of numeric or string values. A *compound* domain is defined as follows.

Definition 1. A *compound* value v of length $n > 0$, is defined by the set of pairs $\{(l_i, v_i) \mid l_i \in L \wedge v_i \in D_i \cup \{\varepsilon\}, i = 1...n\}$, where $L = \{l_1, ..., l_n\}$ is a predefined set of labels and D_i are sets of values, meaningful from the application viewpoint, and called data domains. Let D be a Cartesian product of data domains, $D = D_1 \times D_2 \times ... \times D_n$. A *compound* domain is the set of pairs (L, d), where d belongs to D.

Once given a network protocol P, a *compound* domain M_p can generally be defined by the set of labels and data domains derived from the message format defined in the protocol specification/requirements. A *message* m of a protocol P is any element $m \in M_p$.

For each $m \in M_p$, we add a real number $t_m \in \mathbb{R}^+$ which represents the time when the message m is received or sent by the monitored entity.

Example 1. A possible message for the SIP protocol, specified using the previous definition could be

$m = \{(method, \textbf{'INVITE'}), (time, \text{'210.400123000'}),$
$(status, \varepsilon), (from, \text{'alice@a.org'}), (to, \text{'bob@b.org'}),$
$(cseq, \{(num, 7), (method, \textbf{'INVITE'})\})\}$

representing an INVITE request from *alice@a.org* to *bob@b.org*. The value of *time* '210.400123000' ($t_0 + 210.400123000$) is a relative value since the P.O started its timer (initial value t_0) when capturing traces.

A *trace* is a sequence of messages of the same domain containing the interactions of a monitored entity in a network, through an interface (the P.O), with one or more peers during an arbitrary period of time. The P.O also provides the relative time set $T \subset \mathbb{R}^+$ for all messages m in each *trace*.

3.2 Syntax and Semantics of Our Formalism

In our previous work, a syntax based on Horn clauses is defined to express properties that are checked on extracted traces. We briefly describe it in the following. Formulas in this logic can be defined with the introduction of terms and atoms, as it follows.

Definition 2. A *term* is defined in BNF as $term ::= c \mid x \mid x.l.l...l$ where c is a constant in some domain, x is a variable, l represents a label, and $x.l.l...l$ is called a *selector variable*.

Definition 3. A *substitution* is a finite set of bindings $\theta = \{x_1/term_1, ..., x_k/term_k\}$ where each $term_i$ is a *term* and x_i is a variable such that $x_i \neq term_i$ and $x_i \neq x_j$ if $i \neq j$.

Definition 4. An *atom* is defined as

$$\begin{aligned} A ::= & \overbrace{p(term, ..., term)}^{k} \\ & \mid term = term \\ & \mid term \neq term \\ & \mid term < term \\ & \mid term + term = term \end{aligned}$$

where $p(term, ..., term)$ is a predicate of label p and arity k. The *timed atom* is a particular atom defined as $\overbrace{p(term_t, ..., term_t)}^{k}$, where $term_t \in T$.

The relations between *terms* and *atoms* are stated by the definition of clauses. A *clause* is an expression of the form

$$A_0 \leftarrow A_1 \wedge ... \wedge A_n$$

where A_0 is the head of the clause and $A_1 \wedge ... \wedge A_n$ its body, A_i being *atoms*.

A *formula* is defined by the following BNF:

$$\begin{aligned} \phi ::= & A_1 \wedge ... \wedge A_n \mid \phi \rightarrow \phi \mid \forall_x \phi \mid \forall_{y>x} \phi \\ & \mid \forall_{y<x} \phi \mid \exists_x \phi \mid \exists_{y>x} \phi \mid \exists_{y<x} \phi \end{aligned}$$

where $A_1, ..., A_n (n \geq 1)$ are *atoms*, x, y represent for different messages of a trace and $\{<, >\}$ indicate the order relation of messages.

In our approach, while the variables x and y are used to formally specify the messages of a trace, the quantifiers commonly define "it exists" ($\exists$) and "for all" ($\forall$). Therefore, the formula $\forall_x \phi$ means "for all messages x in the trace, ϕ holds".

The semantics used in our work is related to the traditional Apt–Van Emdem–Kowalsky semantics for logic programs (Emden and Kowalski, 1976), from which an extended version has been provided in order to deal with messages and trace temporal quantifiers. Based on the above described operators and quantifiers, we provide an interpretation of the formulas to evaluate them to $\top$ ('*Pass*'), $\bot$ ('*Fail*') or '?' ('*Inconclusive*') (Che et al., 2012).

