Automated Robustness Testing for Reactive Systems: Application to Communicating Protocols

Fares Saad Khorchef Ismail Berrada Antoine Rollet
Richard Castanet

LaBRI - CNRS - UMR 5800
33405 Talence cedex, France
{saad-kho, berrada, rollet, castanet}@labri.fr

Abstract: In the telecommunications field, protocols have to be seriously validated before their startup. Thus, it is necessary to test the conformance of a protocol, but it is also important to test its robustness in presence of unexpected events. This paper proposes a framework to test the robustness of a system. Firstly, we explain how to increase the nominal specification in order to take into account the hazards. Then, we show how to generate test sequences from the increased specification. Finally, we propose a case study on the SSL protocol, using the TGSE tool.

1 Introduction

Protocol specifications are used to develop products and services. To ensure correctness of such products, testing is the one of the used validation techniques. It consists of checking that the behaviors of a real implementation of a system (IUT for Implementation Under Test) is correct with respect to a specification.

With the exponential growth of Internet and with the growth of other services, protocol testing has become more difficult. While testing to ensure that requirements are met is necessary (i.e. conformance testing), tests aimed at ensuring that the system handles errors and failures appropriately are often neglected (i.e. robustness testing).

Although a precise definition of robustness is somewhat elusive, functionally the meaning is clear : the ability of a system to function in an acceptable way in presence of faults or stressful environmental conditions [CW03]. The term "hazards" will be used to gather faults and stressful conditions.

The aim of this paper is to provide a formal framework for robustness testing for Internet protocols. In order to decide the robustness of an IUT, a clear criterion is needed, taking into account the system behaviors in the presence of hazards. The contributions of this paper are :

(1) A framework for robustness testing including a formal definition of robustness and a test generation method. Our approach consists in enriching the *nominal specification* (i.e. protocol standard specification) with some representable hazards in order to get an

increased specification.

(2) A case study on the SSL protocol [Hic95]. We show how to integrate hazards in the specification of the handshake protocol in order to generate robustness test cases using the TGSE tool [BF05].

The remainder of the paper is organized as follows : Section 2 introduces models and notations used in the paper. Section 3 gives a definition and a classification of hazards. Section 4 presents a formalization and a method to test the robustness. The case study is given in Section 5. The related work is presented in Section 6. Section 7 concludes and draws some perspectives.

2 Basic Concepts

A reactive system is a software component which reacts to stimuli of its environment. I/O labelled transition systems ($IOLTS$) are used to describe the behaviors of such systems. This section introduces the $IOLTS$ model and some notations used throughout this paper.

2.1 Input Output Labelled Transition System

Definition 2.1 *An $IOLTS$ [TRE96] is an quadruplet $S = (Q, A, \rightarrow_S, q_0)$ such that: Q is a nonempty finite set of states, q_0 is the initial state, A is the alphabet of actions and, $\rightarrow_S \subseteq Q \times A \times Q$ is the transition relation.*

The alphabet A is partitioned into three sets $A = A_O \cup A_I \cup I$, where A_O is the output alphabet (an output is denoted by $!a$), A_I is the input alphabet (an input is denoted by $?a$) and I is the alphabet of internal actions (an internal action is denoted by τ). Usual notations are:

Notation	Meaning	Notation	Meaning
$q \stackrel{\varepsilon}{\Rightarrow} q'$	$q = q'$ or $q \stackrel{\tau_1 \cdots \tau_n}{\rightarrow} q'$	$Trace(q)$	$\{\sigma \in A^* \mid q \stackrel{\sigma}{\Rightarrow}\}$
$q \stackrel{a}{\Rightarrow} q'$	$\exists\, q_1, q_2 \mid q \stackrel{\varepsilon}{\Rightarrow} q_1 \stackrel{a}{\rightarrow} q_2 \stackrel{\varepsilon}{\Rightarrow} q'$	$Trace(S)$	$Trace(q_0)$
$q \stackrel{a_1 \cdots a_n}{\Rightarrow} q'$	$\exists\, q_0 ... q_n \mid q = q_0 \stackrel{a_1}{\Rightarrow} q_1 \stackrel{a_2}{\Rightarrow} ... \stackrel{a_n}{\Rightarrow} q_n = q'$	$Out(q)$	$\{a \in A_O \mid q \stackrel{a}{\Rightarrow}\}$
$q\ after\ \sigma$	$q' \in Q \mid q \stackrel{\sigma}{\Rightarrow} q'\}$	$Out(S, \sigma)$	$Out(S\ after\ \sigma)$
$S\ after\ \sigma$	$q_0\ after\ \sigma$	$ref(q)$	$\{a \in A_I \mid a \stackrel{q}{\not\rightarrow})\}$

