Annals of Telecommunications manuscript No.
(will be inserted by the editor)

A survey on formal active and passive testing with applications to the cloud

Ana R. Cavalli · Teruo Higashino · Manuel Núñez

Received: date / Accepted: date

1 Introduction

Testing has become an integral part of innovation, production and operation of systems. The activity of testing is already a flourishing area with the active participation of a large community of researchers and experts who are highly aware of its importance and impact for the future deployment and use of software and systems. Formal methods have proved to be very promising for developing automated and generic testing methods. Actually, the combination of formal methods and testing is currently well understood and tools to automate testing activities are widely available.

(Formal) testing is the assessment, by means of experiments, that a product conforms to its (formal) requirements. Test cases are designed to test particular aspects of the system, called test objectives. In order to formally obtain testing objectives, it is necessary to provide mathematical models for the semantic of the studied system, formal frameworks for testing, languages to describe the expected properties or requirements of the system in a precise and unambiguous way and methods to generate the expected test cases. In addition, active methods require deploying a test environment (test architecture) to execute test cases and to observe implementation reactions. They may also interrupt the system normal functioning arbitrarily, for example by resetting it after each test case execution. However, when a system is deployed in an integrated environment, it becomes quite difficult to access it. Moreover, active methods may disturb the natural operation of the implementation under test. So, these ones may not be suitable in regards to the tested system. Passive testing represents another interesting alternative, which offers several advantages, for example, to not disturb the system while testing.

In this paper we present a survey covering the main approaches to formal testing. We have divided the survey in three parts. First, we distinguish between active and passive approaches to formal testing. Next, we review some of the work on testing the cloud and on testing in the cloud.

Ana R. Cavalli
Télécom SudParis
9 Rue Charles Fourier, 91000, Evry, France
Tel.: +33 1 60 76 44 27
Fax: +33 1 60 76 47 11
E-mail: ana.cavalli@it-sudparis.eu

Teruo Higashino
Graduate School of Information Science and Technology
Osaka University, Japan
Tel.: +81 6 6879 4555
Fax: +81 6 6879 4558
E-mail: higashino@ist.osaka-u.ac.jp

Manuel Núñez
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
E-mail: mn@sip.ucm.es

2 Active testing

Most networked systems, especially communication protocols, are reactive systems. For each user input or packet from another network component, a target system provides the corresponding reactions and waits a new input/packet. Thus, the specifications of such reactive systems are often modeled as the finite state machines (FSMs). In general, since communication is essentially unreliable, many specifications of such networked systems contain several exception handling and timeout mechanisms. Therefore, the total number of states of such a FSM based specification becomes rather

large. In order to design and develop high reliable networked systems, we need to guarantee the reliability of developed implementations of such networked systems. However, it is rather difficult to mechanically reproduce the situations corresponding to all exception handling and timeout mechanisms in real environment and test whether the corresponding reactions for all the inputs are carried out correctly. Therefore, a lot of methods for conformance testing of communication protocols modeled as FSMs have been developed, especially in 1980s and 1990s (for survey, see [21,40,45,69]). Those testing methods for FSMs are classified as active testing. In active testing, a tester provides inputs/packets to each targeted implementation ("Implementation Under Test (IUT)"), and checks whether the corresponding outputs from the IUT are the expected ones.

There are two types of methods in active testing: functional (black-box) testing and structural (white-box) testing. In black-box testing, each implementation is considered as a black-box where for given requirements of a system (specification), test cases from the requirements are derived to check the correctness of their implementation. On the other hand, in white-box testing, test cases are derived from the program of a given implementation using each testing criteria. Examples of testing criteria are (i)all paths are executed in the program, (ii)all conditions in the program are correctly branched, (iii)all uses of variables are correctly implemented, and so on.

Conformance testing is a black-box testing approach where a fault model is given. The most popular fault model is that the targeted IUT is assumed to be modeled as a FSM with the same number of states of a given FSM based specification. The number of the states of a given IUT might be larger than that of the FSM based specification. Under such an assumption, for a given pair of a formal specification *Spec* (requirements) of a system and its black-box implementation *Imp*, we derive test cases from *Spec* to determine if *Imp* conforms to *Spec* where the word "conform" has many definitions depending on its formal models. Examples of conformance relations are "equivalence" and "reduction relations". The "equivalence" conformance relation means that two FSMs are equivalent if they produce the same output sequences for every possible input sequence from their initial states. In FSM based testing, we usually consider the output and transfer faults. The output fault means that a given IUT generates a different output from its specification while the transfer fault means that a given IUT reaches a state generating a different output sequence from its specification.

For FSM based conformance testing, there are a lot of testing methods such as W [18], Wp [25], DS [28], UIO [67], UIOv [80], HSI [62] and H [20] methods. In W and Wp methods, a characterization set W is derived where any two different states of a reduced FSM are distinguished by at least one sequence in the set W. DS method corresponds to a special case of W method where one distinguishing sequence (DS) distinguishes all states of a given FSM. Note that the distinguishing sequence does not always exist for any reduced FSM. In UIO and UIOv methods, for each state of a given FSM, a single sequence called UIO sequence can be used for distinguishing the state from all other states. The UIO sequence does not always exist for any state of a reduced FSM. HSI and H methods are extensions of W and Wp methods where unreduced FSMs are treated.