Then the truth values $\{\top, \bot, ?\}$ are provided to the interpretation of the obtained formulas on real protocol execution traces. However, different from offline testing, definite verdicts should be immediately returned in online testing. Which indicates that only $\top$ ('*Pass*') and $\bot$ ('*Fail*') should be emitted in the final report, and the indefinite verdict '?' ('*Inconclusive*') will be used as temporary unknown status, but finally must be transformed to one of the definite verdicts.

4 ONLINE TESTING FRAMEWORK

In this section, we will introduce our novel online testing framework and provide the relevant algorithm for testers.

4.1 Framework

For the aim of testing conformance and performance requirements in an online way, we design and use a passive online testing architecture. As Figure 1 depicts, the testing process consists of five following parts.

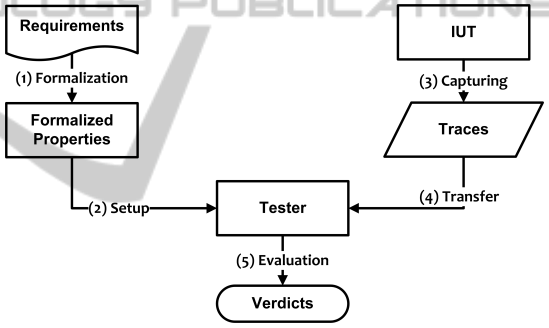

Figure 1: Core functions of our testing framework.

1. **Formalization:** Initially, the informal protocol requirements are formalized to formulas by using Horn-logic based syntax and semantics mentioned in section 3. Due to the space limitation, we will not go into details. The interested readers may have a look at the papers (Che et al., 2012) and (Lalanne and Maag, 2013).
2. **Setup:** When all the requirements are formalized to formulas, they will be sent to the Tester with the definition of verdicts.
   - Pass: The message or trace satisfies the requirement.
   - Fail: The message or trace does not satisfy the requirement.
   - Time-Fail: Since we are testing on-line, 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. If we can not

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 kind of verdict is only provided when no time constraint is required in the requirement. If any time constraint is required, the violation of requirements will be concluded as Fail, not as a Time-Fail verdict.

- Inconclusive: Uncertain status of the properties. It only exists at the beginning of the test or at the end of the test.

3. **Capturing:** The monitor consecutively captures the traces of protocols to be tested from the IUT. When the messages have been captured, each message will be tagged with a time-stamp in order to test the properties with time requirements.

4. **Transfer:** The tagged messages are transferred to the Tester when the Tester is capable for testing. Since we optimize our algorithm in order to have the best effort, the tester is always capable for testing when dealing with less than 20 million packages per minute.

5. **Evaluation:** The Tester checks whether or not the incoming traces satisfy the formalized requirements and provide the final verdicts. Based on different results and the definition of verdicts, we conclude the verdicts as: Pass, Fail or Time-Fail.

4.2 Testing Algorithm

The online testing algorithm is described in Algo 1. Algorithm 1 describes the behaviors of an online tester. Firstly, the tester will capture packets from the predefined interface by using libpcap[1], and it will tag time stamps to all the captured packets at the same time (Line 1-3).

Secondly, it will load all the properties (formalized requirements) have to be tested, and match each packet with the properties in chronological order. In this step, only the packets needed for the current property will be saved and tackled. The other irrelevant packets will be discarded in order to accelerate the testing process (Line 4-15). This process will keep running until all the properties have been checked.

When finishing the checking process, it will report the testing result and empty the buffer immediately in order to make good use of the limited memory (Line 16-29).

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

5 EXPERIMENTS

After introducing our novel framework, we will describe the testing environment and interpret the experiments results in this section.

5.1 Environment

The IP Multimedia Subsystem 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. The core of the IMS network consists on the Call Session Control Functions (CSCF) that redirect requests depending on the type of service, the Home Subscriber Server (HSS), a database for the provisioning of users, and the Application Server (AS) where the different services run and interoperate. Most communication with the core network and between the services is done using the Session Initiation Protocol (Rosenberg et al., 2002).

