Guiding Testers' Hands in Monitoring Tools: Application of Testing Approaches on SIP

Xiaoping Che[1], Stephane Maag[2], Huu Nghia Nguyen[3(✉)], and Fatiha Zaïdi[4]

[1] School of Software Engineering, Beijing Jiaotong University, Beijing, China
xpche@bjtu.edu.cn
[2] Telecom SudParis, CNRS UMR 5157,
9 rue Charles Fourier, 91011 Evry Cedex, France
stephane.maag@telecom-sudparis.eu
[3] Montimage EURL, 39 rue Bobillot, 75013 Paris, France
huunghia.nguyen@me.com
[4] Universite Paris-Sud XI, CNRS UMR 8623, Bat 650, 91405 Orsay Cedex, France
fatiha.zaidi@lri.fr

Abstract. The importance and impact of testing are becoming crucial and strategic for the deployment and use of software and systems. Several techniques have been defined all along the protocol testing process, that allow validating multiple facets of a protocol implementation in particular its *conformance* to the standardized requirements. Among these testing techniques, the ones denoted as *passive* are currently often applied. Indeed, there are non intrusive and based on network observations. In this paper, we intend to help and guide the protocol testers regarding their testing choices by considering the functional protocol properties to check, and the analysis of testing verdicts obtained by applying passive testing tools. We propose a compared analysis of the application of two efficient passive testing methodologies through the study of the Session Initiation Protocol. The results analysis demonstrates that depending on the properties to test, the way to model them, the way of testing (on-line/off-line), the available testing time resources, tradeoffs are needed. Thus, this analysis aims at guiding the testers when tackling the passive testing of communication protocols.

Keywords: Formal methods · Passive testing · Monitoring · SIP

1 Introduction

While todays communications are essential and a huge set of services is available online, computer networks continue to grow and novel communication protocols are continuously being defined and developed. De facto, protocol standards are required to allow different systems to interwork. Though these standards can be formally verified [31], the developers may produce some errors leading to faulty implementations. That is the reason why their implementations must be *tested*.

K. El-Fakih et al. (Eds.): ICTSS 2015, LNCS 9447, pp. 105–123, 2015.
DOI: 10.1007/978-3-319-25945-1_7

Testing is mainly known as the process of checking that a system possesses a set of desired properties and behaviour. Its importance and impact are becoming crucial and strategic for the future deployment and use of software and systems. This can be noticed through the numerous works on testing areas provided by the research communities of course [30] but also by the industry [11] and the standardization institutes [12].

Several techniques have been defined all along the protocol testing process. The main approaches are based on formal models in order, first, to automate the different test phases but also to ease the development and improvement of network protocols. Applying formal techniques allow to validate multiple facets of a protocol implementation such as their reliability, scalability, security, and in particular its *conformance* to the standardized requirements [1]. These techniques are mainly split in two categories: *Active* and *Passive* techniques. While the active ones require a stimulation of the Implementation Under Test (IUT) and an important testing architecture, the passive ones tackled in this work are based on the observation of input and output events of an implementation under test at run-time. Basically, *passive testing* techniques are applied whenever the state of an IUT cannot be controlled by means of test sequences either because access to the interfaces of the system is unavailable or a reset of the IUT is undesired. The term "passive" means that the tests do not disturb the natural run-time of a protocol as the implementation under test is not stimulated. The *trace*, i.e. the record of the event observation, is then compared to the expected behaviour of the IUT allowing to check its conformance.

When testing the implementation of a network protocol, its behaviour is defined either by a formal model or by a set of expected functional properties. In this current work, we consider formal properties to design the expected behaviour of an implementation under test. However, based on the IUT functionalities, the architecture, the system in which it will be integrated, a tester is faced towards the testing methodology to follow, the way to extract relevant protocol properties, how to express them, which tool to apply, etc. Depending on the properties to check, the languages to model them, their expressiveness and the network monitored, the met difficulties could be diverse and the test verdicts different as well. In this paper, we therefore intend to help and guide the protocol testers regarding their testing choices by considering the functional properties to check, and the analysis of testing verdicts obtained by applying testing tools. We propose a compared analysis of the application of two efficient passive testing methodologies by taking into account not only the control parts of the protocol messages but also the data parts. Further, the two chosen techniques proceed differently: on-line versus off-line. The studied comparison is performed through the study of an IP Multimedia Subsystem (IMS) based protocol (the Session Initiation Protocol - SIP). Some traces and formal properties are used as inputs of two open source tools. The results analysis aims at guiding the testers when tackling the passive testing of communication protocols.

Our main contributions are the following:

- The study of two different passive testing approaches on a common network protocol. Based on the same traces sets and functional properties extracted from the SIP standard, the techniques/tools are applied on a real IMS test bed.
- A study of the expressiveness of the languages used to model the functional properties. This allows notably to help the testers when designing certain kinds of properties.
- The analysis and understanding of both sets of obtained test verdicts. Depending on some contexts, it allows to raise false negatives and to reduce inconclusive verdicts.
- To help guiding the protocol testers while choosing some passive testing techniques for a specific system under test.

The remainder of the paper is organized as follows. Both passive testing approaches: Datamon and Prop-tester are described in the Sect. 2. We herein also define the main concepts of protocol messages and traces. In Sect. 3, the experiments are performed on a real IMS platform from which traces are collected and formal SIP properties checked on these execution traces. The results analysis are provided in Section refDiscussion and discussions allowing to guide the testers are given. Section 5 depicts the related works on the passive testing area and we conclude in Sect. 6 with future works mentioned.

2 Basics

In this section, we introduce the general definition of messages and traces in communication protocols. Then, the syntax and semantics of Datamon and Prop-tester are briefly described with the expression equivalence of both tools.

