2009 Seventh IEEE International Conference on Software Engineering and Formal Methods

Applying formal passive testing to study temporal properties of the Stream Control Transmission Protocol *

César Andrés, Mercedes G. Merayo, Manuel Núñez
Departamento Sistemas Informáticos y Computación
Universidad Complutense de Madrid
E-28040 Madrid. Spain.
{c.andres,mgmerayo}@fdi.ucm.es,mn@sip.ucm.es

Abstract

In this paper we present a formal passive testing framework and use it to analyze time aspects in the Stream Control Transmission Protocol (SCTP). This protocol presents different phases where time aspects are critical. In order to represent temporal requirements we use so-called timed invariants since they allow us to easily verify that the traces collected from the observation of the protocol fulfill the corresponding timed constraints. In addition to introduce our theoretical framework, we report on the results obtained from the application of our techniques over (possibly mutated) traces extracted from runs of the SCTP.

1 Introduction

With the growing significance and complexity of software systems, techniques that assist in the production of reliable software are becoming more important. Two of the most promising approaches within software engineering to increase the confidence on the developed software are *formal methods* and *testing*. Traditionally, formal methods and testing have been seen as rivals. Therefore, there was very little interaction between the two communities. In recent years, however, these approaches are seen as complementary [BU91, LY96, Tre96, Pet01, BT01, RMN08, HBH08, HBB+09].

Testing is usually based on the ability of a tester that stimulates the Implementation Under Test (IUT) and checks the correction of the answers provided by it. However, in some situations this activity becomes difficult and even impossible to perform. For example, this is the case if the tester is not provided with a direct interface to the IUT. Another conflictive situation appears when the IUT cannot be shutdown or interrupted for a long period of time. In these situations, there is a particular interest in using other types of validation techniques such as *passive testing*. The main difference between active and passive testing is that in active testing testers can interact, by providing inputs, with the IUT and observe the obtained result. In passive testing, testers cannot interact directly with the IUT. The usual approach of formal passive testing consists in observing the IUT and trying to find *a fault* by *comparing* the observed events and the specification [LNS+97, Mil98, TC99, WZY01, LCH+02, UX07, BDS+07]. It is worth to mention that a similar objective, but using completely different techniques, is pursued by research on *runtime verification* [LS09].

A new methodology to perform passive testing was presented in [CGP03]. The main novelty is that a set of *invariants* is used to represent the most relevant expected properties of the specification. Intuitively, an invariant expresses the fact that each time the IUT performs a given sequence of actions, called *preface*, then it must exhibit a behavior reflected in the last part of the invariant. This approach presented the drawback that the grammar to express invariants was very limited. For example, it did not allow to represent wild-card characters. This first limitation was overcome with a new proposal having a richer class of invariants [BCNZ05]. But there was still a limitation shared by all previous proposals for (formal) passive testing: They cannot deal with temporal information. Even though research on active testing of timed systems is already well established (see, for example, [MMM95, HNTC99, SVD01, EDK02, HW05, MNR08a, MNR08b]), our work [AMN08, AMN09] together with the PASTE tool (see [AMM09] for a description of its main features) represents, as far as we know, the first complete framework to perform passive testing of timed systems. As most of the previously mentioned work on passive testing, our approach

*Research supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01) and by the UCM-BSCH programme to fund research groups (GR58/08 - group number 910606).

978-0-7695-3870-9/09 $26.00 © 2009 IEEE
DOI 10.1109/SEFM.2009.21

is based on the *finite state machine* formalism, more precisely, our machines allow to represent the time that elapses between applying an input and receiving an output.

In order to assess the suitability of our framework we decided to use it in a non-trivial experiment. We chose a real protocol, the Stream Control Transmission Protocol (SCTP), and decided to perform experiments with it. The SCTP is a network protocol with three different phases: *Handshake*, *information sending*, and *disconnection*. This protocol can be seen as a natural extension of TCP, but there are some relevant new features that SCTP supports, such as *multi-homing* and *multi-streaming*. Our first problem to model the protocol was that it was highly unnatural to maintain a strict alternation between inputs and outputs since, very often, we had to use *empty* inputs or outputs to simulate the situation when two or more inputs (resp. outputs) are consecutively generated. Thus, we considered the *translation* of our formal framework to the most extended formalism to model timed systems: Timed Automata [AD94]. Let us remark that by no means we are claiming that (timed) finite state machines should be discarded. In particular, our group is a strong advocate of (timed, probabilistic, stochastic) variants of finite state machines (e.g. [LNR06, MNR08b, MNR08a, HMN09]). In fact, we think that this paper shows that research on (timed) finite state machines can be easily adapted to deal with (timed) automata.

In this paper we report on the application of the new framework, that is, the adaption of [AMN08, AMN09] to deal with timed automata, to analyze the SCTP. Due to space limitations, we will not give the technicalities of the *new* framework, that is, the algorithms to decide the correctness of an invariant with respect to a specification and to match an invariant and a log extracted from the IUT. These algorithms are, more or less cumbersome, adaptions of the ones appearing in [AMN08, AMN09].