The Session Initiation Protocol is an application-layer protocol that relies on request and response messages for communication, and it is an essential part for communication within the IMS framework. Messages contain a header which provides session, service and routing information, as well as an body part to complement or extend the header information. Several RFCs have been defined to extend the protocol to allow messaging, event publishing and notification. These extensions are used by services of the IMS such as the Presence service and the Push to-talk Over Cellular (PoC) service.

For our experiments, communication traces were obtained through ZOIPER[2]. ZOIPER softphone is a VoIP soft client, meant to work with any IP-based communications systems and infrastructure. It provides secure high-quality voice calls and conference, fax sending and receiving functionality, and enhanced IP-calling features wrapped in a compact interface and small download size.

As Figure 2 shows, a simple environment is constructed for our experiments. We run two ZOIPER VoIP clients on the virtual machines using VirtualBox for Mac version 4.2.16. The virtual machines have 4GB of RAM, one processor Intel i5 @2.3GHz and the software being used is Zoiper 3.0.19649. On the other side, the server is provided by Fonality[3], which is running Asterisk PBX 1.6.0.28-samy-r115. The P.Os are placed on the client side. Tests are performed using a prototype implementation of the for-

[2]http://www.zopier.com/softphone/

[3]http://www.fonality.com

Algorithm 1: Algorithm of online tester.

```
Input: open_live_capture_on_interface(INTERFACE_NAME) //Using libpcap
Output: property verdicts report
thread_init(report_live_status) //thread to report the live
for each packet on live_capture do
    last_observed_packet_time ← get_time(packet);
    for each prototype on prototype_packets do
        property ← get_prototype_property(prototype);
        if match_properties_of(prototype, packet) then
            prototype_list ← get_prototype_list(prototype);
            for each prototype_dependency on dependencies(prototype) do
                matched_dependency ← FALSE;
                for each stored_packet on get_dependency_prototype_list(prototype_dependency) do
                    if match_properties_dependency(prototype_dependency, packet, stored_packet) then
                        associate(packet, stored_packet, property), matched_dependency ← TRUE;
                        goto next_dependency;
                    end
                end
                if !matched_dependency then
                    goto next_prototype
                end
            end
            if prototype_determines_property(prototype) then
                associations_list ← get_associations(packet) report_property_pass(property, packet, associations_list)
                delete_from_prototype_lists(associations_list)
            end
            else
                push(prototype_list, packet)
            end
        end
        next_prototype;
    end
end
```


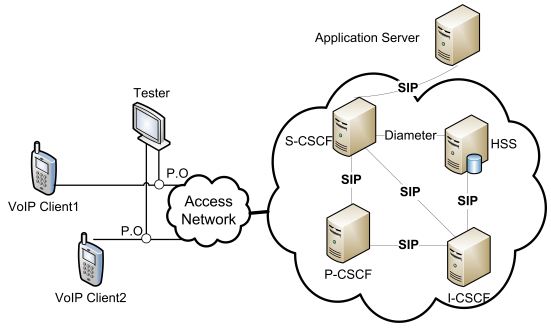

Figure 2: Environment for experiments.

mal approach above mentioned, using the algorithm introduced in the previous section.

5.2 Tests Results

In our approach, the conformance and performance requirement properties are formalized to formulas. These formulas will be tested through the testers in real-time. Simultaneously, not only '*Pass*', '*Fail*', '*Time-Fail*' and '*Inconclusive*' verdicts are returned, but also N_p, N_f, N_{tf} and N_{in} will be given to the tester, which represent the accumulated number of '*Pass*', '*Fail*', '*Time-Fail*' and '*Inconclusive*' verdicts respectively. We may write:

$$N_p(\phi) = \sum[eval(\phi,\theta,\rho) = \text{`}\top\text{'}]$$

$$N_f(\phi) = \sum[eval(\phi,\theta,\rho) = \text{`}\bot\text{'}]$$

$$N_{tf}(\phi) = \sum[\begin{array}{l} eval(\phi,\theta,\rho) = \text{`}\bot\text{'} \\ term_t \notin \phi, timeout \in \theta \end{array}]$$

$$N_{in}(\phi) = \sum[eval(\phi,\theta,\rho) = \text{`?'}]$$

Properties: In order to formally design the properties to be passively tested online, we got inspired from the TTCN-3 test suite (ETSI, 2004) and the RFC 3261 of SIP (Rosenberg et al., 2002). Several properties relevant to session establishment are designed.

Conformance requirements ϕ_1, ϕ_2 ("Every **INVITE** request must be responded", "Every successful **INVITE** request must be responded with a success response") and a performance requirement ψ_1

Table 1: Online Testing result for Client1 and Client2.

