A Formal Online Monitoring approach to Test Network Protocols

Xiaoping Che, Stephane Maag and Jorge Lopez

Institut Mines-Telecom/Telecom SudParis, CNRS UMR 5157 SAMOVAR.
Evry, France
Email: {xiaoping.che,stephane.maag,jorge.eleazar.lopez_coronado}@telecom-sudparis.eu

***Abstract*—While the current network protocols and systems become more and more complex, the testing process of their functional and non-functional behaviors comes to be crucial. Among the testing techniques, we herein focus on their test at runtime in an online way which requires the ability to handle numerous messages in a short time with 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. Furthermore, we experimented our approach with several Session Initiation Protocol (SIP) properties in a real IP Multimedia Subsystem environment and obtained relevant verdicts.**

Keywords–Online Testing; Network Protocols; Monitoring

I. INTRODUCTION

In order to evaluate the quality and conformance of a system or an implementation under test (IUT) in relation with a standard, the testing process is a crucial activity. Among the well known and commonly applied approaches, the *passive* testing techniques (also called *monitoring*) are today gaining efficiency and reliability [1]. 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 continuously tests during the operation phase of the IUT.

When we apply 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 [2] [3] [4]. 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 [5], 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 this work, we propose a novel formal online passive testing approach that is applied at runtime to test the functional and non-functional requirements of an implementation under test.

Based on a previous proposed methodology [6] [7], we propose its extension to present a logic-based passive testing approach for checking the requirements of communicating protocols. In [6] and [7], we presented our formalism that 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 '*Pass*', '*Fail*', '*Time-Fail*', '*Data-Inc*' and '*Inconclusive*'. Finally, in order to demonstrate the efficiency of our online approach, we apply it on a real IP Multimedia Subsystem (IMS) communicating environment.

Our paper's primary contributions are:

- A formal passive online testing approach to avoid stopping the execution of the testing process when monitoring a tested protocol. New verdicts are provided in order to consider that the monitored traces are not cut.
- A testing process that is executed in a transparent way without overloading, overcharging the CPU and memory of the used equipment on which the tester will be run. Mechanisms of notifications is defined.

The reminder of the paper is organized as follows. In Section II, a short review of the related works and problems are provided. In Section III, we describe the architecture and testing process in detail. Our approach has been implemented and relevant experiments are depicted in Section IV. Finally, we conclude and provide perspectives in Section V.

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 [8], 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 [9]. It supports managers and developers to identify risky requirements. In [10], 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. We should also cite the work [11] from which an industrial testing tool has been developed. This work is 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

 ISBN: 978-1-61208-354-4

designed, complex ones with data causality can not. Moreover, although their approach is efficient with an important data flow, the process is still offline with finite traces that are considered as very long. 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 [12].

We may also cite some online active testing approaches from which we got inspired. In [4], the authors presented a framework that automatically generates and executes tests 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 assigns verdicts to every generated state. In [13], they presented an event-based approach for modeling and testing the functional behavior of Web Services (WS). Functions of WS are modeled by event sequence graphs (ESG) and they raised the holistic testing concept that integrates positive and negative testing. [14] proposed a data-centric approach to test a protocol by taking account the control parts of the messages as well as the data values carried by the message parameters contained in an extracted execution trace. Interesting and promising results were obtained while testing the SIP protocol.

Inspired from all these above cited works, we propose an online formal passive testing approach by defining functional properties of IUT, without modeling the complete system (contratry to Model Based Testing - MBT) and by considering eventual false negatives. For this latter, we introduce a new verdict '*Time-Fail*' for distinguishing the real functional 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 to keep the integrity of incoming traces. Furthermore, for the lacking attention to test data portions of messages in current researches, our approach provides the ability to test both the data portion and control portion, accompanying with another new verdict '*Data-Inc*' triggered when no data portions are available during the testing process.

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 [15] 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.

B. Testing Process

As shown in Figure 1, the testing process consists of eight parts: Formalization, Construction, Capturing, Generating Filters/Setup, Filtering, Transfer/Buffering, Load Notification and Evaluation.