The observable behaviors is described by $\Rightarrow$. $q\ after\ \sigma$ is the set of reachable states from q by σ. $Trace(q)$ is the set of observable sequences starting from q. $Out(q)$ is the set of all possible outputs of q. Finally, $ref(q)$ is the set of inputs which can not start a transition from q.

The $IOLTS$ S is called *deterministic* if no state accepts more than one successor with an observable action. It is called *observable* if no transition is labeled by τ. S is called *input-complete* if each state accepts all inputs of the alphabet.

2.2 Suspension Graph

In practice, the tester observes events from a system, but also the absence of events (quiescence). Several kinds of quiescence may happen in a state $q \in Q$: *outputlock* quiescence if the system is blocked on standby input of the environment ($Out(q) = \emptyset$), *deadlock* quiescence if there is no more evolution of the system ($\forall a \in A | q \stackrel{a}{\not\rightarrow}$) or *livelock* quiescence if $q \stackrel{\varepsilon}{\Rightarrow} q$.

Definition 2.2 *The suspension graph [JER03] of $S = (Q, A, \rightarrow, q_0)$ is an IOLTS $S^\delta = (Q, A^\delta, \rightarrow_\delta, q_0)$ such that: $A^\delta = A \cup \{\delta\}$ with $\delta \in A_O^\delta$ (δ is considered as an output). $\rightarrow_\delta$ is obtained from $\rightarrow$ by adding loops $q \stackrel{\delta}{\rightarrow} q$ for all quiescence states*

2.3 Meta-Graph

In order to model the behaviors of a system $S = (Q, A, \rightarrow, q_0)$ in the presence of hazards, we use the concept of the *meta-graph* associated to S. A meta-graph G is a graph such that each state of G corresponds to a set of states of S having the same behaviors in the presence of the same hazards.

Definition 2.3 *A meta-graph associated to S is a triplet $G = (V, E, L)$ such as:*

- *$V = V_d \cup V_m$ is a set of states. $V_m \subseteq 2^Q$ is called the set of meta-states and V_d is called the set of degraded states such that $V_d \cap Q = \emptyset$.*
- *L is an alphabet of actions,*
- *$E \subseteq V \times L \times V$ is a set of edges*

Definition 2.4 (Composition $IOLTS \oplus G$)
Let $S = (Q, q_0, A, \rightarrow_S)$ be an $IOLTS$ and $G = (V, E, L)$ a meta-graph associated to S. The composition of S and G, noted $S \oplus G$, is the $IOLTS$ $(Q^{S\oplus G}, q_0^{S\oplus G}, A^{S\oplus G}, \rightarrow_{S\oplus G})$ defined by: $Q^{S\oplus G} = Q \cup V_d$, $q_0^{S\oplus G} = q_0$, $A^{S\oplus G} = A \cup L$ and,

1. $q \stackrel{a}{\rightarrow} q' \quad \Rightarrow \quad q \stackrel{a}{\rightarrow}_{S\oplus G} q'$
2. $(v, a, v') \in E$ *and* $v, v' \in V_d \quad \Rightarrow \quad v \stackrel{a}{\rightarrow}_{S\oplus G} v'$.
3. $(v, a, v') \in E$, $v \in V_m$ *and* $v' \in V_d \quad \Rightarrow \quad q \stackrel{a}{\rightarrow}_{S\oplus G} v'$ *for all* $q \in v$.
4. $(v, a, v') \in E$, $v \in V_d$ *and* $v' \in V_m \quad \Rightarrow \quad v \stackrel{a}{\rightarrow}_{S\oplus G} q$ *for all* $q \in v'$.
5. $(v, a, v') \in E$ *and* $v, v' \in V_m \Rightarrow q \stackrel{a}{\rightarrow}_{S\oplus G} q'$ *for all* $q \in v$ *and* $q' \in v'$

This composition consists in adding in S the set of transitions and states of meta-graph G. Actually, for a state q of S member of a meta-state (i.e. a set of states) v of G, we add in S the set of transitions and/or states starting from v. In the following, this composition is used to integrate hazards modeled as meta-graph(s) in the nominal specification. Figure 3 illustrates this composition.