2.1 Message and Trace

A message in a communication protocol is, using the most general possible view, a collection of data fields belonging to multiple domains. Data fields in messages are usually either *atomic* or *compound*, i.e. they are composed of multiple elements (e.g. a URI sip: name@domain.org). Due to this, we also divide the types of possible domains in *atomic*, defined as sets of numeric or string values[1], or *compound*, as follows.

Definition 1. *A compound value v of length $k > 0$, is defined by the set of pairs $\{(l_i, v_i) \mid l_i \in L \wedge v_i \in D_i \cup \{\epsilon\}, i = 1 \ldots k\}$, where $L = \{l_1, \ldots, l_k\}$ is a predefined set of labels and D_i are data domains, not necessarily disjoint.*

In a *compound* value, in each element (l, v), the label l represents the functionality of the piece of data contained in v. The length of each compound value is fixed, but undefined values can be allowed by using ϵ (null value). A

[1] Other values may also be considered atomic, but we focus here, without loss of generality, to numeric and strings only.

compound domain is then the set of all values with the same set of labels and domains defined as $\langle L, D_1, \ldots, D_k \rangle$. Notice that, D_i being domains, they can also be either *atomic* or *compound*, allowing for recursive structures to be defined. Finally, given a network protocol P, a compound domain M_p can generally be defined, where the set of labels and element domains derive from the message format defined in the protocol specification. A *message* of a protocol P is any element $m \in M_p$.

A *trace* is a sequence of messages of the same domain (i.e. using the same protocol) containing the interactions of an entity of a network, called the *point of observation* (P.O), with one or more peers during an indeterminate period of time (the life of the P.O).

Definition 2. *Given the domain of messages M_p for a protocol P. A trace is a sequence $\Gamma = m_1, m_2, \ldots$ of potentially infinite length, where $m_i \in M_p$.*

Definition 3. *Given a trace $\Gamma = m_1, m_2, \ldots$, a trace segment is any finite subsequence of Γ, that is, any sequence of messages $\rho = m_i, m_{i+1}, \ldots, m_{j-1}, m_j (j > i)$, where ρ is completely contained in Γ (same messages in the same order). The order relations $\{<, >\}$ are defined in a trace, where for $m, m' \in \rho, m < m' \Leftrightarrow pos(m) < pos(m')$ and $m > m' \Leftrightarrow pos(m) > pos(m')$ and $pos(m) = i$, the position of m in the trace ($i \in \{1, \ldots, len(\rho)\}$).*

2.2 Datamon

A syntax based on Horn clauses is used to express properties. The syntax is closely related to that of the query language Datalog, described in [2], for deductive databases, however, extended to allow for message variables and temporal relations. Both syntax and semantics are described in the current section.

Syntax. Formulas in this logic can be defined with the introduction of terms and atoms, as defined below.

Definition 4. *A term is either a constant, a variable or a selector variable. In BNF: $t ::= c \mid x \mid x.l.l \ldots l$ where c is a constant in some domain (e.g. a message in a trace), x is a variable, l represents a label, and $x.l.l \ldots l$ is called a selector variable, and represents a reference to an element inside a compound value, as defined in Definition 1.*

Definition 5. *An atom is defined as $A ::= p(\overbrace{t, \ldots, t}^{k}) \mid t = t \mid t \neq t$ where t is a term and $p(t, \ldots, t)$ is a predicate of label p and arity k. The symbols $=$ and $\neq$ represent the binary relations "equals to" and "not equals to", respectively.*

In this logic, relations between terms and atoms are stated by the definition of clauses. A *clause* is an expression of the form $A_0 \leftarrow A_1 \wedge \ldots \wedge A_n$, where A_0, called the head of the clause, has the form $A_0 = p(t_1^*, \ldots, t_k^*)$, where t_i^* is a restriction on terms for the head of the clause ($t^* = \mathrm{c} \mid \mathrm{x}$). $A_1 \wedge \ldots \wedge A_n$ is called the body of the clause, where A_i are atoms.

A *formula* is defined by the following BNF:

$$\phi ::= A_1 \wedge \ldots \wedge A_n \mid \phi \rightarrow \phi \mid \forall_x \phi \mid \forall_{y>x} \phi \\ \mid \forall_{y<x} \phi \mid \exists_x \phi \mid \exists_{y>x} \phi \mid \exists_{y<x} \phi$$

where $A_1, \ldots, A_n$ are atoms, $n \geq 1$ and x, y are variables. Some more details regarding the syntax are provided in the following:

- The $\rightarrow$ operator indicates causality in a formula, and should be read as *"if-then"* relation.
- The $\forall$ and $\exists$ quantifiers, are equivalent to its counterparts in predicate logic. However, as it will be seen on the semantics, here they only apply to messages in the trace. Then, for a trace $\rho, \forall_x$ is equivalent to $\forall(x \in \rho)$ and $\forall_{y<x}$ is equivalent to $\forall(y \in \rho; y < x)$ with the '$<$' indicating the order relation. These type of quantifiers are called *trace temporal quantifiers*.

Semantics. The semantics used in our work is related to the traditional Apt-Van Emdem-Kowalsky semantics for logic programs [10], 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*'). We formalize properties by using the syntax above described and the truth values $\{\top, \bot, ?\}$ are provided to the interpretation of the obtained formulas on real protocol execution traces. Due to the space limitation, we will not go into details of the semantics. However, the interesting reader can refer to [17] in which all the algorithms are defined.

2.3 Prop-tester

Prop-tester was presented in [27] to verify SOAP messages exchanged between Web services. It is an online passive testing tool relying on XML Query processor. In this section we introduce briefly some of its notions and adapt them to be able to verify SIP messages. Let us start with definition of a message.