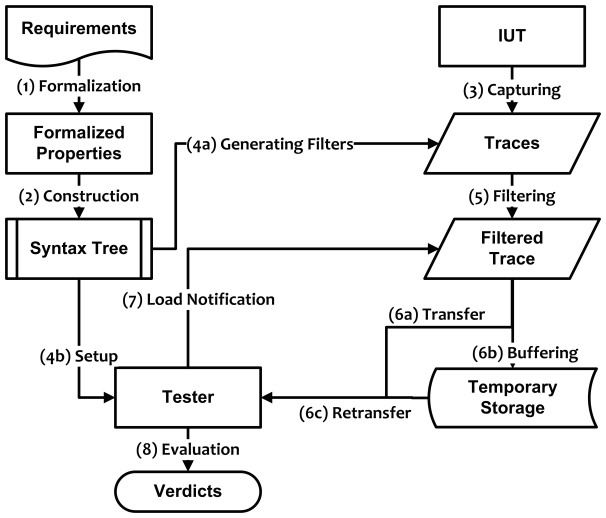

Figure 1. Architecture of our online testing approach

a) Formalization: Initially, informal protocol requirements are formalized using Horn-logic based syntax. A *message* 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. Data domains are defined as *atomic* or *compound*. Once given a network protocol P, a *compound* domain M_p can be defined by the set of labels and data domains derived from the message format defined in the protocol specification/requirements.

A *term* is defined in Backus-Naur Form (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*. An *atom* is defined as the relations between *terms*, $A ::= p(term, ..., term) \mid term = term \mid term \neq term \mid term < term$. The relations between *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 $A_1, ..., A_n$ are *atoms*. Finally, a *formula* is defined by the BNF: $\phi ::= A_1 \wedge ... \wedge A_n \mid \phi \rightarrow \phi \mid \forall_x \phi \mid \forall_{y>x} \phi \mid \forall_{y<x} \phi \mid \exists_x \phi \mid \exists_{y>x} \phi \mid \exists_{y<x} \phi$, where $\exists$ and $\forall$ represent for "it exists" and "for all" respectively. The semantics used in our work is related to the traditional Apt-Van Emdem-Kowalsky semantics for logic programs [16], from which an extended version has been provided in order to deal with messages and trace temporal quantifiers. Due to the space limitation, we will not go into details of the semantics. The interested readers may have a look at the works [6] and [1].

Then the verdicts {'*Pass*', '*Fail*', '*Time-Fail*', '*Inconclusive*', '*Data-Inc*'} are provided to the interpretation of obtained formulas on real protocol execution traces. However, different from offline testing, definite verdicts should be immediately returned in online testing process. This indicates that only '*Pass*', '*Fail*' and '*Time-Fail*' should be emitted in the final report, and indefinite verdicts '*Data-Inc*' and '*Inconclusive*' will be used as temporary unknown status, but finally must be transformed to one of the definite verdicts at the end of the testing process.

b) Construction: From formalized formulas, a syntax tree is constructed for further testing processes. In this process, each formula representing a requirement will be transformed

 ISBN: 978-1-61208-354-4

to an Abstract Syntax Tree (AST) using the TREEGEN algorithm [17]. 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 [18]. The syntax tree will be transferred to the tester as requirements and will be used to filter the captured traces.

c) 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.

d) 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:
- PASS: The message or trace satisfies the requirements.
- FAIL: The message or trace does not satisfy the requirements.
- 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*, not as a *Time-Fail* verdict.
- 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).
- 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, in the future, to apply a support vector machine (SVM) approach [19] in order to train our testing processes and predicate the *Data-Inc* verdicts.

e) 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.

f) 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 this computer of the end-user), a "load notification" (7) is provided to the monitor in order to transfer/store incoming traces. Based on the message format of the protocols to be tested, different buffering methods will be applied. itemsep=2.5pt, topsep=2pt, partopsep=0pt

- 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).

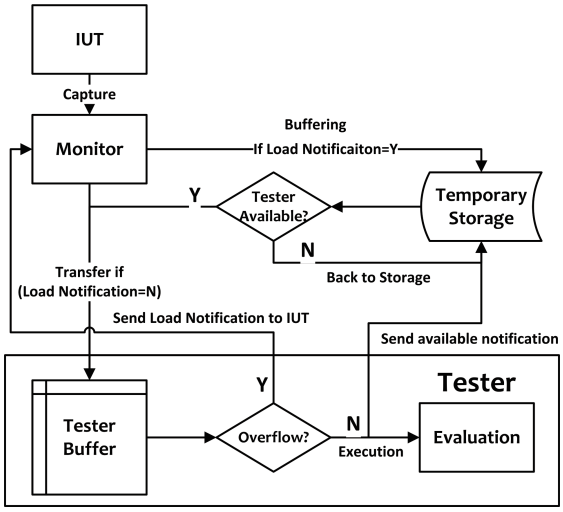

Figure 2. Process of buffering and notification

g) 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 N" to release stored traces and to pursue incoming packets is sent. A brief description of processes 6 and 7 is shown in Figure 2.