The first step was to model the protocol as a collection of timed automata. We used the *informal* description available at [Int07] and the RFC 2960 [SXM+00]. Once we had each of the relevant entities of the SCTP modeled as a timed automaton, we had to ensure, better say, convince ourselves, that we had accurately modeled the protocol. As usual, there is no automatic way of proving that a formal specification correctly reflects the (informal) requirements of the system. This is a detail that it is very often overlooked in case studies: The main problem might be that either the specification or the formalization of informal requirements is wrong. Therefore, we decided to start with a sanity-check, *self-feedback* process: Instead of using passive testing on an implementation to detect errors, we checked *correct* traces with respect to our specification. In this case, if we find an *error*, then we know that we were a bit careless since this error must be caused by a wrong specification. Following the procedure described in [CGP03], we automatically derived all the invariants with length up to five (including wild-card characters). We checked these invariants against real traces of the SCTP protocol.[1] We did not find a mismatch between traces and invariants. Let us remark again that this fact does not imply that our specification is a correct representation of the informal requirements: We are simply confident that we modeled the correct system. This *uncertainty* is common to any research that takes as basis a specification *manually* generated from informal requirements. Other possibilities are to assume that the specification is correct *by definition* or that requirements are expressed in a certain formal language (e.g. as temporal logic formulae) and that we *model check* the correctness of the specification with respect to these requirements. In our case, we reached a point where we had a *correct* specification and we wanted to put into practice our timed passive testing approach.

The next problem that we confronted was that we could not find errors in the recorded traces since they were extracted from correct implementations of the protocol. Thus, in order to evaluate our proposal, we considered a *relevant* set of invariants and matched them against *mutated* traces. Mutation testing techniques (e.g. [How82, OL94, OPTZ99]) can be used to choose those tests that are more likely to find errors in a faulty IUT. Intuitively, the idea consists in taking the original, correct program/specification and introduce some errors to generate a set of *mutants*. If a test finds most of the added errors, then the test is very likely to find errors in *real* implementations. In our case, we could have introduced errors in the original code of the SCTP since we can access the kernel of our Devian Operating System. However, this was too complicated and we preferred an alternative approach with the same final result: We extracted correct traces and (automatically) introduced *mutations* in the traces, instead of introducing them in the protocol. This paper reports on these experiments.

The structure of the rest of the paper is as follows. In Section 2 we present the basis behind the timed automata formalism. In Section 3 we give the formal representation, using timed automata, of the SCTP protocol. Next, in Section 4 we introduce our notion of timed invariants. The results of our experiments are presented in Section 5. Finally, in Section 6 we present our conclusions and future lines of research.

[1] We obtained traces from three different sources. We used the official download web site samples of the *Wireshark* sniffer at http://wiki.wireshark.org/SampleCaptures and at http://www.techtraces.com/sample_captures, where representative sets of logs of the SCTP protocol can be found. In order to increase the number of traces, we installed the SCTP in one of our systems and recorded traces of varying length.

2 A review of timed automata

In this paper we use *timed automata* with a finite set of clocks over a dense time domain to represent specifications of systems. Since we will not use most of the technical machinery behind timed automata, the reader is referred to [AD94] for further details. The clock domain is defined in $\mathbf{R}_+$. The choice of a next state in the automaton does not only depend on the action, but also on the timed constraints associated to each transition. Only when the time condition is satisfied by the current values of the clocks, the transition can be triggered.

Definition 1 A *clock* is a variable c in $\mathbf{R}_+$. A set of clocks will be denoted by $\mathcal{C}$. A *timed constraint* φ on $\mathcal{C}$ is defined by the following EBNF:

$$\varphi ::= \varphi \wedge \varphi \mid c \leq t \mid c < t \mid \neg\varphi$$

where $c \in \mathcal{C}$ and $t \in \mathbf{R}_+$. The set of all timed constraints over a set $\mathcal{C}$ of clocks is denoted by $\phi(\mathcal{C})$.

A *clock valuation* ν for a set $\mathcal{C}$ of clocks assigns a real value to each of them. For $t \in \mathbf{R}_+$, the expression $\nu + t$ denotes the clock valuation which maps every clock $c \in \mathcal{C}$ to the value $\nu(c)+t$. For a set of clocks $\mathcal{Y} \subseteq \mathcal{C}$, the expression $\nu[\mathcal{Y} := 0]$ denotes the clock valuation for $\mathcal{C}$ which assigns 0 to each $c \in \mathcal{Y}$ and agrees with ν over the rest of the clocks. The set of all clock valuations is denoted by $\Omega(\mathcal{C})$.

Let ν be a clock valuation and φ be a timed constraint. We write $\varphi \vdash \nu$ iff ν holds φ; $\varphi \nvdash \nu$ denotes that ν does not hold φ.