	ϕ_1				ϕ_2				ψ_1			
P.O	Pass	Fail	Time-Fail	Incon	Pass	Fail	Time-Fail	Incon	Pass	Fail	Time-Fail	Incon
Client1	50	0	0	0	43	0	7	0	645	240	0	0
Client2	36	0	0	0	34	0	2	0	473	6	0	0

("The response time for each request should not exceed $T_1 = 8s$") are tested. They can be formalized as the following formulas:

$$\phi_1 = \begin{cases} \forall_x(request(x) \wedge x.method = \textbf{'INVITE'} \\ \rightarrow \exists_{y>x}(nonProvisional(y) \wedge responds(y,x))) \end{cases}$$

$$\phi_2 = \begin{cases} \forall_x(request(x) \wedge x.method = \textbf{'INVITE'} \\ \rightarrow \exists_{y>x}(success(y) \wedge responds(y,x))) \end{cases}$$

$$\psi_1 = \begin{cases} \forall_x(request(x) \wedge x.method != \textbf{'ACK'} \\ \rightarrow \exists_{y>x}(nonProvisional(y) \wedge responds(y,x) \\ \wedge withintime(y,x,T_1))) \end{cases}$$

The two hours online testing results are illustrated in Table 1. A number of '*Fail*' and '*Time-Fail*' verdicts can be observed when testing ϕ_2 and ψ_1. Let's have a look at the '*Time-Fail*' verdicts in ϕ_2. They indicate that during the testing, the tester can not detect some successful responses to '**INVITE**' requests within the maximum response time. However, there is no time constraint required in ϕ_2, we have to conclude these verdicts as '*Time-Fail*'. Which means probably the server refused some '**INVITE**' requests, the responses are lost during the transmission or the responses will arrive later than the timeout time. Combining the results of ϕ_1, we can know that it was due to the first reason.

Similarly, we can observe some '*Fail*' verdicts when testing ψ_1, they are caused by the same reason that the tester can not find the target message within the required time. On the contrary, there is a specific time constraint required in ψ_1, which is 8 seconds. It shows some responses exceeded the required time 8s, they exactly violated the requirement and we have to conclude them as '*Fail*'.

These testing results successfully show that our approach can detect both the usual and unusual faults. Moreover, by using the verdicts obtained from these properties. We can also test the performance issues of session establishment in real-time, which can be defined as:

- Session Attempt Number: $N_p(\phi_1)$
- Session Attempt Rate: $N_p(\phi_1)$ / t_{slot}
- Session Attempt Successful Rate: $N_p(\phi_2)$ / $N_p(\phi_1)$
- Session establishment Number: $N_p(\phi_2)$
- Session establishment Rate: $N_p(\phi_2)$ / t_{slot}
- Session Packets Delay: $N_p(\psi_1)$.

Figure 3 illustrates the testing results of Session Attempt Number, Rate and Successful Rate in each hour. We can observe that in the 4th and 5th hour, the successful attempt rates are zero while the attempts numbers/rates are not, which denotes that in those two periods, all the session attempts ('**INVITE**' requests) were refused. And it returned to normal in the 6th hour. In this way, we can have a clear view of the protocol performance during online testing. It has to be noticed that when the tester receive an incoming trace, it will apply the formalized requirements on the trace and get the verdicts in a short time. During our experiments, all the testing results for requirements without time constraints are obtained in 1s, which proves the efficiency of our approach.

6 PERSPECTIVES AND CONCLUSION

This paper introduces a novel online approach to test conformance and performance of network protocol implementation. Our approach allows to define relations between messages and message data, and then to use such relations in order to define the conformance and performance properties that are evaluated on real protocol traces. The evaluation of the property returns a *Pass*, *Fail*, *Time-Fail* or *Inconclusive* result, derived from the given trace.

The approach also includes an online testing framework. To verify and test the approach, we design several SIP properties to be evaluated by our approach. Our methodology has been implemented into an environment which provides the real-time IMS communications, and the results from testing several properties online have been obtained successfully.

Furthermore, our approach can not only test requirements and return relevant verdicts, but also it can reflect current protocol performance status based on these verdicts. We extended several performance measuring indicators for SIP. As Figure 3 shows, these indicators are used for testing the performance of session establishment in SIP. The real time updated results displayed in the screen can precisely reflect the performance of the protocol in different time periods.