Definition 6. *Given a finite set of names $\mathcal{O}$, of labels L, and of atomic data values D, a message m takes the form: $o(l_1 = v_1, \ldots, l_n = v_n)$, where $o \in \mathcal{O}$ represents the name of the message. The composite data of the message is represented by a set $\{l_1 = v_1, \ldots, l_n = v_n\}$, rewritten as $(\bar{l} = \bar{v})$ for short, in which each field of this data structure is pointed by a label $l_i \in L$ and its value is $v_i \in D$.*

We define a *candidate event* (CE) e/ϕ as a set of messages e that satisfy some predicate ϕ that represents either functional conditions or non-functional conditions, *e.g.*, conditions of QoS. The predicate can be omitted if it is *true*. As the SIP response messages do not contain operation names but status code numbers, we then extend our definitions with *empty operation name* and *any operation*

name by ϵ and $\sim$ respectively. For example, the `INVITE`$(requestURI = x)/(x =$ "*sip:ua2@CA.cym.com*") represents any `INVITE` message whose *requestURI* is "*http://sip:ua2@CA.cym.com*", while the $\sim (method = x)/(x \neq$ "`ACK`") represents the any message except `ACK`, and the $\epsilon(statusCode = x)/(x \geq 200 \wedge x <$ 300) represents any 2xx response.

Definition 7. *A property is described by the form:*

$$\mathcal{P} ::= \texttt{Context} \xrightarrow{d} \texttt{Consequence} \quad \textit{(positive)}$$
$$\mathcal{P} ::= \texttt{Context} \xrightarrow{d} \neg\texttt{Consequence} \textit{ (negative)}$$

where $d > 0$ is an integer, `Context` *is a sequence of CEs, and* `Consequence` *is a set of CEs.*

This definition allows to express that *if* the `Context` is satisfied *then* the `Consequence` should or should not (depending on the formula type $\mathcal{P}$ or $\neg\mathcal{P}$) be validated after at most d messages. The `Context` is satisfied when all of its CEs are satisfied while the `Consequence` is satisfied when there exists at least one CE which is satisfied and, the `Consequence` is not satisfied when all of its CEs are not satisfied.

Semantics of a Prop-tester property are given by its evaluation on a trace segment. A verdict is emitted if and only if the context of property is satisfied. If there is no non-functional conditions, the verdict is either *Pass* or *Inconclusive* depending on the consequence is satisfied or not respectively. The *Fail* verdict is emitted only if the consequence is not satisfied and there exists a message which violates a non-functional condition of the consequence.

The evaluation of a property on an arbitrary (potential infinite) trace Γ is relied on its evaluation on a segment of Γ as above. In a property, a later CE may depend on a former one, consequently, the verification of a message may require the presence of its precedence. Since we can forward only read data in a continuous stream mode, we need to create buffer which contains some segment of messages stream, what we call a *window*. A created window contains firstly messages validating the context of the property and the d next messages in Γ. Once a window is created, the verification process on the window can start in parallel with the other created windows.

3 Experiments

3.1 Description of the Tools

For the experiments, traces were obtained from SIPp [13]. SIPp is an Open Source implementation of a test system conforming to the IMS, and it is also a test tool and traffic generator for the SIP protocol, provided by the Hewlett-Packard company. It includes a few basic user agent scenarios (UAC and UAS) and establishes and releases multiple calls with the INVITE and BYE methods. It can also read custom XML scenario files describing from very simple

to complex call flows (e.g. subscription including SUBSCRIBE and NOTIFY events). It also supports IPv6, TLS, SIP authentication, conditional scenarios, UDP retransmissions, error robustness, call specific variable, etc. SIPp can be used to test many real SIP equipments like SIP proxies, B2BUAs and SIP media servers. The traces obtained from SIPp contain all communications between the client and the SIP core. Based on these traces and properties extracted from the SIP RFC, tests were performed using our above mentioned methodologies and tools. And all the experiments have been performed on one laptop (2.5 GHz Intel Core i5 with 4 GB RAM).

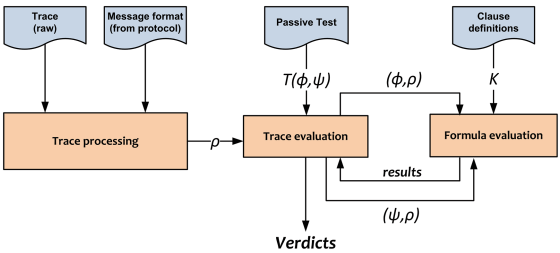

Fig. 1. Testing framework of Datamon

Datamon. The testing framework of Datamon[2] is implemented by using Java. It is composed of three main modules: (1) Filtering and conversion of collected traces; (2) Evaluation of tests; and (3) Evaluation of formulas. Figure 1 shows the way the modules interact and the inputs and outputs from each one. The trace processing module takes the raw traces collected from the network exchange, and it converts the messages from the input format. In our particular implementation, the input trace format is PDML, an XML format that can be obtained from Wireshark traces. The purpose of the module is to convert each packet in the raw trace into a data structure (a compound value) conforming to the definition of a message. This module also performs filtering of the trace in order to only take into account messages of the studied protocol.

The test evaluation module receives input of a passive test, as well as a trace from the trace processing module, and produces a verdict from the satisfaction results of the test and conditional formulas. The formula evaluation module receives a trace and a formula, along with the clause definitions and returns a set of satisfaction results for the query in the trace, as well as the messages and variable bindings obtained in the process.

[2] The implementation and the files used for the experiments can be found at http://www-public.it-sudparis.eu/~maag/Datamon/web/Datamon.html.