3 Hazards

In robustness testing, a *hazard* denotes any event not expected in the nominal specification of the system. In this section, we propose to extend the hazards classification given in [CW03]. Our classification considers :

Position of hazards relative to the system boundaries. We distinguish the internal (e.g. memory overflow, processor failure, etc...), external (e.g. intrusion, stressful conditions, etc...) and beyond the system boundaries hazards.

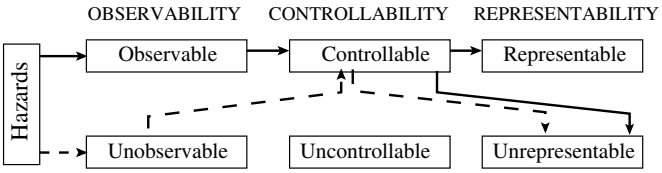

Figure 1: Classification of hazards

Position of hazards relative to the tester controllability. We regroup the hazards basing on their observability, representability and controllability by the tester. Figure 1 gives the different possible combinations :

1. Observable, controllable and representable hazards. They regroup *invalid inputs* (e.g. modified inputs, erroneous inputs or lost inputs) or *inopportune inputs* (e.g. messages in advance or late). These terms are defined below.
2. Observable, controllable and unrepresentable hazards. They are composed by some *external failures* whose influence on the inputs are not very clear or difficult to represent (e.g. pressure, radiation, temperature).
3. Unobservable, controllable and unrepresentable hazards. They are composed by some *complex internal failures* which we can not describe with classic models (e.g, memory overflow, processor bugs).

The other possible combinations of figure 1 are not considered because we will not able to test something not controllable. For observable and representable events, we identify three kinds of hazards :

Invalid Inputs describing some erroneous, specified inputs (e.g. incorrect values, errors of initialization, temporization faults).

Inopportune Inputs corresponding to actions which exist in the alphabet of the specification, but not expected in the given state. $ref(q)$ (see standard notations of $IOLTS$) denotes the inopportune inputs in a state $q \in Q$.

Unexpected outputs. Taking into account the hazards can lead the system, in some cases, to send some unexpected outputs. Sometimes, such outputs may be considered as acceptable. For example, restarting a session, resetting or closing a connection may be acceptable behaviors. As a consequence, all acceptable outputs must be added to the specification.

4 Proposed framework

In this part, we propose a formal approach to generate robustness test cases. Firstly, we show how to integrate the representable hazards in the initial model. The obtained model is called *increased specification*. Secondly, we formalize the robustness of an implementation compared to the increased specification. Basing on the previous relation, we explain how to produce robustness test cases using a test purpose. The approach diagram is given in figure 2.