In general, networked systems are often updated. Thus, their specifications might often be changed. In such a case, depending on the change of a given specification, the corresponding IUT also needs to be modified. Even if the original IUT is confirmed to the original specification, the modified IUT might not confirm to the changed specification. Since it takes much time for each conformance testing when the specification is complicated, we need some devices for efficient conformance testing. For this purpose, incremental testing methods have been studied where incremental test cases are generated to check that the modified parts of the specification are correctly implemented in the modified IUT. As the types of modifications of specifications, (i) modification of outputs of transitions, (ii) modification of ending states of transitions, (iii) addition/deletion of transitions and (iv) addition/deletion of states are considered. In [23], the authors show that incremental testing methods are more efficient than complete conformance testing methods if the modified part is less than 20 % of the whole specification.

From the late 1990s, high-speed networks and multimedia communication systems have become popular. Also, from the early 2000s, wireless communication networks are widely used. In those network systems, we need to consider contents of data and their real-time constraints. In order to treat such aspects, extended FSM (EFSM) based conformance testing methods have been proposed [22,33,47,81]. EFSM based conformance testing uses mathematical techniques such as "unfolding", "abstraction" and "separation". Unfolding transforms a given EFSM into an equivalent FSM where the values of variables are mapped into the control states. Abstraction transforms a given EFSM into a FSM where each "if_then_else" statement is abstractly transformed into simple two branches. Separation considers control flows and data flows of a given EFSM separately where FSM based testing methods are applied for the control flows and software testing methods

are used for testing the data flows. Some EFSM based methods restrict a class of transition conditions and variables. For example, in [33,47] the authors restrict the transition conditions and operations of variables to addition, subtraction and linear inequalities ($+, -, =, <, >$) on integers, and apply integer linear programming (ILP) techniques for finding satisfiable values in a given test sequence with variables.

Recent high-speed networks and multimedia communication systems have several types of timing constraints. In order to test such real-time systems, the authors of [2] have proposed the timed automata model where the predicates with multiple clock variables and parameters can be specified as transition conditions of an automaton. In general, such a predicate is represented as a logical formula of linear inequalities. In [72], the authors have proposed testing methods for the timed automata. In [24], the authors introduce a timed I/O automaton (TIOA) model and consider timing faults, transfer faults and output faults. They classify a given TIOA into the control part and clock part, transform them into the corresponding non-deterministic FSM, and apply the characterization set for the control part. Since the exact timing of the clock part at which the IUT responds with outputs remains unknown, such uncertainty is modeled as a non-deterministic FSM. In [34], the authors propose a similar timed I/O automaton model with data values. In order to trace a test sequence (I/O sequence) on the timed I/O automaton, we need to execute all I/O actions in the test sequence at adequate execution timing which satisfy all timing constraints in the test sequence. However, since outputs are given from IUTs and uncontrollable, we cannot designate their output timing in advance. Also their output timing affects the executable timing for the succeeding I/O actions in the test sequence. The authors of [34] propose a method for generating an executable timing of each input action in a test sequence independently of its preceding output actions' timing.

In multimedia communication systems, QoS functional testing is also important [66,73,78]. The subjects of those QoS functional testing methods are to check (i) play-back quality of each media object (frame rate fluctuation, etc.), (ii) temporal relationship among objects (lip synchronization, etc.), and (iii) timeliness (delay, play-back speed, etc.). The test procedure of QoS functional testing is to (i) transmit packets to the IUT at various timing and observe outputs from the IUT for a long time interval, (ii) measure the frame rate (time lag between two media objects) every short interval (e.g., 1sec.), (iii) register the measured value as a sampling to get a statistical distribution of the values, and (iv) determine those testing results statistically. In QoS functional testing, for a specified probability P, we treat that a given IUT is correct when the execution sequences satisfy the constraints specified in the specification at a probability P' such that $P' \geq P$.

Recently, WiFi is becoming very popular. In WiFi, a base station controls several mobile terminals. Such a system is similar to client-server systems. Thus, testing methods for client-server systems can be used for WiFi based systems. On the other hand, ad hoc networks (including MANET and VANET) and sensor networks consist of several mobile terminals, and there are no infrastructure nor control nodes. Therefore, we need new testing methods for ad hoc networks [16,35,49]. On the other hand, the performance and reliability of multihop communications strongly depends on node density distributions. Thus, for interoperability testing and end-to-end performance testing, mainly passive testing methods can be used. This is discussed in the next section.

3 Passive testing

Since the early 1980s there has been an increasing interest in passive testing, technically defined as a testing activity in which a tester does not influence (stimulate) an implementation under test in any way, it does not apply any test stimuli. Rather, the usual approach of passive testing consists on recording the trace (for instance, the sequence of exchange of messages) produced by the implementation under test and mapped to the property to be tested or specification if it exists. Passive testing helps to observe abnormal behavior in the implementation under test on the basis of observing any deviation from the predefined behavior. Moreover, it is usually considered that the implementation is taken without knowledge of its internal state that is to say that we do not consider the event trace record to start from the initial state or a predefined state. In general, the specification has the form of a finite state machine (FSM) and the work consists in verifying that the executed trace is accepted by the FSM specification. A drawback of the first approaches is the low performance of the proposed algorithms in terms of complexity and especially if non deterministic specifications are considered.