Prop-tester. The architecture of Prop-tester[3] is depicted on Fig. 2. The property to be tested is translated into an XQuery such that it returns *false* iff the property is violated, and *true* iff the property is validated. A parser[4] is constructed to parse the log file captured by SIPp tool in *pcap* format. It extracts necessary information, then writes these information into an opened pipeline between the tester and the parser, where it will be verified by an XQuery processor. The properties to be tested in XQuery form will be executed by MXQuery processor on the XML pipeline supplying by the parser. A verdict is emitted as soon as it is found.

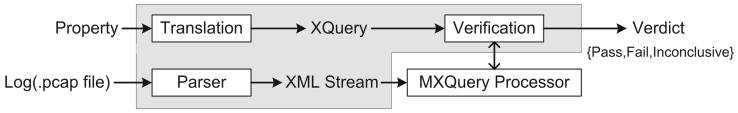

Fig. 2. Testing framework of Prop-tester

3.2 Architecture of SIP

The IMS (IP Multimedia Subsystem) is a standardized framework for delivering IP multimedia services to users in mobility. It was originally intended to deliver Internet services over GPRS connectivity. This vision was extended by 3GPP, 3GPP2 and TISPAN standardization bodies to support more access networks, such as Wireless LAN, CDMA2000 and fixed access network. The IMS aims at facilitating the access to voice or multimedia services in an access independent way to develop the fixed-mobile convergence. Further, the IMS makes now part of the LTE core network for the voice and visio over LTE.

The core of 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 communications with the core network and between the services are done using the Session Initiation Protocol [28]. Figure 3 shows the core functions of the IMS framework and the inherent protocols.

The Session Initiation Protocol (SIP) 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 a body part (optional) to complement or extend the header information. Several RFCs have been defined to extend the protocol. These extensions are used by services of the IMS such as the Presence service [3] and the Push to-talk Over Cellular (PoC) service [4].

[3] The tool is freely available at https://github.com/nhnghia/prop-tester.
[4] https://github.com/nhnghia/pcap2xml.

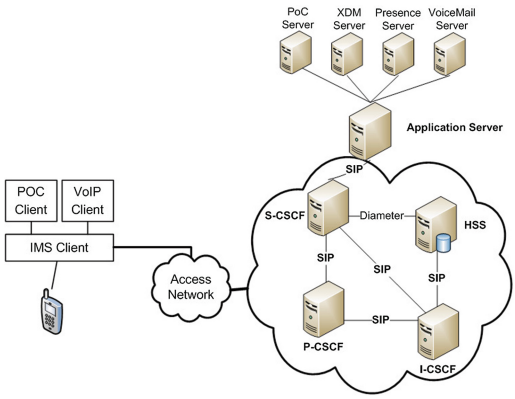

Fig. 3. Core functions of IMS framework

3.3 Properties

In the experiments, a set of properties are tested through Datamon and Prop-tester, in order to analyse their functionality and performance under different conditions.

Property 1. Initially, a simple conformance property *"For every request there must be a response"* is tested.

Table 1. For every request there must be a response

Trace	#Messages	Datamon				Prop-tester			
		#Pass	#Fail	#Inc	Time(s)	#Pass	#Fail	#Inc	Time(s)
1	500	153	0	2	1.652	153	0	2	1.103
2	1000	297	0	0	1.518	297	0	0	1.487
3	2000	575	0	0	3.071	575	0	0	2.246
4	4000	1189	0	1	7.506	1189	0	1	3.190
5	8000	2376	0	1	11.365	2376	0	1	5.480
6	16000	4796	0	1	25.942	4796	0	1	10.106
7	32000	9593	0	0	43.105	9593	0	0	18.728
8	64000	19252	0	1	88.578	19252	0	1	37.128
9	128000	38468	0	1	182.305	38468	0	1	70.390

As Table 1 shows, Datamon and Prop-tester obtain the same number of 'Pass' and non-pass verdicts. Since a finite segment of an infinite execution is being tested in our experiments, it is not possible to declare a '*Fail*' verdict in Datamon and Prop-tester, for the indeterminacy that testers do not know if it may

become a '*Pass*' in the future. As a result, they treat the non-pass verdicts as '*Inconclusive*' verdicts. In this simple property, there is no essential difference between the results returned by Datamon and Prop-tester.

Property 2. Therefore, a more complex conformance property *"For successfully established sessions, every **INVITE** request should be responded with a 200 response"* is tested for delving deeper into the differentiation between the tools.

The results shown in Table 2 illustrate that a difference between mechanisms can result on evaluation times. Although both tools still obtain the same number of '*Pass*' and non-pass verdicts, it can be observed that Prop-tester takes much less evaluation time than Datamon, especially when handling numerous messages. As introduced in previous sections, Prop-tester introduces a predefined distance value d into its evaluation process for instantly concluding verdicts. With the help of this value, Prop-tester will omit comparisons with messages beyond this distance.

Table 2. For successfully established sessions, every INVITE request should be responded with a 200 response

Trace	#Messages	Datamon				Prop-tester			
		#Pass	#Fail	#Inc	Time(s)	#Pass	#Fail	#Inc	Time(s)
1	500	57	0	11	1.700	57	0	11	1.058
2	1000	119	0	22	4.038	119	0	22	1.385
3	2000	248	0	53	13.505	248	0	53	2.114
4	4000	459	0	123	46.358	459	0	123	2.782
5	8000	926	0	233	180.388	926	0	233	5.019
6	16000	1842	0	440	658.148	1842	0	440	8.476
7	32000	3667	0	905	2559.239	3667	0	905	14.542
8	64000	7230	0	1911	7510.563	7230	0	1911	28.735
9	128000	14511	0	3767	28187.956	14511	0	3767	56.579

Conversely, Datamon has to compare all the following messages till the end of a trace, in order to confirm the non-existence of a target message. However, the mechanism used in Prop-tester raises a question: How will Prop-tester react if target messages appear after the predefined distance d?