A *timed automaton* is a tuple $\mathcal{A} = (\mathcal{S}, s_0, \Sigma, \mathcal{C}, \mathcal{Z}, \mathcal{E})$ where $\mathcal{S}$ is a finite set of locations, $s_0 \in \mathcal{S}$ is the initial location, Σ is the alphabet of actions, $\mathcal{C}$ is a finite set of clocks, $\mathcal{Z} : \mathcal{S} \to \phi(\mathcal{C})$ associates a time condition to each location, and $\mathcal{E} \subseteq \mathcal{S} \times \Sigma_\tau \times \phi(\mathcal{C}) \times \wp(\mathcal{C}) \times \mathcal{S}$ is the set of transitions where $\Sigma_\tau = \Sigma \cup \{\tau\}$, being $\tau \notin \Sigma$ a special symbol to represent internal, non-observable activity. We will consider that Σ is partitioned into two (disjoint) sets of *inputs*, preceded by `?`, and *outputs*, preceded by `!`.

We overload the $\vdash$ symbol. Let $s \in \mathcal{S}$ and $\nu \in \Omega(\mathcal{C})$. We denote by $s \vdash \nu$ the fact that ν holds $\mathcal{Z}(s)$ (resp. $s \nvdash \nu$ represents that ν does not hold $\mathcal{Z}(s)$). Let $e = (s, \alpha, \varphi, \mathcal{Y}, s') \in \mathcal{E}$. We denote by $e \vdash \nu$ the fact that ν holds φ (resp. $e \nvdash \nu$ represents that ν does not hold φ). □

Intuitively, a transition $(s, \alpha, \varphi, \mathcal{Y}, s')$ indicates that if the system is at state s and φ holds for the current valuation of the clocks, then the system moves to the state s' performing the action α and resetting the clocks in $\mathcal{Y}$. For each state s, $\mathcal{Z}(s)$ represents a timed constraint for s, that is, the system can remain in s while $\mathcal{Z}(s)$ holds for the current valuation of the clocks. We will assume the following usual condition on timed automata: For all $s \in \mathcal{S}$ and all configuration $\nu \in \Omega(\mathcal{C})$ if $s \nvdash \nu$ then there exists at least a transition $e = (s, \alpha, \varphi, \mathcal{Y}, s') \in \mathcal{E}$ with $e \vdash \nu$. This property allows to leave a state once the restrictions on clocks do not hold in that state.

As usual, the semantics of a timed automaton is given by translating it into a *labeled transition system* with an uncountably number of states. Let us remark that, in general, we will not construct the associated labeled transition system; we will use it to reason about the traces of the corresponding timed automaton.

Definition 2 A *labeled transition system*, or `LTS`, is a tuple $\mathcal{M} = (\mathcal{Q}, q_0, \Sigma, \rightarrow)$, where $\mathcal{Q}$ is a set of states, $q_0 \in \mathcal{Q}$ is the initial state, Σ is the alphabet of actions, and the relation $\rightarrow \subseteq \mathcal{Q} \times \Sigma_\tau \cup \mathbf{R}_+ \times \mathcal{Q}$ represents the set of transitions, being $\Sigma_\tau = \Sigma \cup \{\tau\}$. We will use the notation $q \xrightarrow{\alpha} q'$ to express $(q, \alpha, q') \in \rightarrow$.

The *semantics* of a timed automaton $\mathcal{A} = (\mathcal{S}, s_0, \Sigma, \mathcal{C}, \mathcal{Z}, \mathcal{E})$ is defined by its associated `LTS` $\mathcal{M}_\mathcal{A} = (\mathcal{Q}, q_0, \Sigma, \rightarrow)$, where $\mathcal{Q} = \{(s, \nu) \mid s \in \mathcal{S} \wedge \nu \in \Omega(\mathcal{C}) \wedge s \vdash \nu\}$, $q_0 = (s_0, \nu_0)$, being $\nu_0(c) = 0$ for all $c \in \mathcal{C}$, and we apply two rules in order to generate the elements of $\rightarrow$. For all $(s, \nu) \in \mathcal{Q}$ we have:

- If for all $0 \leq t' \leq t$ we have $s \vdash (\nu + t')$, then $((s, \nu), t, (s, \nu + t)) \in \rightarrow$.
- If $e \vdash \nu$, for $e = (s, \alpha, \varphi, \mathcal{Y}, s') \in \mathcal{E}$, then $((s, \nu), \alpha, (s', \nu[\mathcal{Y} := 0])) \in \rightarrow$.

In addition, we consider the following conditions: (a) If we have $q \xrightarrow{t} q'$ and $q' \xrightarrow{t'} q''$, then we also have $q \xrightarrow{t+t'} q''$ and (b) if $q \xrightarrow{0} q'$ then $q = q'$, that is, a passage of 0 time units does not change the state. □

Next, we introduce the notion of *visible trace*, or simply *trace*. As usual, a trace is a sequence of visible actions and time values.