New approaches have been proposed: a set of properties, called *invariants*, are extracted from the system specification and checked on the traces observed from the implementation to test their correctness. In this approach, information will be extracted from the specification and then used to process the trace. New algorithms are applied that check the properties on the real

implementation traces. These algorithms are adaptations of classical algorithms for pattern matching [76].

Another innovation proposed these last years is the combination of passive testing and fault or attacks injection techniques. Faults or attacks are be injected in the code and passive testing techniques will be used to analyse their impact.

One of the first works on passive testing has been on its application to network fault management [44]. In this approach, the faults of a network protocol are detected by passively observing its input/output behaviours without interrupting the normal network operations. The authors introduce methods for passive fault detection in deterministic and non-deterministic FSMs. The proposed technique is applied to a signaling network protocol operating under Signaling System 7 and report on experimental results, which show the feasibility of applying passive testing to practical systems. This work had a strong impact in other works on the application of passive testing based on FSMs [83,86,77] and EFSMs [74,42,1,43,76,10] and to systems specified as communicating FSMs (CFSM) [53–55]. In particular, in [86] the authors have proposed an approach that provides information about possible starting states and possible trace compatibility with the observed I/O behavior at the end of passive fault detection. In addition, the proposed approach utilizes a Hybrid method to evaluate constraints in predicates associated with transitions in an EFSM which combines the use of both Interval Refinement and Simplex methods for performance improvement during passive fault detection

Many passive testing techniques consider only the control part of the system and neglect data, or are confronted with an overwhelming amount of data values to process. Passive testing using EFSM, mainly concentrates on checking the correctness of event sequences (appearing in the collected trace), but it must also consider the variables and the parameter values. In [74], the authors have proposed the first passive testing method based on deducing the variable and parameter values from an event trace with EFSM specifications. In this approach, the algorithm suffers from an information loss problem. To overcome this issue, a more efficient passive testing approach for fault detection is proposed in [42]. In this paper, the authors present two algorithms by using an event-driven EFSM. First, an effective passive testing algorithm for EFSMs is proposed; second, an algorithm based on variable determination with the constraints on variables is presented. This algorithm allows users to trace the values of variables as well as the system state. One drawback of this approach is that not all transfer errors can be detected. Based on the works of [42], the authors of [1] developed a similar approach of passive testing but following the trace in the backward direction. In this approach the partial trace is processed backward to narrow down the possible specifications. The algorithm performs two steps. It first follows a given trace backward, from the current configuration to a set of starting ones, according to the specification. The goal is to find the possible starting configurations of the trace, which leads to the current configuration. Then, it analyzes the past of this set of starting configurations, also in a backward manner, seeking for configurations in which the variables are determined. When such configurations are reached, a decision is taken on the validity of the studied paths (traces are completed). Such an approach is usually applied as a complement to forward checking to detect more errors. This new algorithm was applied to the Simple Connection Protocol (SCP) that allows connecting two entities after a negotiation of the quality of service required for the connection. In [10], the backward and forward methods are combined for online passive testing of web services. Here, the algorithm attempts to find a set of candidates in the past of the trace, that matches the observed event. Using those information, passive fault-detection is carried out, using the forward approach.

In [57], control and data parts are considered by integrating the concepts of symbolic execution and by improving trace analysis by introducing trace slicing techniques. Properties are described using Input Output Symbolic Transition Systems (IOSTSs). These properties can be designed to test the functional conformance of a protocol as well as security properties. In IOSTS, the parameters and variable values are represented by symbolic values (called fresh variables) instead of concrete ones. Enumeration of data values is therefore not required. This allows to reduce the huge amount of data values commonly applied in many passive testing approaches. Besides, in this work a parametric trace slicing technique is used for trace analysis. Trace analysis plays a very important part in passive testing. A parametric trace is defined as a trace containing events with parameters that have been bound to a concrete data value (i.e., valuation) and parametric trace slicing is defined as a technique to slice (or cut) the real protocol execution trace into various slices based on the valuation. Each slice corresponds to a particular valuation. These trace slices merged together constitutes the execution trace. We then apply the symbolic execution of our properties on the trace slices to provide a test verdict. The proposed symbolic passive testing approach was implemented in a tool called TestSym-P and applied to two different protocols: Session Initiation Protocol (SIP) and Bluetooth Protocol. A more extended

version of the approach is provided in the Journal article [59] to specify the protocol properties as well as several kinds of attack patterns. This helps to detect conformance as well as security anomalies. In [60] it is presented an interesting online verification technique of service choreographies , which takes into account complex data constraints. However, they assume that the IUT conforms to the model (which are based on Symbolic Transition Graph with Assignments (STGA)) and also they prove the scalability of their approach for a maximum of 20,000 packets.