Property 3. Before answering to the question, a related property relevant to time *"For each **INVITE** request, the response should be received within 16s"* is tested for verifying the extensibility of both monitoring tools.

Time relevant properties can be seen as performance requirements which are different from the conformance requirements tested above, having the ability to test performance requirements is a crucial step for monitoring tools to extend its

Table 3. For each **INVITE** request, the response should be received within 16s

Trace	#Messages	Datamon				Prop-tester			
		#Pass	#Fail	#Inc	Time(s)	#Pass	#Fail	#Inc	Time(s)
1	500	57	11	0	1.098	57	11	0	1.043
2	1000	119	22	0	3.192	119	22	0	1.383
3	2000	248	53	0	9.841	248	53	0	1.870
4	4000	459	123	0	35.214	459	123	0	2.765
5	8000	926	233	0	131.578	926	233	0	4.533
6	16000	1842	140	0	486.181	1842	440	0	8.069
7	32000	3667	905	0	1728.003	3667	905	0	14.512
8	64000	7230	1911	0	7286.181	7230	1911	0	28.321
9	128000	14511	3767	0	30804.213	14511	3767	0	56.817

functionality. Not surprisingly, as Table 3 shows, both tools can test this performance property and they obtain the same results. Nevertheless, non-pass verdicts are concluded as '*Fail*' verdicts which is different from testing the previous conformance requirements. Because when testing such performance requirements with timing constraint, there is no indeterminacy in the trace. Definite verdicts ('*Pass*' or '*Fail*') should be emitted, rather than indefinite ones ('*Inconclusive*'). That is notably the reason why the reader will notice that the results are here similar to the ones obtained with Property 2 in the way that all '*Inconclusive*' verdicts of Property 2 are now '*Fail*'. Besides, Prop-tester still takes the lead in evaluation time.

Property 4. Back to figuring out the answer raised in Property 2, a more complicated property *"Every 2xx response for* ***INVITE*** *request must be responded with an* ***ACK****"* is tested.

Different from previous properties, obvious discrepancies between the verdicts returned from Datamon and Prop-tester can be observed from Table 4. Take a closer look at trace 6, all the '*Inconclusive*' verdicts reported from Prop-tester are caused by missing '**ACK**' responses. In fact, these '**ACK**' responses do exist in the trace, but appear after the predefined d in Prop-tester. Consequently Prop-tester treats these 'missing' '**ACK**' responses as '*Inconclusive*' verdicts could be considered as false negatives. The false negatives also occur in trace 7, 8 and 9 due to the same reason.

These phenomena answer to the question raised in Property 2: the mechanism used in Prop-tester would lead to inconclusive verdicts if the predefined distance d is set improperly. In contrast, owing to its rigorous mechanism for obtaining verdicts, Datamon does not have such problems but its evaluation times are still far behind Prop-tester.

Table 4. Every 2xx response for **INVITE** request must be responded with an **ACK**

Trace	#Messages	Datamon				Prop-tester			
		#Pass	#Fail	#Inc	Time(s)	#Pass	#Fail	#Inc	Time(s)
1	500	57	0	0	1.241	57	0	0	24.805
2	1000	119	0	0	3.884	119	0	0	50.570
3	2000	248	0	0	12.102	248	0	0	103.088
4	4000	459	0	0	45.365	459	0	0	199.890
5	8000	926	0	0	181.758	926	0	0	400.920
6	16000	1842	0	0	658.033	1831	0	11	796.477
7	32000	3666	0	1	2631.765	3588	0	79	1617.233
8	64000	7217	0	13	7501.719	6931	0	299	3204.401
9	128000	14493	0	18	28616.957	13868	0	643	6216.099

Property 5. Furthermore, a sophisticated conformance property *"No session can be initiated without a previous registration"* is tested for exploring the functionality of both tools in depth.

Table 5. No session can be initiated without a previous registration

Trace	#Messages	Datamon			
		#Pass	#Fail	#Inc	Time(s)
1	500	56	0	1	10.318
2	1000	114	0	5	41.272
3	2000	243	0	5	165.090
4	4000	457	0	2	660.361
5	8000	912	0	14	2531.445
6	16000	1840	0	2	10565.782
7	32000	3659	0	8	40439.623
8	64000	7225	0	5	160578.492
9	128000	14506	0	5	593073.968

Besides different mechanisms, the diverse logic used for formalizing properties in both tools affect testing results likewise. As shown in Table 5, Datamon appears its potentiality on formalizing and testing sophisticated properties which Prop-tester can not handle. Although the evaluation times seem a bit high, it has to be noticed that the low performance of evaluation is due to memory limitation of the computer we used. If a more powerful server is applied, the evaluation times will be apparently reduced to satisfying numbers.

4 Discussions and Testers' Guidance

In this section, we will first point out the drawbacks and advantages of each approach according to different evaluation criteria. Then, we will give some advices to the tester to guide him depending on his testing objectives.

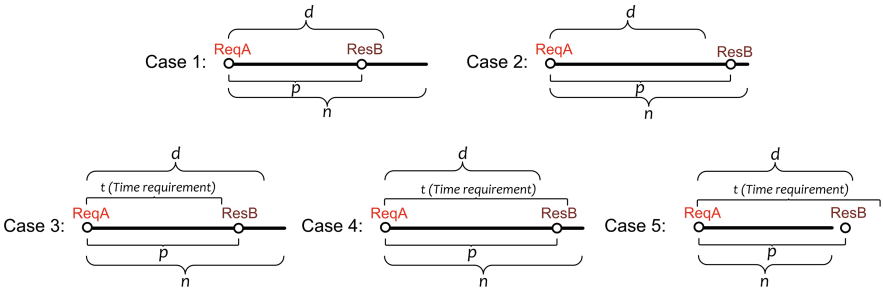

Fig. 4. Different cases, ReqA, ResB represent for a request and its response respectively.