Consequently, extending more testers in a distributed environment based on the work (Che and

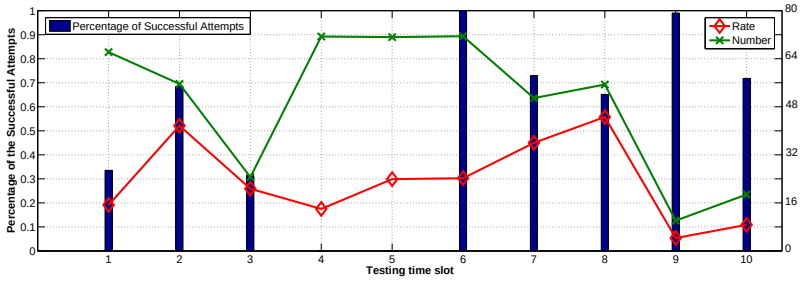

Figure 3: Session Attempt Number, Rate and Successful Rate.

Maag, 2013) and building an online testing system for all the network protocols would be the work we will focus on in the future. In that case, the efficiency and processing capacity of the system would be the crucial point to handle, leading to an optimization of our algorithms to severe situations.

REFERENCES

Bauer, A., Leucker, M., and Schallhart, C. (2011). Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology*, 20(4):14.

Cao, T.-D., Félix, P., Castanet, R., and Berrada, I. (2010). Online testing framework for web services. In *Third International Conference on Software Testing, Verification and Validation*, pages 363–372.

Che, X., Lalanne, F., and Maag, S. (2012). A logic-based passive testing approach for the validation of communicating protocols. In *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering, Wroclaw, Poland*, pages 53–64.

Che, X. and Maag, S. (2013). A formal passive performance testing approach for distributed communication systems. In *ENASE 2013 - Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering, Angers, France, 4-6 July, 2013*, pages 74–84.

Dumitrescu, C., Raicu, I., Ripeanu, M., and Foster, I. (2004). Diperf: An automated distributed performance testing framework. In *5th International Workshop in Grid Computing*, pages 289–296. IEEE Computer Society.

Emden, M. V. and Kowalski, R. (1976). The semantics of predicate logic as a programming language. *Journal of the ACM*, pages 23(4):733–742.

ETSI (2004). Methods for testing and specification (mts); conformance test specification for sip.

Hallé, S. and Villemaire, R. (2012). Runtime enforcement of web service message contracts with data. *IEEE Transactions on Services Computing*, 5(2):192–206.

Hofmann, R., Klar, R., Mohr, B., Quick, A., and Siegle, M. (1994). Distributed performance monitoring: Methods, tools and applications. *IEEE Transactions on Parallel and Distributed Systems*, 5:585–597.

Lalanne, F. and Maag, S. (2013). A formal data-centric approach for passive testing of communication protocols. In *IEEE / ACM Transactions on Networking*, volume 21, pages 788–801.

Larsen, K. G., Mikucionis, M., and Nielsen, B. (2004). Online testing of real-time systems using uppaal. In *Formal Approaches to Software Testing, 4th International Workshop*, pages 79–94.

Lee, D. and Miller, R. (2006). Network protocol system monitoring-a formal approach with passive testing. *IEEE/ACM Transactions on Networking*, pages 14(2):424–437.

Nguyen, H. N., Poizat, P., and Zaïdi, F. (2012). Online verification of value-passing choreographies through property-oriented passive testing. In *14th International IEEE Symposium on High-Assurance Systems Engineering*, pages 106–113.

Raimondi, F., Skene, J., and Emmerich, W. (2008). Efficient online monitoring of web-service slas. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 170–180.

Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., and Peterson, J. (2002). Sip: Session initiation protocol.

Veanes, M., Campbell, C., Schulte, W., and Tillmann, N. (2005). Online testing with model programs. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 273–282.

Wei, W., Suh, K., Wang, B., Gu, Y., Kurose, J. F., Towsley, D. F., and Jaiswal, S. (2009). Passive online detection of 802.11 traffic using sequential hypothesis testing with tcp ack-pairs. *IEEE Transactions on Mobile Computing*, 8(3):398–412.

Yuen, C.-H. and Chan, S.-H. (2012). Scalable real-time monitoring for distributed applications. *IEEE Transactions on Parallel and Distributed Systems*, 23(12):2330 –2337.