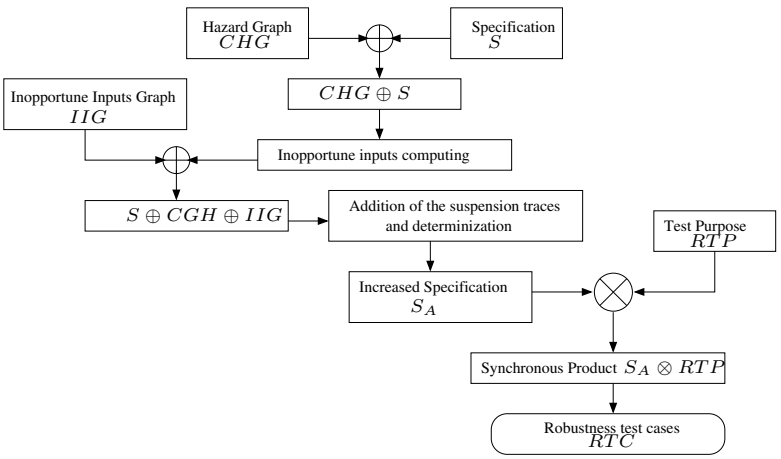

Figure 2: Approach diagram

4.1 Increase of the specification

The aim of the increased specification is to formally describe the acceptable behaviors in presence of controllable and representable hazards. In our approach, we describe the behavior of the system in presence of one or several hazard(s) modeled by meta-graph(s). Figures 3 (a) and (b) illustrate this concept. Assuming that S (figure 3 (a)) is in its initial state, if it receives the hazard $?a'$, it has to move to the degraded state $d2$ according to the meta-graph of figure 3 (b). Besides, if it sends the acceptable output $!x'$, it has to move to the degraded state $d1$, which permits the system to come back to a nominal behavior (here the initial state) in case of reception of $?a$.

In the following, we suppose that hazards (invalid inputs and acceptable outputs) are modelled by one or more meta-graph(s) CHG (*Controllable Hazard's Graph*). Then, inopportune inputs can be automatically computed and are represented by meta-graph(s) IIG (*Inopportune Input Graph*). The increased specification (see figure 3 (f)) is obtained as follows. For a nominal specification S and a set of hazards modeled by a meta-graph CHG, we firstly compose them to obtain $CGH \oplus S$ (see figure 3 (c)). Secondly, we compute the inopportune inputs of $CGH \oplus S$ to construct IIG (figure 3 (d)). Then, we build $CGH \oplus S \oplus IIG$ (figure 3 (e)). Finally, we add suspension traces and we proceed to determinization of $CGH \oplus S \oplus IIG$ (figure 3 (f)) in order to obtain the increased specification.

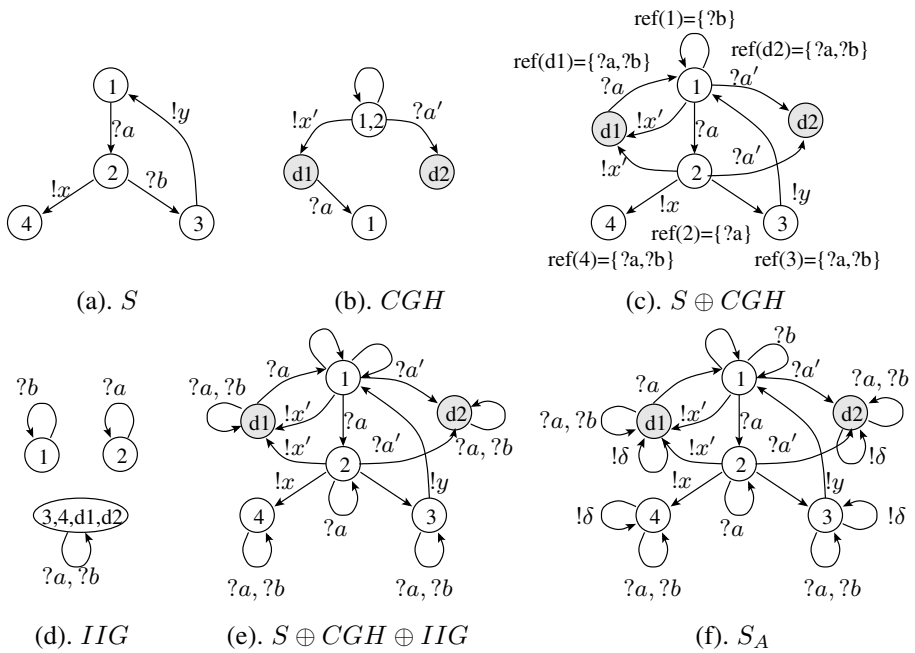

Figure 3: Increase of specification

4.2 Robustness relation

In order to describe formally the notion of robustness, the following hypothesis are needed :

Increased specification. We suppose that the nominal specification is modeled by an IOLTS $S = (Q, A, \rightarrow, q_0)$. The increased specification of S is modeled by a deterministic, observable and input-complete IOLTS $S_A = (Q^{S_A}, A^{S_A}, \rightarrow_{S_A}, q_0^{S_A})$ (the construction of S_A is the one given in the previous sub-section).