- The two approaches are property-based passive testing techniques. The properties are checked on the real execution traces. The Datamon tool is based on definition of Horn clauses which are closely related to the query Datalog language. Such formulas are made of atoms and terms. Formulas with quantifiers and data can be defined. Prop-tester is based on *if then* clause where the quantifiers are implicit and data can also be expressed. The main difference relies on the distance used by Prop-tester. Prop-tester is clearly an on-line testing tool and it is why such a distance is needed to buffer the traces. Regarding expressiveness issue, such a distance can be a drawback. Indeed, if the distance is not explicitly stated in the requirements, the distance is an artefact used by the testing method. In this case, if the trace does not satisfy the property because of the distance, an inconclusive verdict is emitted. On the contrary, if the distance is a constraint of the requirement, a fail verdict should be emitted. Concerning the property 4 that needs to verify a triple of SIP messages with a partial order between them ($m1 \leq m2 \leq m3$), Prop-tester is not able to express it. For that purpose, a combinatorial numbers of properties has to be written, in this case 50 properties. Moreover, Prop-tester, as an online tool, is not able to express the property 5 which is a complex property that relates to a behaviour occurred in the past of the trace.
  Except this difference, we demonstrate that the properties expressed by both tools are LTL+FO equivalent because the part of the formula related to the distance is always true. The properties equivalence is not shown in this paper for lack of room. Interested readers can refer to the technical report [8].
- One interesting feature of the Prop-tester tool is that negative property can be written. We can specify what should never occur in the system. For that purpose, prop-tester negates positive property.

– Both approaches have different complexity. In Datamon, the algorithm uses a recursive procedure to evaluate formulas, coupled with a modification of SLD (Selective Linear Definite-clause) resolution algorithm [5] for evaluation of Horn clauses. In the work, it is shown that the worst-case time complexity for a formula with k quantifiers is $\mathcal{O}(n^k)$ to analyse the full trace, where n is the number of messages in the trace. Although the complexity seems high, this corresponds to the time to analyse the complete trace, and not for obtaining individual solutions, which depends on the type of quantifiers used. For instance for a property $\forall_x p(x)$, individual results are obtained in $\mathcal{O}(1)$, and for a property $\forall_x \exists_y q(x, y)$, results are obtained in the worst case in $\mathcal{O}(n)$. Finally, it can also be shown that a formula with a '$\rightarrow$' operator, where Q are quantifiers

$$\underbrace{Q \ldots Q}_{k}(\underbrace{Q \ldots Q}_{l}(A_1 \wedge \ldots \wedge A_p) \rightarrow \underbrace{Q \ldots Q}_{m}(A'_1 \wedge \ldots \wedge A'_q))$$

has a worst-case time complexity of $\mathcal{O}(n^{k+max(l,m)})$, which has advantages with respect to using formulas without the '$\rightarrow$' operator. For instance, evaluation of the formula $\forall_x(\exists_y p(x, y) \rightarrow \exists_z q(z))$ has a complexity of $\mathcal{O}(n^2)$, while the formula $\forall_x \exists_y \exists_z (p(x, y) \wedge q(z))$ has a complexity of $\mathcal{O}(n^3)$ in the worst case [17].

For Prop-tester, the complexity to verify of a property $\langle e_1, \ldots, e_k \rangle \xrightarrow{d} \{e'_1, \ldots, e'_m\}$ on a trace containing n messages is as follows. Prop-tester forwards only read data in a continuous stream mode. The verification is done on a buffer which contains some fragments of message streams, what we call a *window*. The window size is $k + d$. There are $n - k$ windows. The complexity is $\mathcal{O}((n - k) * (k + d))$. Since k and d are constant and usually highly smaller than n, the complexity would be $\mathcal{O}(n)$. In the worst case where one wants buffer the entire trace, *i.e.*, $d \geq n - k$, the complexity is $\mathcal{O}(n^2)$.
The better complexity of Prop-tester is demonstrated in the experiments that have been conducted. Prop-tester is very performant in time to evaluate the properties.

– The Datamon tool has been designed to perform off-line analysis. Indeed, execution traces are recorded and afterwards analysed while Prop-tester is mainly efficient to perform on-line analysis during the real execution of the system. To perform on-line testing, the tool needs to have good performance and as consequence to give rapid answer for the verification process. The efficiency of Prop-tester is dependent on the efficiency of the XQuery engine that it relies on.
– Concerning the conformance verdicts emitted by both tools, there exist some differences in their accuracy. To exhibit this point, we illustrate it with Fig. 4. For the case 1, the distance d of the Prop-tester tool has no impact on the verdict as d is greater than the distance p between the request and its response. As for case 2, it proves the deduction we had in the experiments. When the distance d is shorter than p, Prop-tester emits '*Inconclusive*' verdicts.
For the cases 3 to 5, when timing constraints are expressed by the properties, both tools can emit different verdicts depending on the time requirement t and

the size of the trace n, as illustrated in Table 6. In case 3, it is almost the same case as case 1. The distance d does not influence on the verdict if d is greater than the distance p and the time requirement t, both tools return a '*Fail*' verdict when the timing constraint is violated. However, in case 4, when the distance d is shorter than t and p, Prop-tester will emit '*Inconclusive*' verdicts while Datamon still can detect the response and emit definite verdicts. For case 5, let us assume that the response *resB* is present in the trace but will appear after the captured trace. For Datamon, if *resB* appears after n, it will issue a '*Fail*' verdict even if the timing constraint is not violated. Contrarily, with Prop-tester a '*Fail*' verdict can be emitted if the time is elapsed during the d distance otherwise it will emit an '*Inconclusive*' verdict.

Table 6. Verdicts of tools under different cases, case 1 and 2 are tested through property 1, case 3 to 5 are tested through property 3.