Definition 3 Let $\mathcal{M} = (\mathcal{Q}, q_0, \Sigma, \rightarrow)$ be a `LTS` and $q, q' \in \mathcal{Q}$. If there exist $q_1, \ldots, q_{n-1} \in \mathcal{Q}$ such that $q \xrightarrow{\tau} q_1 \xrightarrow{\tau} q_2, \ldots, q_{n-1} \xrightarrow{\tau} q'$, then we write $q \overset{\epsilon}{\Longrightarrow} q'$. Let us note that if $n = 0$ then $q' = q$. Let $\alpha \in \Sigma \cup \mathbf{R}_+$. If $q_1 \xrightarrow{\alpha} q_2$ and $q \overset{\epsilon}{\Longrightarrow} q_1 \xrightarrow{\alpha} q_2 \overset{\epsilon}{\Longrightarrow} q'$, then we write $q \overset{\alpha}{\Longrightarrow} q'$.

We say that $(q, \langle \alpha_1, \ldots, \alpha_n \rangle, q')$, where $q, q' \in \mathcal{Q}$ and for all $1 \leq i \leq n : \alpha_i \in \Sigma \cup \mathbf{R}_+$, is a *visible trace* of $\mathcal{M}$ if there exist $q_1, \ldots, q_{n-1} \in \mathcal{Q}$ such that $q \overset{\alpha_1}{\Longrightarrow} q_1 \overset{\alpha_2}{\Longrightarrow} \ldots q_{n-1} \overset{\alpha_n}{\Longrightarrow} q'$. In this case we write $q \overset{\sigma}{\Longrightarrow} q'$; we write $q \overset{\sigma}{\Longrightarrow}$ if we are not interested in the reached state.

The set of *normalized visible traces* of $\mathcal{M}$, denoted by `Norm_Traces`$(\mathcal{M})$, is given by

$$\left\{ \sigma \in (\Sigma \cup \mathbf{R}_+)^* \;\middle|\; \begin{array}{c} \sigma = \langle \alpha_1, \beta_1, \ldots, \beta_{n-1}, \alpha_n \rangle \\ \wedge\ q_0 \overset{\sigma}{\Longrightarrow} \ \wedge \forall\, 1 \leq i \leq n : \alpha_i \in \Sigma \\ \wedge\ \forall\, 1 \leq i \leq n-1 : \beta_i \in \mathbf{R}_+ \end{array} \right\}$$

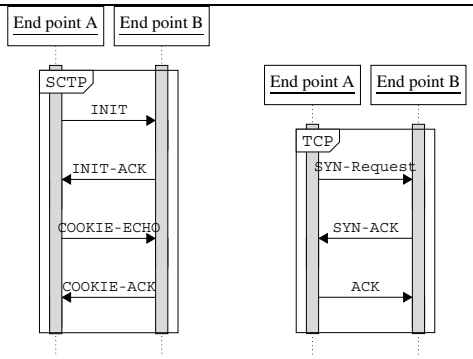


Figure 1. `SCTP` **and** `TCP` **handshake diagram.**

Finally, a *log* from a timed automaton $\mathcal{A}$ is a sequence of actions and delays that belongs to $\texttt{Norm\_Traces}(\mathcal{M}_{\mathcal{A}})$, being $\mathcal{M}_{\mathcal{A}}$ the LTS associated to $\mathcal{A}$. □

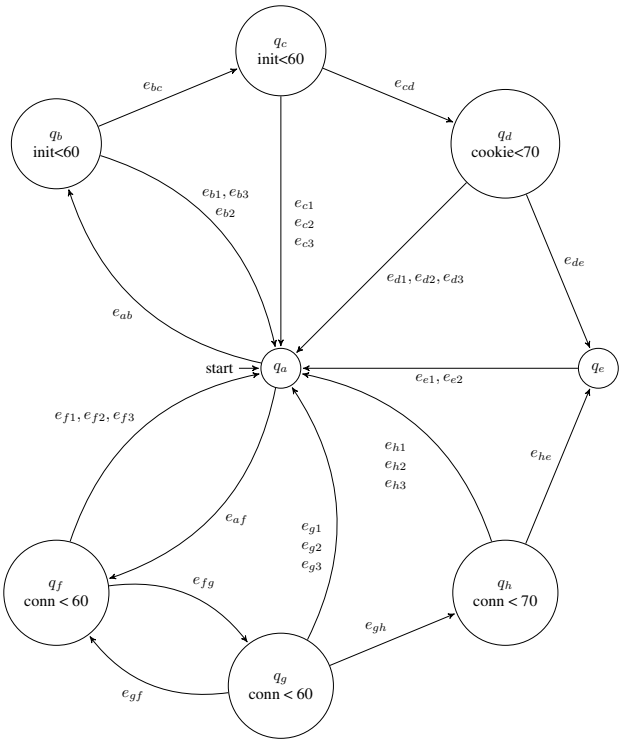
We will usually consider normalized visible traces since this is what we observe from the execution of a system. We cannot observe either internal activity (that is, the performance of internal actions) or different passages of time associated to different transitions. For example, if a system performs the sequence $3, 2, !a_1, 1, 1, \tau, 2, ?a_2, 2, 2, 4, \tau, ?a_3, 3, 4, \tau, 1$ then we will observe the normalized visible trace $\langle !a_1, 4, ?a_2, 8, ?a_3 \rangle$, that is, we remove the initial and final lapses of time, and consecutive time values, possibly interspersed with τ actions, are added (e.g. $1, 1, \tau, 2$ becomes 4).