Implementation. The real implementation under test (IUT) is unknown, but to be able to reason formally on the robustness of an implementation I with respect to a specification S, we assume that :

1. I is modelled by an IOLTS,
2. I conforms to S,
3. I is input-complete on the alphabet A^{S_A}.

Robustness relation. Let I be an implementation of a specification S and S_A its increased specification. The robustness relation **Robust** is defined by :

$$I \textbf{ Robust } S_A \equiv_{def} \forall \sigma \in Trace(S_A) \backslash Trace(S^\delta) \Rightarrow Out(I^\delta, \sigma) \subseteq Out(S_A, \sigma).$$

Only the increased behaviors (added) are useful for robustness testing because the nominal behaviors (including valid quiescence) already passed the conformance testing.

Example 1 *Let us consider figure 4.*

- I_1 ***Robust*** S_A *because all traces in* I_1 *are included in* S_A
- *not(*I_2 ***Robust*** S_A*) because* I_2 *after* $?a'$ *sends* $!y$ *but* S_A *after* $?a'$ *sends* $!x'$.
- *not(*I_3 ***Robust*** S_A*):* I_3 *after* $?a'$ *reaches another state not specified in* S_A.
- *not(*I_4 ***Robust*** S_A*):* I_4 *after* $?a'$ *sends* $!y$ *and reaches another state not specified in* S_A.

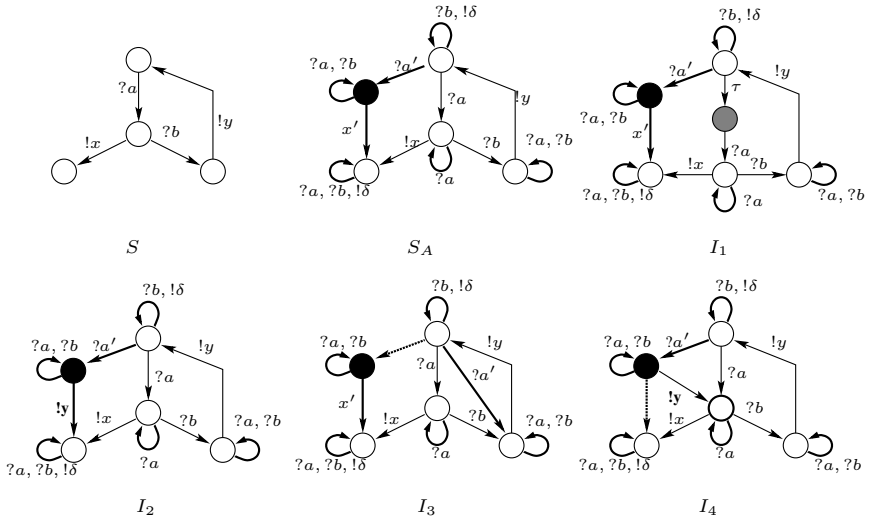

Figure 4: Robustness relation

4.3 Test generation

After the integration of hazards (invalid inputs, inopportune inputs and acceptable outputs), the size of the increased specification becomes very large. In order to reduce test costs, we propose to select the tests by using *test purposes*. This method was used in some conformance testing works [BF05, JER03]. The principle of this method is founded on the synchronization between the specification and the test purpose modelled by two $IOLTS$.

The robustness test generation may be summarized as follows. 1) Choice of robustness test purpose. 2) Synchronization between the specification and the test purpose in order to deduce the behaviors which satisfied the test purpose. 3) Mirror image (i.e. inputs become outputs and outputs become inputs) of the *synchronous product*. 4) Extraction of *robustness test cases*.
The *Robustness Test Purpose (RTP)* is used to check some robustness properties in the IUT. In the proposed case study, we use TGSE tool [BF05] in order to generate test cases.

5 Case study: The SSL Protocol

Netscape describes SSL as follows [Hic95] : "The SSL protocol is designed to provide privacy between two communicating applications (a client and a server). Second, the protocol is designed to authenticate the server, and optionally the client". SSL is standardized by the IETF (Internet Engineering Task Force). The full specification of the SSL proto-

col is written in the RFC 2246. There exists several implementations of the SSL protocol (Open SSL, SSLeay, BSAFF 3.0, SSL Plus, SSL Ref 3.0).