Case	Datamon				Prop-Tester			
	#Pass	#Fail	#Inc	Time(s)	#Pass	#Fail	#Inc	Time(s)
1	1	0	0	1.382	1	0	0	2.509
2	1	0	0	1.750	0	0	1	2.562
3	0	1	0	1.022	0	1	0	2.665
4	1	0	0	0.939	0	0	1	2.485
5	0	1	0	0.939	0	0	1	2.485

We have mentioned the advantages and drawbacks of each approach and their related tools. What is important to point out is for what purpose each tool has been designed. Datamon is clearly well suited for off-line analysis of a system while Prop-tester is very efficient for on-line analysis. Regarding this main feature, the drawbacks and advantages are closely related. As already pointed out above, the expressiveness is better for Datamon. Indeed, the off-line analysis allows to express complex properties and even properties that express constraints on the past of the trace. Obviously, for an on-line analysis which analyses the stream in a forward manner and with the form of *if then clause* of Prop-tester such properties cannot be expressed. Moreover, always due to the form of its properties, properties expressing relations with several variables (more than two) cannot be expressed by Prop-tester. Furthermore, Prop-tester needs for its on-line analysis to determine a d distance. Such a distance can be seen as a constraint of the requirements and in this case, the verdicts will be impacted. Otherwise, it must not have an impact on the verdict as it represents an implementation constraint needed by the approach to limit the stream to be analysed. A very important strength of Prop-tester relies on its performance which is of very important interest to test complex system in a continuous way.

Both tools are complementary. Indeed, for a rapid analysis of the running system, the main behaviours of a system can be tested as the expressiveness is not

always an issue for some tested systems. It can help to fix rapidly an erroneous system by providing rapid feedback of discrepancy between the system and what it is expected to do. Meanwhile, Datamon can be used as a background tool to carefully analyse recorded system traces and by having more complex properties that can be checked.

To conclude, Prop-tester can be used as an off-line tool and in this case, the d distance is no longer used in the expression of the property and as a consequence some limitations can be overcome. The form of the properties can also be modified in order to increase the expressiveness. Concerning Datamon, this tool is clearly not designed to be an on-line tool.

5 Related Work

Formal testing methods have been used for years to prove correctness of implementations by combining test cases evaluation with proofs of critical properties. In [14,17] the authors present a description of the state of the art and theory behind these techniques. Within this domain, and in particular for network protocols, passive testing techniques have to be used to test already deployed platforms or when direct access to the interfaces is not available. Some examples of these techniques using Finite State Machine derivations have been used in the past which are described in [21,25]. Most of these techniques consider only control portions, in [15,20,29], data portion testing is approached by evaluation of traces by use of EEFSM (Event-based Extended Finite State Machine), SEFSM (Simplified Extended Finite State Machine) and IOTS (Input-Output Transition Systems) models. They focus on testing correctness in the specification states and internal variable values. Our approach, although inspired by it, is different in the sense that we test critical properties directly on the trace without any generation or specification of state models of the tested protocol or functional properties. A study of the application of invariants to an IMS service was also presented by us in [17,18].

In [26], the authors defined a methodology for the definition and testing of time extended invariants, where data is also a fundamental principle in the definition of formulas and a *packet* (similar to a *message* in our work) is the base container data. In this approach, the satisfaction of the packets to certain *events* is evaluated, and properties are expressed as $e_1 \xrightarrow{When,n,t} e_2$, where e_1 and e_2 are events defined as a set of constraints on the data fields of packets, n is the number of packets where the event e_2 should be expected to occur after finding e_1 in the trace, and t is the amount of time where event e_2 should be found on the trace after (or before) event e_1. This work served as an inspiration for both approaches described in the current document, however we improved it by allowing the definition of formulas that test data relations and causality between multiple messages/packets.

Although closer to runtime monitoring, the authors of [7] propose a framework for defining and testing security properties on Web Services using the Nomad [9] language, based on previous works by the authors of [22]. As a work

on web services, data passed to the operations of the service is taken into account for the definition of properties, and multiple events in the trace can be compared, allowing to define, for instance, properties such as "Operation *op* can only be called between operations *login* and *logout*". Nevertheless, in web services, operations are atomic, that is, the invocation of each operation can be clearly followed in the trace, which is not the case with network protocols where operations depend on many messages and sometimes on the data associated with the messages.

Further, other recent works like [23] present distributed passive testing frameworks aiming at simplifying and automating service testing. And, techniques based on "geometric approaches" [19] have been used in continuous distributed monitoring for analyzing the behaviors of communication protocols.

Besides, some researchers presented a tool for exploring online communication and analyzing clarification of requirements over the time in [16]. It supports managers and developers to identify risky requirements. Another interesting tool is PTTAC [6] which automatizes a formal framework to perform passive testing for systems where there is an asynchronous communications channel between the tester and the system. We should also cite the recent extension of PASTE [24] that performs passive testing of communication systems with temporal constraints associated to performance and delays. Though these tools are interesting, they need specific state models or do not allow to analyze data payloads.

6 Conclusion and Perspectives

In this paper, we described two passive testing approaches to test efficiently, in a non intrusive way, the main properties of a communicating protocol, the Session Initiation Protocol. The approaches and their associated freely available tools, Datamon and Prop-tester, allow to test real execution traces provided by SIPp. Both approaches are based on formal definition of desired properties to be tested. The performances and accuracy of verdicts for both tools are dependent on the expressiveness of properties and also on the techniques used, i.e. off- or on-line techniques. The approaches can be used by a tester in a complementary way. In one hand, Prop-tester can be used to have rapid testing answer on some properties to be tested and it can be launched in a continuous way to analyse the execution traces. On the other hand, Datamon, as a back-end tool, can be used to test more intensively the protocol with the definition of complex properties on the recorded traces.