As the figure illustrates, 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

 ISBN: 978-1-61208-354-4

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.

h) 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 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) [20].

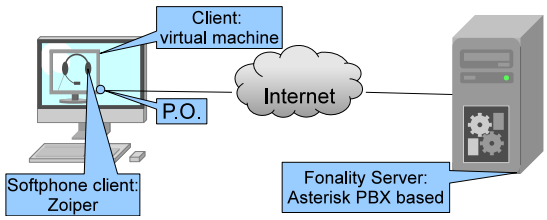

Figure 3. Experiments environment

For our experiments, communication traces were obtained through ZOIPER [21] which is a VoIP soft client, meant to work with any IP-based communication systems and infrastructure. We run four ZOIPER VoIP clients on the virtual machines using VirtualBox for Mac version 4.2.16. On the other side, the server is provided by Fonality [22], which is running Asterisk PBX 1.6.0.28-samy-r115. As Figure 3 shows, the tests are performed in the virtual machines by opening a live capture on the client local interface. This live capture is processed by the clients using an implementation of the formal approach above mentioned and was developed in C code.

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 4, 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: $\forall_x(request(x) \wedge x.method = \textbf{INVITE} \rightarrow \exists_{y>x}(responds(y,x) \wedge success(y)) \rightarrow \exists_{z>y}(ackResponse(z,x,y) \wedge withintime(z,y,2s)))$.

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 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 presents a new logic-based online passive testing approach to test conformance and performance of network protocol implementation. Our approach 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 that are evaluated on real-time protocol traces. The evaluation of the property returns a *Pass*, *Fail*, *Time-Fail*, *Inconclusive* or *Data-Inc* result, derived from the given trace. The approach also includes an online testing framework. To verify and test the approach, we designed 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 we successfully obtained relevant results from testing several properties online.

Furthermore, as future works, we aim at applying our approach under billions of messages and extending more

 ISBN: 978-1-61208-354-4

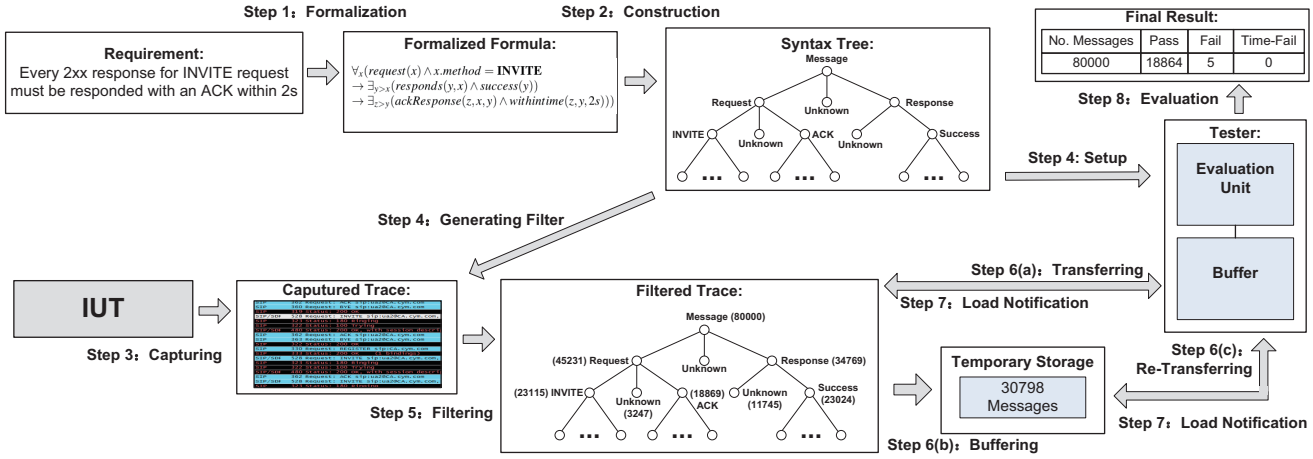

Figure 4. Use case for Testing Process

Properties	Total Messages	Filtered out Messages	Filtered out Rate	Pass	Fail	Time-Fail	Incon	Data-Inc
Prop.1	2324506	1631797	70.19%	631271	0	61432	52	2164
Prop.2	2324506	1631797	70.19%	498124	194579	0	52	2164
Prop.3	2324506	1979904	85.17%	314923	0	29673	14	1086
Prop.4	2324506	1979904	85.17%	247257	97339	0	14	1086
Prop.5	2324506	2259032	97.18%	61550	0	3924	6	371