3 Specification of the `SCTP`

Next we present the `SCTP` protocol. As we have already explained, we will use timed automata for modeling it. In this case study we consider two of the three stages of this protocol: The *handshake* and the *disconnection*.

Before we present the handshake automaton, we briefly describe the *security at startup* of the `SCTP` by comparing it with the one of `TCP`. Both protocols, `SCTP` and `TCP`, carry out an exchange of messages to establish an end-to-end relationship. However, the way these messages are sent is different: `TCP` uses a three-way handshake whereas `SCTP` uses a four-way one. In this case, a state cookie signed by the first host is involved in the `SCTP` four-way handshake. This helps to protect from denial of service attacks. Figure 1 illustrates the handshake procedures in `TCP` and `SCTP`.

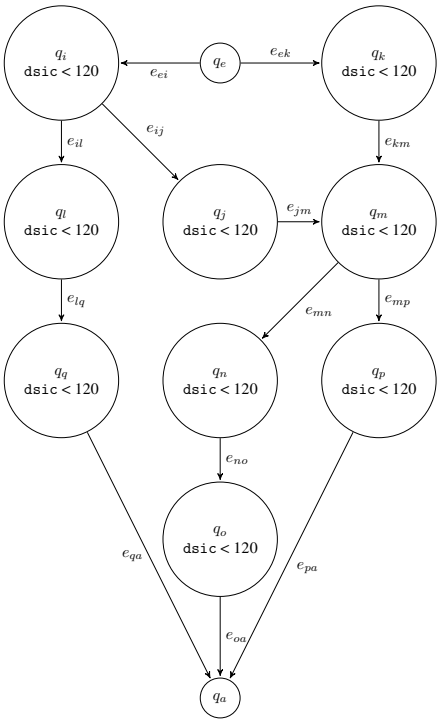
A timed automaton model of the `SCTP` handshake is depicted in Figure 2. Next, we describe a typical run of this part of the protocol. In order to simplify the presentation we assume that we have two hosts. The generalization to n hosts essentially consists in labeling each message $!m$ (resp. $?m$) as $!m_{xy}$ (resp. $?m_{xy}$) to denote that host x sends



$$
\begin{array}{lccccccc}
e_{ab} = & (q_a & , & \texttt{!INIT} & , & \texttt{true} & , & \{\texttt{init}\} & , & q_b) \\
e_{af} = & (q_a & , & \texttt{?INIT} & , & \texttt{true} & , & \{\texttt{conn}\} & , & q_f) \\
e_{b1} = & (q_b & , & \tau & , & \texttt{init} \geq 60 & , & \{\} & , & q_a) \\
e_{b2} = & (q_b & , & \texttt{!ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{b3} = & (q_b & , & \texttt{?ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{bc} = & (q_b & , & \texttt{?INIT-ACK} & , & \texttt{init} < 60 & , & \{\} & , & q_c) \\
e_{c1} = & (q_c & , & \tau & , & \texttt{init} \geq 60 & , & \{\} & , & q_a) \\
e_{c2} = & (q_c & , & \texttt{!ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{c3} = & (q_c & , & \texttt{?ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{cd} = & (q_c & , & \texttt{!COOKIE-ECHO} & , & \texttt{init} < 60 & , & \{\texttt{cookie}\} & , & q_d) \\
e_{d1} = & (q_d & , & \tau & , & \texttt{cookie} \geq 70 & , & \{\} & , & q_a) \\
e_{d2} = & (q_d & , & \texttt{!ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{d3} = & (q_d & , & \texttt{?ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{de} = & (q_d & , & \texttt{?COOKIE-ACK} & , & \texttt{cookie} < 70 & , & \{\} & , & q_e) \\
e_{e1} = & (q_e & , & \texttt{?ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{e2} = & (q_e & , & \texttt{!ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{fg} = & (q_f & , & \texttt{!INIT-ACK} & , & \texttt{conn} < 60 & , & \{\} & , & q_g) \\
e_{f1} = & (q_f & , & \tau & , & \texttt{conn} \geq 60 & , & \{\} & , & q_a) \\
e_{f2} = & (q_f & , & \texttt{!ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{f3} = & (q_f & , & \texttt{?ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{gh} = & (q_g & , & \texttt{?COOKIE-ECHO} & , & \texttt{conn} < 60 & , & \{\texttt{conn}\} & , & q_h) \\
e_{gf} = & (q_g & , & \texttt{?INIT} & , & \texttt{true} & , & \{\texttt{conn}\} & , & q_f) \\
e_{g1} = & (q_g & , & \tau & , & \texttt{conn} \geq 60 & , & \{\} & , & q_a) \\
e_{g2} = & (q_g & , & \texttt{!ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{g3} = & (q_g & , & \texttt{?ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{he} = & (q_h & , & \texttt{!COOKIE-ACK} & , & \texttt{conn} < 70 & , & \{\} & , & q_e) \\
e_{h1} = & (q_h & , & \tau & , & \texttt{conn} \geq 70 & , & \{\} & , & q_a) \\
e_{h2} = & (q_h & , & \texttt{!ABORT} & , & \texttt{true} & , & \{\} & , & q_a) \\
e_{h3} = & (q_h & , & \texttt{?ABORT} & , & \texttt{true} & , & \{\} & , & q_a)
\end{array}
$$

Figure 2. Transitions of the handshake phase automaton of the `SCTP` **protocol.**

the message m to y (resp. host x receives the message m from y). Obviously, both hosts are specified with the same timed automata. As usual, inputs of one host will be the outputs of the other one, and viceversa.