The SSL Protocol contains four under-protocols: *Handshake protocol*, *SSL Changes Cipher Spec protocol*, *SSL Alert protocol* and *SSL Record protocol*. The Handshake protocol is composed of two phases. First step deals with the selection of a cipher, the exchange of a master key and the authentication of the server. Second step handles client authentication, if requested and finishes the handshaking. After the handshake stage is complete, the data transfer between client and server begins. All messages during handshaking and after, are sent over the SSL Record protocol Layer.

Here, we deal only with the specification of the handshake protocol which describes three scenarios of communication as shown in the following table :

Case	**Sequences**
No Session Identifier, No Client authentication	!Client-Hello, ?Server-Hello, ?Client-Master-Key, ?Client-Finished, !Server-Verify, !Server-Finished
Session Identifier Used, No Client authentication	!Client-Hello, ?Server-Hello, ?Client-Finished, !Server-Verify, !Server-Finished
Session Identifier Used, Client authentication	!Client-Hello, ?Server-Hello, ?Client-Master-Key, ?Client-Finished, !Server-Verify, ?Request-Certificate, !Client-Certificate !Server-Finished

Table 1: Nominal scenarios of the Handshake protocol

The standard specification (RFC2246) defines the following errors :

- NO-CIPHER-ERROR. This error is returned by the client to the server when it can not find a cipher or key size. This error is not recoverable.
- NO-CERTIFICATE-ERROR. When a REQUEST-CERTIFICATE message is sent, this error may be returned if the client has no certificate to reply with. This error is recoverable (for client authentication only).
- BAD-CERTIFICATE-ERROR. This error is returned when a certificate is deemed bad by the receiving party. Bad means that either the signature of the certificate was bad or that the values in the certificate were inappropriate (e.g. a name in the certificate did not match the expected name). This error is recoverable (for client authentication only).
- UNSUPPORTED-CERTIFICATE-TYPE-ERROR. This error is returned when a client/server receives a certificate type that it can not support. This error is recoverable (for client authentication only).

Two error messages have been omitted from the specification document (this problem is noticed in [BD95]). The first, an UNSUPPORTED AUTHENTICATION TYPE ERROR message, is a mistake which would prevent the protocol using different methods of authentication of a client. The second, an UNEXPECTED-MESSAGE-ERROR would allow an implementation to close the connection cleanly if an implementation sent an out-off-order message.

In order to verify the robustness of the Handshake protocol, we increase the nominal specification by integrating hazards (*invalid inputs* and *inopportune inputs*). Besides, to model the previous hazards, we consider the following hypothesis. 1) if the implementation receives an invalid input then it closes the connection and, 2) if it receives an inopportune input then it loops in the same state. Formally, the previous hypothesis may be modelled by meta-graphs. The increased specification is given in figure 5 (Annexe), it is composed of 20 states and 176 transitions. In figure 5, the inopportune inputs are represented by $ref(q)$ for any state q but in the experimentations, they are automatically computed.

5.1 Robustness test generation with TGSE tool

In order to generate robustness test cases, we have defined a set of robustness test purposes aiming at checking the behavior of an implementation in presence of the certificate failures (No-Certificate-Error, Bad-Certificate-Error, Unsupported-Certificate-Type-Error), the cipher failures (No-Cipher-Error) and two other failures (Unexpected-Message-Error et Unsupported-Authentification-Type-Error) :

- **RTP1**: Closing the connection between the client and the server after detection of a certificate failure.
- **RTP2**: Closing the connection after detection of a cipher failure.
- **RTP3** : Closing the connection after detection of unexpected error message.

In addition, we mention that both the inopportune inputs and quiescence are automatically generated by the TGSE tool[BF05]. The following table summarizes the different results obtained by a TGSE implementation under Linux Fedora 3 station (Intel Pentium 4 CPU 1.80GHz, 128Mo of memory). "RTC size" represents the average size of the robustness test cases and "CPU Time" represents the average time needed to extract the RTC. We notice that Robustness Test Cases are significantly longer than in usual conformance testing methods. The reason is that the integration of hazards in the specification increases the number of possible transitions.