Some works propose the use of invariants for passive testing. In [6] the authors propose that invariants be provided by the expert/tester. Therefore, the first step consists in checking that invariants are in fact correct with respect to the specification. An algorithm to check this correctness is provided. The complexity, in the worst case, of the algorithm was linear with respect to the number of transitions of the specification. Once a set of (correct) invariants is available, the second step consists in checking whether the trace produced by the IUT matched the invariants. In order to do so, a simple adaptation of the classical algorithms for pattern matching on strings was implemented. This work was extended [8] to study a new type of invariants (obligation), to present a tool that implements the approach and to give a complete case study on the Wireless Application Protocol. It is worth pointing out that this protocol represents a typical example where active testing cannot be applied because, in general, there is no direct access to the interfaces between the different layers. Thus, testers cannot control how internal communications are established.

However, most of the described invariant based testing approaches are derived from the FSM formalism, where only control parts are considered. Then, the authors of [15] proposed a method to extract the constraint information separately from the involved transitions in addition to extraction of control sequences. According to their approach the correct sequence must be found and the constraints must hold true, otherwise a fault is detected. In another work [6] proposes a minor modification to the obligation invariant, to deal with constant data parameters. This approach paved the way for another work by the authors of [39] to extend the simple and obligation invariant to match the EFSM formalism. In [8] the authors assume that the current states of the observed trace are known. In [58] these assumptions are not required. Moreover, points of observation are set in a black-box framework that does not allow any homing phase. Since no formal specification of the implementation is provided, the extracted traces are not related to any known states. Another recent interesting work on invariants was proposed by the authors in [41]. In this approach, the authors discuss the importance of testing for data relations and constraints between exchanged messages and they also show how they can be tested directly on traces using logic programming.

Monitoring is based on passive testing, i.e. the observation of the system traces without interfering with the system's normal operation [8,3]. Monitoring is also close to *run time verification* [46]. The goal of monitoring is to obtain improved visibility of the studied communicating or inter-operating systems. Many techniques can be applied but they all suppose the definition of a monitoring architecture (including the selection of observations points to collect the traces) and the description of the system security requirements using, for instance, formal specification languages. Different types of monitoring can be considered, in this paper we consider the following ones: i) monitoring of network communication between inter-operating systems (that we will refer to as blackbox monitoring); and, ii) the monitoring of application execution (that we will refer to as white-box monitoring).

Different techniques for network monitoring exist in the literature and can be based on, for instance, SNMP [68,85], Deep Packet Inspection (DPI) [19] and invariants [56]. DPI is a technique that is used for completely analyzing communication packets (both headers and payloads) and can be useful for security analysis of network traffic and detecting and preventing intrusions (IDPS, Intrusion Detection and Prevention Systems) [52]. Most techniques used depend on pattern marching to detect intrusions or attacks (e.g. SNORT) [71] but very few use correlation of events to generate alarms (e.g. BRO) [11].

White-box monitoring is similar to run-time verification where the application is analysed during its execution. Several techniques exist, including code instrumentation using: just-in-time compilation (e.g. Valgrind) [79], debugging tools (e.g. GDB) [27], etc. [64] proposes the use of aspect-oriented monitoring approaches for validation and testing of a software against constraints specified on an associated UML design model.

Monitoring can be performed online and offline. In the former, the passive tester tries to detect a fault during the execution of the system [29,70], whereas, in the latter, the evaluation of the system is done by collecting the recorded traces [65,1,4,84].

Property modeling for monitoring have been defined and adapted from currently used formalisms, some of the works using trace analysis techniques: TIC [50] and TIPS [56] are based on the analysis of, respectively the execution traces of a system and the exchanged commu-

nication packets. They aim at determining if real traces of the system respect functional and security requirements.

Some research works have been dedicated to passive testing of timed systems. In [5] it is presented a formal passive testing methodology for timed systems. The paper presents two algorithms to check the correctness of proposed invariants with respect to a given specification and algorithms to check the correctness of a log, recorded from the implementation under test, with respect to an invariant. The soundness of this methodology is shown by relating it to an implementation relation. A tool, called PASTE, which implements the proposed algorithms is also proposed.

An interesting work has been performed on the passive testing of asynchronous systems. The work presented in [30] is focused in the analysis, based on passive testing, of the asynchronous communication between the system and the tester taking into account the different observations that can be expected due to the assumption of asynchrony. The proposed technique checks properties against traces in polynomial time, with a low need for storage. This technique is well adapted to real-time systems.

Different tools have been proposed to automate all the phases of the passive testing approach. In particular, these tools include the algorithms to check the correctness of invariants with respect to the specification and to decide whether the trace observed from the implementation respects the invariants. MMT is a monitoring tool developed by MI [82]. This tool allows capturing and analysing network traffic and application traces. It can be used to verify functional properties, QoS and security properties and is composed of MMT-Capture probes and a MMT-Operator application that allow deploying a mixed distributed/centralized network monitoring solution.