$e_{ei} = (q_e, \texttt{!SHUTDOWN}, \texttt{true}, \{\texttt{dsic}\}, q_i)$
$e_{ek} = (q_e, \texttt{?SHUTDOWN}, \texttt{true}, \{\texttt{dsic}\}, q_k)$
$e_{ij} = (q_i, \texttt{?SHUTDOWN}, \texttt{true}, \{\}, q_j)$
$e_{il} = (q_i, \texttt{?SHUTDOWN-ACK}, \texttt{true}, \{\}, q_l)$
$e_{jm} = (q_j, \texttt{!SHUTDOWN-ACK}, \texttt{true}, \{\}, q_m)$
$e_{lq} = (q_l, \texttt{!SHUTDOWN-COMPLETE}, \texttt{true}, \{\}, q_q)$
$e_{km} = (q_k, \texttt{!SHUTDOWN-ACK}, \texttt{true}, \{\}, q_m)$
$e_{mn} = (q_m, \texttt{?SHUTDOWN-ACK}, \texttt{true}, \{\}, q_n)$
$e_{mp} = (q_m, \texttt{?SHUTDOWN-COMPLETE}, \texttt{true}, \{\}, q_p)$
$e_{no} = (q_n, \texttt{!SHUTDOWN-COMPLETE}, \texttt{true}, \{\}, q_o)$
$e_{oa} = (q_o, \tau, \texttt{true}, \{\}, q_a)$
$e_{pa} = (q_p, \tau, \texttt{true}, \{\}, q_a)$
$e_{qa} = (q_q, \tau, \texttt{true}, \{\}, q_a)$
$e_{ia} = (q_i, \tau, \texttt{dsic} \geq 120, \{\}, q_a)$
$e_{ja} = (q_j, \tau, \texttt{dsic} \geq 120, \{\}, q_a)$
$e_{ka} = (q_k, \tau, \texttt{dsic} \geq 120, \{\}, q_a)$
$e_{la} = (q_l, \tau, \texttt{dsic} \geq 120, \{\}, q_a)$
$e_{ma} = (q_m, \tau, \texttt{dsic} \geq 120, \{\}, q_a)$
$e_{na} = (q_n, \tau, \texttt{dsic} \geq 120, \{\}, q_a)$

Figure 3. Transitions of the disconnection phase automaton of the SCTP protocol.

Let us consider that host A starts at state q_a by sending an INIT message (!INIT) to host B; a clock named init is activated (see transition e_{ab}), its value is initialized to zero, and host A moves to state q_b. While $\nu(\texttt{init})$ is less than sixty time units, host A can remain in this state. If the valuation $\nu(\texttt{init})$ becomes greater than or equal to sixty, then host A returns to q_a by performing the transition e_{b1}, that is, an internal transition.

Let us consider now host B, which starts also at state q_a and receives the INIT message (?INIT). Then, this host moves to q_f by performing the transition e_{af}, initializes to zero the conn clock, and while $\nu(\texttt{conn})$ is less than sixty time units, it can send an INIT-ACK message (!INIT-ACK) and move to q_g by performing the transition e_{fg}. Let us note that host A has an activated clock init. If $\nu(\texttt{init})$ is less than sixty time units and it receives an INIT-ACK message (?INIT-ACK), it moves to q_c (see transition e_{bc}). Afterwards, host A can send COOKIE-ECHO message (!COOKIE-ECHO) if $\nu(\texttt{init})$ is less than sixty, and it would move to q_d (see transition e_{cd}). If host A moves to q_d, then it will activate the cookie clock; if $\nu(\texttt{init})$ is greater than or equal to sixty in state q_c then host A returns to q_a (see transition e_{c1}).

Host B remains in q_g and receives a COOKIE-ECHO message (?COOKIE-ECHO) (see transition e_{gh}). If $\nu(\texttt{conn})$ in host B is less than seventy time units, then host B can send a COOKIE-ACK message (!COOKIE-ACK) and move to q_e, which is the established communication state. On the contrary, if the conn clock has a value greater than or equal to seventy time units, then host B moves to q_a. Host A receives this message and if the cookie clock is less than seventy time units, then it moves to q_e.

Let us note that hosts A and B can abort the handshake phase by sending the ABORT message (!ABORT) from any state. These transitions (see e_{b2}, e_{c2}, e_{d2}, e_{f2}, etc) do not have any associated time constraint, and both hosts are able to, in any state, perform a transition ABORT to receive an abort message (?ABORT) (see e_{b3}, e_{c3}, e_{d3}, e_{f3}, etc).