Robustness test purposes	RTC size	CPU Time(ms)
RTP1	53	0.9618
RTP2	10	0.3599
RTP3	20	0.15797

Table 2: Results obtained by TGSE tool

6 Related work

Many research have been done in the domain of protocol testing . The majority of these works deals with conformance testing [IEE04] (an overview may be found in [JER03]). In this section, we focus particularly on robustness testing works.

[CW03] proposes a study on robustness testing, focusing on hazard classification and some possible directions to handle the problem. Authors define the robustness notion as "the ability of a system to function acceptably in the presence of faults or stressful environmental conditions" and provide a state of the contributions in this domain.

The PROTOS project [RLT02] proposes to describe the system with a high level of abstraction and then to simulate abnormal inputs in the specification. It is mainly focused on the detection of vulnerabilities of a network software system. In this case, robustness is restricted to the notion of network security.

Some approaches are based on software fault injection :
The FIAT tool [BCSS90] modifies a processus binary image in memory, [Reg05] applies randomly interruptions in the IUT, whereas the BALLISTA tool works on data unexpected modifications. These approaches are based on integration of faults directly in the software implementation of the system, but do not care about interpretation of different behaviours.

Another approach consists in using model-based test generation. The main difficulty of such technics is to describe the hazards in the model. Many works consider such approach : [SKD05, FMP05, Rol03].

[SKD05] proposes a first approach based on a refusal graph used to model hazards. Contrary to our method, it only deals with inopportune inputs, but not with invalid inputs. Moreover, our approach distinguishes between inputs and outputs in the model.

[FMP05] uses a formal fault model in order to build a "mutant" specification. They use a fault model in order to add "fault" transitions in the specification. They define a robustness relation based on a robustness property. Contrary to our approach, they do not permit to integrate unexpected inputs in the model.

[Rol03] uses a degraded specification to model the behavior in case of critical situation, and integrates the hazards directly in the test sequences. A major difference between [Rol03] and this work is in the concept of robustness : we consider here that robustness implies conformance; [Rol03] does not.

7 Conclusion

In this paper, we have presented a formal framework and a generation technic to test the robustness of a protocol modeled as IOLTS. We proposed to integrate representable hazards in the specification after suspension traces addition and determinization. Then we used this increased specification in order to generate robustness test cases using a test purpose. Secondly, we proposed a case study on the SSL protocol by describing how to increase

the specification, and by generating test sequences with the TGSE tool. This case study permits to show the complementary aspect of conformance and robustness testing.

Currently, we are working on tools to help the designer of a system to describe the behavior of the system in case of unexpected events. Besides, we are studying a way to handle time constraints in robustness testing.

References

[BCSS90] J. H. Barton, E. W. Czeck, Z. Z. Segall, and D. P. Siewiorek. Fault Injection Experiments Using FIAT. *IEEE Trans. Comput.*, 39(4):575–582, 1990.

[BD95] J. Bradley and N. Davies. Analysis of the SSL Protocol. Technical Report CSTR-95-021, Department of Computer Science, University of Bristol, June 1995.

[BF05] I. BERRADA and P. FELIX. TGSE: Un outil générique pour le test. In *CFIP'2005: Ingénérie des Protocoles*, pages 67–84, 29 Mars 2005.

[CW03] R. CASTANET and H. WAESELYNK. Techniques avancées de test de systèmes complexes: Test de robustesse. Technical report, Action spécifique 23 du CNRS, 11 2003.

[FMP05] J-C. FERNANDEZ, L. MOUNIER, and C. PACHON. A Model-Based Approach for Robustness Testing. In LNCS, editor, *Testing of Communication Systems*, volume 3502, pages 333–348. ifip, may/june 2005.

[Hic95] Kipp Hickman. The SSL Protocol. Technical report, Netscape Communications Corp., Feb 9 1995.

[IEE04] IEEE. *International Organization for Standardization, Conformance testing methodology and framework - part 2: abstract test suite specification*, 2004.

[JER03] T. JERON. Génération de tests pour les systèmes réactifs. Un survol des théories et techniques. In IRIT, editor, *ETR2003. Systèmes, Réseaux et Applications*, pages 105–122. IRIT, Septembre 2003.