TABLE I. Online Testing result for Properties

testers in a distributed environment. Thus, the efficiency and processing capacity of the approach will be scalably tested. Meanwhile, we will work on the optimization of our algorithms to severe situations in case of several distributed P.Os, and try to use SVM for predicting *Data-Inc* verdicts and thus to avoid non relevant situations.

REFERENCES

[1] F. Lalanne and S. Maag, "A formal data-centric approach for passive testing of communication protocols," in IEEE / ACM Transactions on Networking, vol. 21, no. 3, 2013, pp. 788–801.

[2] M. Veanes, C. Campbell, W. Schulte, and N. Tillmann, "Online testing with model programs," in Proceedings of the 10th European Software Engineering Conference, 2005, pp. 273–282.

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

[4] T.-D. Cao, P. Félix, R. Castanet, and I. Berrada, "Online testing framework for web services," in Third International Conference on Software Testing, Verification and Validation, 2010, pp. 363–372.

[5] A. C. Viana, S. Maag, and F. Zaïdi, "One step forward: Linking wireless self-organizing network validation techniques with formal testing approaches," ACM Comput. Surv., vol. 43, no. 2, 2011, p. 7.

[6] X. Che, F. Lalanne, and S. Maag, "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, 2012, pp. 53–64.

[7] X. Che and S. Maag, "A formal passive performance testing approach for distributed communication systems," in Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering, 2013, pp. 74–84.

[8] W. Wei, K. Suh, B. Wang, Y. Gu, J. F. Kurose, D. F. Towsley, and S. Jaiswal, "Passive online detection of 802.11 traffic using sequential hypothesis testing with tcp ack-pairs," IEEE Transactions on Mobile Computing, vol. 8, no. 3, 2009, pp. 398–412.

[9] E. Knauss and D. Damian, "V:issue:lizer: Exploring requirements clarification in online communication over time," in 35th International Conference on Software Engineering (ICSE), 2013, pp. 1327–1330.

[10] H. N. Nguyen, P. Poizat, and F. Zaïdi, "Online verification of value-passing choreographies through property-oriented passive testing," in 14th International IEEE Symposium on High-Assurance Systems Engineering, 2012, pp. 106–113.

[11] G. Morales, S. Maag, A. R. Cavalli, W. Mallouli, E. M. de Oca, and B. Wehbi, "Timed extended invariants for the passive testing of web services," in IEEE International Conference on Web Services (ICWS), 2010, pp. 592–599.

[12] G. Fraser, F. Wotawa, and P. Ammann, "Testing with model checkers: a survey," Software Testing and Verification Reliability, vol. 19, no. 3, 2009, pp. 215–261.

[13] F. Belli and M. Linschulte, "Event-driven modeling and testing of real-time web services," Service Oriented Computing and Applications, vol. 4, no. 1, 2010, pp. 3–15.

[14] F. Lalanne, X. Che, and S. Maag, "Data-centric property formulation for passive testing of communication protocols," in Proceedings of the 13th IASME/WSEAS, ser. ACC'11/MMACTEE'11, 2011, pp. 176–181.

[15] A. Horn, "On sentences which are true of direct unions of algebras," Journal of Symbolic Logic, vol. 16, no. 1, 1951, pp. 14–21.

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

[17] R. E. Noonan, "An algorithm for generating abstract syntax trees," Computer Languages, vol. 10, no. 3-4, 1985, pp. 225–236.

[18] M. R. Brown and R. E. Tarjan, "A fast merging algorithm," Journal of the ACM, vol. 26, no. 2, 1979, pp. 211–226.

[19] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, 1995, pp. 273–297.

[20] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, and J. Peterson, "Sip: Session initiation protocol," 2002.

[21] "Zoiper," http://www.zopier.com/softphone/, 2014.

[22] "Fonality," http://www.fonality.com, 2014.

 ISBN: 978-1-61208-354-4