A timed automaton model for the SCTP disconnection stage is shown in Figure 3. Let us remark that the complete set of transitions are not drawn in Figure 3, but are explicitly defined in the accompanying set of transitions (see the last six transitions). As with the previous stage of the protocol, let us describe a typical run of the disconnection phase. We consider that the two hosts are initially placed in the q_e state. Let us note that q_e corresponds to the same state in Figures 2 and 3. Next, we consider that host A wants to finish the connection with host B. Host A sends the SHUTDOWN message (!SHUTDOWN) and starts its dsic clock, changing its state to q_i (see transition e_{ei}). Host A can stay in q_i while $\nu(\texttt{dsic})$ is less than one hundred and twenty time units. If the valuation of dsic is greater than or equal to this amount, then host A performs the e_{ia} transition. Host B acknowledges the reception of the SHUTDOWN message (?SHUTDOWN) and changes to q_k, starting the counter of the dsic clock. Next, host B generates the !SHUTDOWN-ACK output and changes its state to q_m (see transition e_{km}). Host A receives ?SHUTDOWN-ACK and changes its state to q_l. Then, host A generates a !SHUTDOWN-COMPLETE (see transition e_{lq}). Host B receives a ?SHUTDOWN-COMPLETE message and changes to state q_p, finishing the transmission between hosts A and B.

4 Definition of timed invariants

In this section we show how to define timed invariants in our framework. First, we briefly discuss some approaches on how to obtain the set of timed invariants. The first approach assumes that this set is supplied by the expert/tester. Being provided with a specification, we determine the correctness of the set of timed invariants with respect to the specification. Essentially, an invariant is correct if the property that it describes does not contradict the specification. Obviously, incorrect invariants are discarded. Another approach consists in automatically extract invariants from the specification. In this case, we can adapt to our framework the algorithms given in [CGP03]. The problem with this approach is that the number of possible invariants is usually huge, and we are not provided with a criterium to choose one invariant instead of another. A third alternative is to assume that invariants are correct *by definition*. In this situation, a specification is not needed and the invariants can be considered as the *requirements* of the system to be implemented. Next we define the formal syntax to express timed invariants.

Definition 4 We say that $\hat{p} = [p_1, p_2]$ is a *time interval* if $p_1 \in \mathbf{R}_+$, $p_2 \in \mathbf{R}_+ \cup \{\infty\}$, and $p_1 \leq p_2$. We assume that for all $t \in \mathbf{R}_+$ we have $t < \infty$ and $t + \infty = \infty$. We consider that $\mathcal{IR}$ denotes the set of time intervals. Let us note that in the case of $[t, \infty]$ we are abusing the notation since this interval represents, in fact, the interval $[t, \infty)$.

Let $\mathcal{A} = (\mathcal{S}, s_0, \Sigma, \mathcal{C}, \mathcal{Z}, \mathcal{E})$ be a timed automaton. We say that the sequence ψ is a *timed invariant* for $\mathcal{A}$ if the following two conditions hold:

1. ψ is defined according to the following EBNF:

$$\psi ::= \alpha/\hat{p}, \psi \mid \star, \psi' \mid \alpha'/\hat{p} \mapsto A \rhd \hat{q}$$
$$\psi' ::= \alpha'/\hat{p}, \psi \mid \alpha'/\hat{p} \mapsto A \rhd \hat{q}$$

   In this expression we consider $\hat{p}, \hat{q} \in \mathcal{IR}$, $\alpha' \in \Sigma$, $\alpha \in \Sigma \cup \{?\}$, and $A \subseteq \Sigma$.

2. ψ is *correct* with respect to $\mathcal{A}$.

□

Due to the space limitations we do not include the algorithm to decide correctness. This algorithm is an adaption of the one presented in [AMN08] for timed finite state machines. Essentially, ψ is correct for $\mathcal{A}$ if its associated LTS $\mathcal{M}_{\mathcal{A}}$ cannot produce a trace that *contradicts* the meaning of ψ.

Intuitively, the previous EBNF expresses that an invariant is a sequence of symbols where each component, but the last one, is either a pair $\alpha/\hat{p}$, with α being an action or the wild-card character ? and $\hat{p}$ being a time interval, or an expression $\star$. There are two restrictions to this rule. First, an invariant cannot contain two consecutive $\star$'s since this is the same as having just one. The second restriction is that an invariant cannot have a $\star$ followed by ?, that is, the action of the next component must be a *real* action belonging to Σ. In fact, given an invariant $\ldots, \star, \alpha'/\hat{p_2}, \ldots$ we have that $\star$ represents any sequence of actions without occurrences of α'. Finally, the last component of an invariant, $\alpha'/\hat{p} \mapsto A \rhd \hat{q}$, represents an action associated with a time interval, followed by a set of actions and another time interval. This last interval is used to control the sum of time values associated to all the actions performed during matching the invariant. For a more precise, and formal, explanation of timed invariants the interested reader is referred to [AMN08].