4 Testing (in) the cloud

In this short section we consider some of the work on testing (in) the cloud. Although the special issue targeted the broad scope of testing distributed and networked systems, where formal testing is well established [48, 36, 13, 63, 75, 31, 32], cloud computing represents the natural evolution of these systems and work on testing is in their initial stages. First, we would like to mention the difference between two apparently similar concepts: testing *in the cloud* and testing *the cloud*. The former one targets the validation of applications, environments and infrastructures that are available on a cloud environment. This ensures the correct operation of each part of the cloud system against the expectations of the cloud computing business model. Cloud testing, or Testing as a Service (in short, TaaS), involves using cloud infrastructures in testing products and services. In other words, *cloud testing* can be seen as a service while *testing a cloud* is an inexorable activity that must be undertaken for any application developed to be executed in a cloud system [51].

Although testing (in) the cloud is a flourishing activity, there is almost no work on formal testing. First, this special issue includes a paper by Alberto Núñez and Robert M. Hierons where the authors present a formal methodology for validating cloud models using metamorphic testing. Another paper dealing with formal testing of cloud systems presents a formalism where a computing cloud is modelled as a graph, computing resources, such as services and intellectual property access rights, are attributes of a graph node, and the use of a resource is modelled as a predicate on an edge of the graph [17]. Cloud computations are modelled as a set of paths in a subgraph of the cloud such that every edge contains a predicate that must hold to enable the edge. This formalism also includes a family of model-based testing criteria to support the testing of cloud applications.

Next we review some work on testing cloud systems although they do not consider formal methods. One approach for testing cloud systems executes a real virtual machine instead of a model that mimics its underlying behaviour. D-Cloud [7] is a software testing environment that manages virtual machines and includes fault injection capabilities. Basically, D-Cloud sets up a test environment on cloud resources using a given system configuration file and automatically executes several tests according to a given scenario. D-Cloud has been built on top of Eucalyptus and uses QEMU [9] to build virtual machines that simulate faults in parts of the hardware including disk, network and memory. PreFail [37] is a programmable and efficient failure testing framework where testers can express a variety of failure exploration policies, skip redundant fault-injection tests, run failure testing in parallel, and reduce the time to debug failed test runs. Unlike D-Cloud, that provides simulated *actual* faults, PreFail inserts a *failure surface* between the target system and the OS library.

In recent years, simulation has become a widely adopted loosely formalised approach for testing cloud systems. The developer builds a simulation model that imitates the behaviour of the target system and then different measures, like performance and power consumption, are gathered by running simulations. Researchers have designed cloud models and then performed ad-hoc testing by manually simulating different scenarios and comparing obtained results. Among the available

simulation tools that can be used to model and simulate cloud computing environments are CloudSim [12], GreenCloud [38], SimGrid [14], Virtual-GEMS [26] and iCanCloud [61].

Acknowledgements This research has been partially supported by the Spanish MEC project ESTuDIo (TIN2012-36812-C02), the Comunidad de Madrid project SICOMORo-CM (S2013/ICE-3006) and the UCM-Santander program to fund research groups (group 910606).