[Reg05] John Regehr. Random testing of interrupt-driven software. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 290–298, New York, NY, USA, 2005. ACM Press.

[RLT02] J. Röning, M. Laakso, and A. Takanen. PROTOS - systematic approach to eliminate software vulnerabilities. http://www.ee.oulu.fi/research/ouspg, May 2002. 2002.

[Rol03] A. Rollet. Testing robustness of real-time embedded systems. *In Proceedings of Workshop On Testing Real-Time and Embedded Systems (WTRTES), Satellite Workshop of Formal Methods (FM) 2003 Symposium, Pisa, Italy*, September 13 2003.

[SKD05] F. SAAD-KHORCHEF and X. DELORD. Une méthode pour le test de robustesse adaptée aux protocoles de communication. 29 mars 2005.

[TRE96] J. TRETMANS. Conformance Testing with Labelled Transition Systems: Implementation Relations and Test Generation. *Computer Networks and ISDN Systems*, 29:49–79, 1996.

Annexe

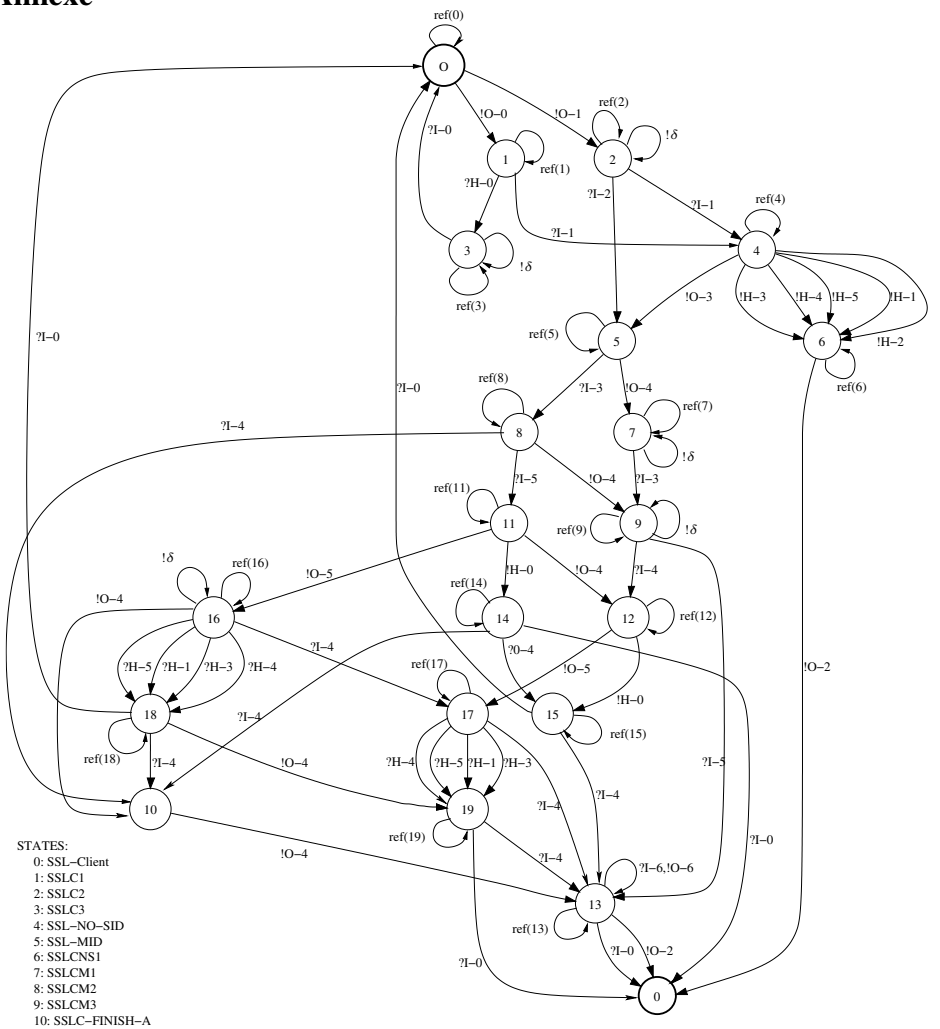

Figure 5: Increased specification of the Handshake protocol