As an immediate perspective line, we expect to integrate more smoothly both techniques in order to provide to testers more accurate verdicts, by reducing the number of inconclusive verdicts. Moreover, both tools can take advantage of each other and then improve for one its expressiveness and for the other its performances. Such improvements can be reached by learning from each technique. Prop-tester has been used for its first time in the testing of such communicating protocol. We expect to promote the use of such tools to other real-life protocols.

References

1. ISO/IEC 9646-1: Information technology - open systems interconnection - conformance testing methodology and framework - part 1: General concepts. Technical report, ISO, January 1994
2. Abiteboul, S., Hull, R., Vianu, V.: Datalog and Recursion, 2nd edn. Addison-Wesley, Reading (1995)
3. Open Mobile Alliance: Internet messaging and presence service features and functions. Technical report, OMA (2005)
4. Open Mobile Alliance: Push to talk over cellular requirements. Technical report, OMA (2006)
5. Apt, K., Van Emden, M.: Contributions to the theory of logic programming. J. ACM (JACM) **29**(3), 841–862 (1982)
6. Camacho-Magrinan, M.A., Merayo, M.G., Medina-Bulo, I.: PTTAC: passive testing tool for asynchronous systems. In: Proceedings of SITIS, pp. 223–229 (2014)
7. Cao, T.D., Phan-Quang, T.T., Félix, P., Castanet, R.: Automated runtime verification for web services. In: Proceedings of ICWS, pp. 76–82 (2010)
8. Che, X., Maag, S., Nguyen, H.N., Zaïdi, F.: Guiding testers' hands in monitoring tools/appendix: expression equivalence of the two approaches. Technical report RR15001-RS2M, Institut Mines-Telecom/Telecom SudParis, August 2015
9. Cuppens, F., Cuppens-Boulahia, N., Sans, T.: Nomad: a security model with non atomic actions and deadlines. In: Proceedings of CSFW, pp. 186–196 (2005)
10. Emden, M.V., Kowalski, R.: The semantics of predicate logic as a programming language. J. ACM **23**(4), 733–742 (1976)
11. ETSI/ES 201 873–1: Methods for testing and specification (MTS); the testing and test control notation version 3; part 1: TTCN-3 core language, v3.2.1. Technical report, ETSI (2007)
12. European Telecommunications Standards Institute/ETSI TS 134 123–3: Universal mobile telecommunications system (UMTS); user equipment (UE) conformance specification; part 3: abstract test suite (ATS). Technical report, ETSI, June 2013
13. Hewlett-Packard: SIPp (2004). http://sipp.sourceforge.net/
14. Hierons, R.M., Krause, P., Luttgen, G., Simons, A.J.H.: Using formal specifications to support testing. ACM Comput. Surv. **41**(2), 176 (2009)
15. Hierons, R.M., Merayo, M.G., Núñez, M.: Passive testing with asynchronous communications. In: Beyer, D., Boreale, M. (eds.) FORTE 2013 and FMOODS 2013. LNCS, vol. 7892, pp. 99–113. Springer, Heidelberg (2013)
16. Knauss, E., Damian, D.: V:Issue:lizer: exploring requirements clarification in online communication over time. In: Proceedings of ICSE, pp. 1327–1330 (2013)
17. Lalanne, F., Maag, S.: A formal data-centric approach for passive testing of communication protocols. IEEE/ACM Trans. Netw. **21**(3), 788–801 (2013)
18. Lalanne, F., Maag, S., de Oca, E.M., Cavalli, A.R., Mallouli, W., Gonguet, A.: An automated passive testing approach for the IMS PoC service. In: Proceedings of ASE, pp. 535–539 (2009)
19. Lazerson, A., et al.: Monitoring distributed streams using convex decompositions. VLDB Endow. **8**(5), 545–556 (2015)
20. Lee, D., Miller, R.: Network protocol system monitoring - a formal approach with passive testing. IEEE/ACM Trans. Netw. **14**(2), 424–437 (2006)
21. Lee, D., Netravali, A.N., Sabnani, K.K., Sugla, B., John, A.: Passive testing and applications to network management. In: Proceedings of ICNP, pp. 113–119 (1997)

22. Li, Z., Jin, Y., Han, J.: A runtime monitoring and validation framework for web service interactions. In: Proceedings of ASWEC, pp. 70–79 (2006)
23. Lopez, J., Maag, S., Morales, G.: Behavior evaluation for trust management based on formal distributed network monitoring. World Wide Web 1–19 (2015)
24. Merayo, M.G., Núñez, A.: Passive testing of communicating systems with timeouts. Inf. Softw. Technol. **64**, 19–35 (2015)
25. Miller, R.: Passive testing of networks using a CFSM specification. In: Proceedings of IPCCC, pp. 111–116 (1998)
26. Morales, G., Maag, S., Cavalli, A.R., Mallouli, W., de Oca, E.M., Wehbi, B.: Timed extended invariants for the passive testing of web services. In: Proceedings of ICWS, pp. 592–599 (2010)
27. Nguyen, H.N., Poizat, P., Zaïdi, F.: Online verification of value-passing choreographies through property-oriented passive testing. In: Proceedings of HASE, pp. 106–113 (2012)
28. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: session initiation protocol (2002)
29. Ural, H., Xu, Z.: An EFSM-based passive fault detection approach. In: Petrenko, A., Veanes, M., Tretmans, J., Grieskamp, W. (eds.) TestCom/FATES 2007. LNCS, vol. 4581, pp. 335–350. Springer, Heidelberg (2007)
30. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Softw. Test. Verification Reliab. **22**, 297–312 (2012)
31. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.: Formal methods: practice and experience. ACM Comput. Surv. **41**, 19:1–19:36 (2009)