References

1. Alcalde, B., Cavalli, A., Chen, D., Khuu, D., Lee, D.: Network protocol system passive testing for fault management: A backward checking approach. In: Proc. of 24th IFIP Int. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE'04), *Lecture Notes in Computer Science*, vol. 3235, pp. 150–166. Springer (2004)
2. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126**, 183–235 (1994)
3. Andrés, C., Cambronero, M., Nuñez, M.: Formal passive testing of service-oriented systems. In: Proc. of 2010 IEEE Int. Conf. on Services Computing (SCC 2010), pp. 610–613 (2010)
4. Andrés, C., Merayo, M., Núñez, M.: Passive testing of timed systems. In: Proc. of 6th Int. Symp. on Automated Technology for Verification and Analysis (ATVA 2008), *Lecture Notes in Computer Science*, vol. 5311, pp. 418–427. Springer (2008)
5. Andrés, C., Merayo, M.G., Núñez, M.: Formal passive testing of timed systems: theory and tools. Software Testing, Verification and Reliability **22**(6), 365–405 (2012)
6. Arnedo, J.A., Cavalli, A., Núñez, M.: Fast testing of critical properties through passive testing. In: Proc. of 15th IFIP Int. Conf. on Testing of Communicating Systems (TestCom 2003), *Lecture Notes in Computer Science*, vol. 2644, pp. 295–310. Springer (2003)
7. Banzai, T., Koizumi, H., Kanbayashi, R., Imada, T., Hanawa, T., Sato, M.: D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology. In: Proc. of 10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing (CCGrid'10), pp. 631–636 (2010)
8. Bayse, E., Cavalli, A., nez, M.N., Zaïdi, F.: A passive testing approach based on invariants: application to the wap. Computer Networks **48**(2), 247 – 266 (2005)
9. Bellard, F.: Qemu, a fast and portable dynamic translator, a fast and portable dynamic translator. In: Proc. of 2005 USENIX Annual Technical Conference (ATEC '05), pp. 41–41. USENIX Association (2005)
10. Benharref, A., Dssouli, R., Serhani, M., En-Nouaary, A., Glitho, R.: New approach for efsm-based passive testing of web services. In: Proc. of 19th IFIP Int. Conf. on Testing of Software and Communicating Systems (TestCom 2007), *Lecture Notes in Computer Science*, vol. 4581, pp. 13–27. Springer (2007)
11. http://bro-ids.org/ (2014)
12. Buyya, R., Ranjan, R., Calheiros, R.N.: Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In: Proc. of 7th IEEE Int. Conf. High Performance Computing and Simulation (HPCS'09), pp. 1–11 (2009)
13. Cacciari, L., Rafiq, O.: Controllability and observability in distributed testing. Information and Software Technology **41**(11–12), 767–780 (1999)
14. Casanova, H., Legrand, A., Quinson, M.: SimGrid: A generic framework for large-scale distributed experiments. In: Proc. of 10th Int. Conf. on Computer Modeling and Simulation (UKSIM' 08), pp. 126–131 (2008)
15. Cavalli, A., Gervy, C., Prokopenko, S.: New approaches for passive testing using an extended finite state machine specification. Information and Software Technology **45**(12), 837 – 852 (2003)
16. Cavalli, A., Grepet, C., Maag, S., Tortajada, V.: A validation model for the dsr protocol. In: Proc. of IEEE Int. Workshop on Wireless Ad Hoc Networking (WWAN'04), pp. 768–773 (2004)
17. Chan, W., Mei, L., Zhang, Z.: Modeling and testing of cloud applications. In: Proc. of 4th IEEE Int. Conf. on Asia-Pacific Services Computing (APSCC'09), pp. 111–118 (2009)
18. Chow, T.S.: Test design modeled by finite-state machines. IEEE Trans. on Software Engineering **4**(3), 178–187 (1978)
19. Corwin, E.H.: Deep packet inspection: Shaping the internet and the implications on privacy and security. Information Security Journal: A Global Perspective **20**(6), 311–316 (2011)
20. Dorofeeva, M., El-Fakih, K., Yevtushenko, N.: An improved conformance testing method. In: Proc. of 25th IFIP Int. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE'05), pp. 363–378. Springer (2005)
21. Dorofeeva, R., El-Fakih, K., Maag, S., Cavalli, A.R., Yevtushenko, N.: Fsm-based conformance testing methods: A survey annotated with experimental evaluation. Information and Software Technology **52**, 1286–1297 (2010)
22. Duale, A.Y., Uyar, M.U.: A method enabling feasible conformance test sequence generation for efsm models. IEEE Trans. on Computers **53**(5), 614–627 (2004)
23. El-Fakih, K., Yevtushenko, N., v. Bochmann, G.: Fsm-based incremental conformance testing methods. IEEE Trans. on Software Engineering **30**(7), 425–436 (2004)
24. En-Nouaary, A., Dssoul, R., Khendek, F., Elqortobi, A.: Timed test cases generation based on state characterization technique. In: Proc. of 19th IEEE Real-Time System Sympsium (RTSS'98), pp. 220–229. IEEE (1998)
25. Fujiwara, S., v. Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. IEEE Trans. on Software Engineering **17**(6), 591–603 (1991)
26. García-Guirado, A., Fernández-Pascual, R., García, J.M.: Virtual-GEMS: An infrastructure to simulate virtual machines. In: Proc. of 5th annual workshop on modeling, benchmarking and simulation (MoBS'09), pp. 1–10 (2009)
27. `http://www.gnu.org/software/gdb/` (2014)
28. Gonenc, G.: A method for the design of fault detection experiments. IEEE Trans. on Computers **C-19**(6), 551–558 (1970)
29. Hallé, S., Villemaire, R.: Runtime monitoring of web service choreographies using streaming xml. In: 2009 ACM Symposium on Applied Computing, SAC'09, pp. 2118–2125 (2009)
30. Hierons, R., Merayo, M., Núñez, M.: Passive testing with asynchronous communications. In: Proc. of 33rd IFIP Int. Conf. on Formal Techniques for Distributed Systems (FORTE'13), *Lecture Notes in Computer Science*, vol. 7892, pp. 99–113. Springer (2013)

31. Hierons, R.M.: Oracles for distributed testing. IEEE Trans. on Software Engineering **38**(3), 629–641 (2012)
32. Hierons, R.M., Merayo, M.G., Núñez, M.: Implementation relations and test generation for systems with distributed interfaces. Distributed Computing **25**(1), 35–62 (2012)
33. Higashino, T., v. Bochmann, G.: Automatic analysis and test case derivation for a restricted class of lotos expressions with data parameters. IEEE Trans. on Software Engineering **20**(1), 29–42 (1994)
34. Higashino, T., Nakata, A., Taniguchi, K., Cavalli, A.R.: Generating test cases for a timed i/o automaton model. In: Proc. of 12th IFIP Int. Workshop on Testing of Communicating Systems (IWTCS'99), pp. 197–214. Springer (1999)
35. Hiromori, A., Umedu, T., Yamaguchi, H., Higashino, T.: Protocol testing and performance evaluation for manets with non-uniform node density distribution. In: Proc. of 24th IFIP Int. Conf. on Testing Software and Systems (ICTSS 2012), pp. 231–246. Springer (2012)
36. Jard, C., Jéron, T., Kahlouche, H., Viho, C.: Towards automatic distribution of testers for distributed conformance testing. In: Proc. of Int. Conf. on Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE'98/PSTV'98), pp. 353–368. Kluwer Academic Publishers (1998)
37. Joshi, P., Gunawi, H., Sen, K.: PREFAIL: a programmable tool for multiple-failure injection. ACM SIGPLAN Notices **46**(10), 171–188 (2011)
38. Kliazovich, D., amd S. U. Khan, P.B.: GreenCloud: A packet-level simulator of energy-aware cloud computing data centers. The Journal of Supercomputing **62**(3), 1263–1283 (2012)
39. Ladani, B., Alcalde, B., Cavalli, A.: Passive testing: A constrained invariant checking approach. In: Proc. of 17th IFIP Int. Conf. on Testing of Communicating Systems (TestCom 2005), *Lecture Notes in Computer Science*, vol. 3502, pp. 9–22. Springer (2005)
40. Lai, R.: A survey of communication protocol testing. The Journal of Systems and Software **62**, 21–46 (2002)
41. Lalanne, F., Maag, S.: A formal data-centric approach for passive testing of communication protocols. IEEE/ACM Trans. on Networking **21**(3), 788–801 (2013)
42. Lee, D., Chen, D., Hao, R., Miller, R., Wu, J., Yin, X.: A formal approach for passive testing of protocol data portions. In: Proc. of 10th IEEE Int. Conf. on Network Protocols (ICNP 2002), pp. 122–131 (2002)
43. Lee, D., Chen, D., Hao, R., Miller, R., Wu, J., Yin, X.: Network protocol system monitoring-a formal approach with passive testing. IEEE/ACM Trans. on Networking **14**(2), 424–437 (2006)
44. Lee, D., Netravali, A., Sabnani, K., Sugla, B., John, A.: Passive testing and applications to network management. In: Proc. of 5th IEEE Int. Conf. on Network Protocols (ICNP'97), pp. 113–122 (1997)
45. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE **84**(8), 1090–1123 (1996)
46. Leucker, M., Schallhart, C.: A brief account of runtime verification. The Journal of Logic and Algebraic Programming **78**(5), 293 – 303 (2009)
47. Li, X., Higashino, T., Higuchi, M., Taniguchi, K.: Automatic generation of extended uio sequences for communication protocols in an efsm model. In: Proc. of 7th IFIP Int. Workshop on Protocol Test Systems (IWPTS'94), pp. 225–240. Springer (1994)
48. Luo, G., Dssouli, R., Bochmann, G.v.: Generating synchronizable test sequences based on finite state machine with distributed ports. In: Proc. of 6th IFIP Int. Workshop on Protocol Test Systems (IWPTS'93), pp. 139–153. North-Holland (1993)
49. Maag, S., Zaidi, F.: Testing methodology for an ad hoc routing protocol. In: Proc. of 2006 ACM Int. Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks, pp. 48–55 (2006)
50. Mammar, A., Cavalli, A., Jimenez, W., Mallouli, W., Montes, E.: Using testing techniques for vulnerability detection in c programs. In: Proc. of 23rd IFIP Int. Conf. on Testing Software and Systems (ICTSS'11), *Lecture Notes in Computer Science*, vol. 7019, pp. 80–96. Springer (2011)
51. Mehrotra, N.: Cloud-Testing vs. Testing a Cloud. In: Proc. 10th Int. Software Testing Conf., pp. 1–7 (2010)
52. Mell, P., Scarfone, K.: Guide to intrusion detection and prevention systems (IDPS). http://csrc.nist.gov/publications/ nistpubs/800-94/SP800-94.pdf (2007)
53. Miller, R.: Passive testing of networks using a cfsm specification. In: Proc. of 1998 IEEE Int. Conf. on Performance, Computing and Communications (IPCCC '98), pp. 111–116 (1998)
54. Miller, R., Arisha, K.: On fault location in networks by passive testing. In: Proc. of 2000 IEEE Int. Conf. on Performance, Computing, and Communications (IPCCC '00), pp. 281–287 (2000)
55. Miller, R., Arisha, K.: Fault identification in networks by passive testing. In: Proc. of 34th Simulation Symp., pp. 277–284 (2001)
56. Morales, G., Maag, S., Cavalli, A., Mallouli, W., Montes, E., Wehbi, B.: Timed extended invariants for the passive testing of web services. In: Proc. of 2010 IEEE Int. Conf. on Web Services (ICWS 2010), pp. 592–599 (2010)
57. Mouttappa, P., Maag, S., Cavalli, A.: An iosts based passive testing approach for the validation of data-centric protocols. In: Proc. of 12th Int. Conf. on Quality Software (QSIC 2012), pp. 49–58 (2012)
58. Mouttappa, P., Maag, S., Cavalli, A.: Monitoring based on iosts for testing functional and security properties: Application to an automotive case study. In: Proc. of 37th IEEE Int. Conf. on Computer Software and Applications (COMPSAC '13), pp. 1–10 (2013)
59. Mouttappa, P., Maag, S., Cavalli, A.: Using passive testing based on symbolic execution and slicing techniques: Application to the validation of communication protocols. Computer Networks **57**(15), 2992 – 3008 (2013)
60. Nguyen, H., Poizat, P., Zaidi, F.: Online verification of value-passing choreographies through property-oriented passive testing. In: Proc. of 14th IEEE Int. Symp. on High-Assurance Systems Engineering (HASE 2012), pp. 106–113 (2012)
61. Núñez, A., Vázquez-Poletti, J.L., Caminero, A.C., Castañé, G.G., Carretero, J., Llorente, I.M.: iCanCloud: A flexible and scalable cloud infrastructure simulator. Journal of Grid Computing **10**(1), 185–209 (2012)
62. Petrenko, A., Yevtushenko, N., Lebedev, A., Das, A.: Nondeterministic state machines in protocol conformance testing. In: Proc. of 6th IFIP Int. Workshop on Protocol Test Systems (IWPTS'93), pp. 363–378. North-Holland (1993)
63. Rafiq, O., Cacciari, L.: Coordination algorithm for distributed testing. The Journal of Supercomputing **24**(2), 203–211 (2003)

64. Richters, M., Gogolla, M.: Aspect-oriented monitoring of UML and OCL constraints. In: 4th Int. Workshop on Aspect-Oriented Modeling with UML on 6th Int. Conf. on the Unified Modeling Language (UML'03). (2003)
65. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.E., et al.: Object-oriented modeling and design. Prentice-hall Englewood Cliffs, NJ (1991)
66. S. C. Cheung, S.T.C., Xu, Z.: Toward generic timing tests for distributed multimedia software systems. In: Proc. of 12th IEEE Int. Symp. on Software Reliability Engineering (ISSRE'01), pp. 210–220 (2001)
67. Sabnani, K., Dahbura, A.: A protocol test generation procedure. Computer Networks ISDN Systems **15**(4), 285–297 (1988)
68. Shin, K.S., Jung, J.H., Cheon, J.Y., Choi, S.B.: Real-time network monitoring scheme based on SNMP for dynamic information. Journal of Network and Computer Applications **30**(1), 331 – 353 (2007)
69. Sidhu, D.P., Leung, T.K.: Formal methods for protocol testing: a detailed study. IEEE Trans. on Software Engineering **15**(4), 413–426 (1989)
70. Simmonds, J.: Dynamic analysis of web services. Ph.D. thesis, University of Toronto (2011)
71. http://www.snort.org/ (2015)
72. Springintveld, J., Vaandrager, F., D'Argenio, P.R.: Testing timed automata. Theoretical Computer Science **254**(1-2), 225–257 (2001)
73. Sun, T., Yasumoto, K., Mori, M., Higashino, T.: Qos functional testing for multi-media systems. In: Proc. of 23rd IFIP Int. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE'03), pp. 319–334. Springer (2003)
74. Tabourier, M., Cavalli, A.: Passive testing and application to the gsm-map protocol. Information and Software Technology **41**(11-12), 813 – 821 (1999)
75. Ural, H., Williams, C.: Constructing checking sequences for distributed testing. Formal Aspects of Computing **18**(1), 84–101 (2006)
76. Ural, H., Xu, Z.: An efsm-based passive fault detection approach. In: Proc. of 19th IFIP Int. Conf. on Testing of Software and Communicating Systems (TestCom 2007), *Lecture Notes in Computer Science*, vol. 4581, pp. 335–350. Springer (2007)
77. Ural, H., Xu, Z., Zhang, F.: An improved approach to passive testing of fsm-based systems. In: Proc. of 2nd IEEE Int. Workshop on Automation of Software Test (AST '07), p. 6 (2007)
78. V. Misic, S.T.C., Cheung, S.C.: Towards a framework for testing distributed multimedia software systems. In: Proc. of 1998 IEEE Int. Symp. on Software Engineering for Parallel and Distributed Systems (PDSE'98), pp. 72–81 (1998)
79. `http://valgrind.org/` (2014)
80. Vuong, S.T., Chan, W., Ito, M.: The uiov-method for protocol test sequence generation. In: Proc. of 2nd IFIP Int. Workshop on Protocol Test Systems (IWPTS'89), pp. 161–175. North-Holland (1989)
81. Wang, C.J., Liu, M.T.: Axiomatic test sequence generation for extended finite state machines. In: Proc. of 12th IEEE Int. Conf. on Distributed Computing Systems (ICDCS-12), pp. 252–259 (1992)
82. Wehbi, B., Montes, E., Bourdelles, M.: Events-based security monitoring using mmt tool. In: Proc. of 5th IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2012), pp. 860–863 (2012)
83. Wu, J., Zhao, Y., Yin, X.: From active to passive: Progress in testing of internet routing protocols. In: Proc. of 22nd IFIP Int. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE 2002), pp. 101–116. Springer (2002)
84. Xiaoping, C., Lalanne, F., Maag, S.: A logic-based passive testing approach for the validation of communicating protocols. In: Proc. of 7th Int. Conf. on Evaluation of Novel Approaches to Software Engineering (ENASE'12), pp. 53–64 (2012)
85. Zeng, W., Wang, Y.: Design and implementation of server monitoring system based on snmp. In: Proc. of 2009 Int. Joint Conf. on Artificial Intelligence (JCAI '09), pp. 680–682 (2009)
86. Zhao, Y., Yin, X., Wu, J.: Problems in the information dissemination of the internet routing. Journal of Computer Science and Technology **18**(2), 139–152 (2003)