Example 1 Next we present two timed invariants for the SCTP specification. A simple timed invariant is

$$\psi_1 = \ \texttt{?INIT}/[0,50] \mapsto \{\texttt{!ABORT}, \texttt{!INIT-ACK}\} \rhd [0,50]$$

This invariant represents that if we observe ?INIT and we have a delay less than 50 time units before we observe the next action, then this next action must be either !ABORT or !INIT-ACK. Let us show an informal way to check the correctness of ψ_1 with respect to SCTP. If host A starts on q_a and performs e_{af}, then the action !INIT appears in the trace. If host B starts on q_a and receives ?INIT, then it performs e_{ab} and the set of all possible actions that host A can perform in a time belonging to $[0, 50]$ are associated to the transitions e_{f2} and e_{fg}. This means that either !ABORT or !INIT-ACK are performed.

Next, we present another invariant using the wild-card character $\star$. The invariant

$$\psi_2 = \ \texttt{?INIT}/[0,70], \star, \texttt{!ABORT}/[0,60] \mapsto \{\texttt{!INIT}, \texttt{?ABORT}\} \rhd [0,\infty]$$

represents that when a host performs the action ?INIT (the handshake stage have just started), we allow the trace to perform actions (matching the $\star$ symbol) until the action !ABORT appears in the trace. If this occurrence happens before 70 time units and the next action is observed before 60 time units pass, then this last observed action must be either !INIT or ?ABORT. The performance of all these actions can take any time (since it belongs to the interval $[0, \infty]$). □

5 Application of our methodology to the SCTP

In addition to our theoretical framework, and in order to automatize the passive testing of the SCTP, we extended

$\psi_1 =$?INIT/$[0,50] \mapsto \{$!ABORT, !INIT-ACK$\} \rhd [0,50]$

$\psi_2 =$!ABORT/$[0,60] \mapsto \{$!INIT, ?ABORT$\} \rhd [0,\infty]$

$\psi_3 =$!INIT/$[0,60]$, ?INIT/$[0,60]$, !INIT-ACK/$[0,60]$, ?INIT-ACK/$[0,60]$, !COOKIE-ECHO/$[0,60]$, ?COOKIE-ECHO/$[0,70]$, !COOKIE-ACK/$[0,70] \mapsto \{$!ABORT, ?COOKIE-ACK$\} \rhd [0,130]$

$\psi_4 =$!SHUTDOWN/$[0,120]$, ?SHUTDOWN/$[0,120]$, !SHUTDOWN-ACK/$[0,120]$, ?SHUTDOWN-ACK/$[0,120]$, !SHUTDOWN-COMPLETE/$[0,120] \mapsto \{$!INIT, ?SHUTDOWN-COMPLETE$\} \rhd [0,120]$

$\psi_5 =$?SHUTDOWN-ACK/$[0,120] \mapsto \{$!INIT, !SHUTDOWN-COMPLETE$\} \rhd [0,120]$

$\psi_6 =$!SHUTDOWN/$[121,\infty] \mapsto \{$!INIT$\} \rhd [121,\infty]$

Figure 4. Invariants used to analyze the disconnection and handshake phases of the SCTP.

our PASTE tool to deal with the new framework (some experiments with the *old* tool are reported in [AMN09]). The original tool was implemented in JAVA and it was an isolated project. Later, it was decided to include our academic tool as a module of a more important monitoring tool, developed by the SME Peopleware, called OSMIUS www.osmius.net. This is a monitoring tool released under the GPLv2 license and available in the open source repository Sourceforge. Osmius is designed to be extensible so that it can be used to monitor any device or software connected to a network. Therefore, the new version of PASTE represents the complete migration of the code from JAVA to C++. This new version includes the timed framework described in this paper, labeled transition systems and timed invariants, as well as two algorithms to check the correctness of a timed invariant with respect to a specification and of traces with respect to timed invariants.

In this paper we focus on a functionality of PASTE that provides a *monitor* to check timed invariants, that is, a service that reads a finite log and yields a certain verdict. Normally, monitors are classified either as *online monitoring*, where the monitor should consider executions in an incremental fashion and in an efficient manner, or as *offline monitoring*, where the monitor works on a finite set of recorded logs. PASTE is implemented by using an offline monitoring paradigm.

Next we present the general scheme of all the experiments performed with PASTE. They start with the insertion in the system, by using an XML format file, of the specification (in our case, the timed automata specifying the SCTP). Next, we have to provide to PASTE the set of proposed timed invariants. Invariants are automatically checked with respect to the specification to determine their correctness. After that, PASTE is ready to check the logs with respect to the set of correct invariants. Real logs are files containing the sequential interactions of systems under test performing the SCTP. We use the Wireshark sniffer for monitoring tasks. As we have already mentioned, we use three different sources to obtain traces. Wireshark allows us to export the traces in an XML format file. Each XML file includes several XML-nodes which have the following fields: N, Source, Destination, Protocol, and Info. N represents the order into the XML file of this XML-node, Source and Destination represent the Source IP and the destination IP of the hosts involved in this communication, respectively, Protocol includes the name of this XML-node protocol, and Info belongs to the Σ vocabulary of the SCTP specification.

When we provide a set of logs to PASTE, the tool formats them into its internal format. Essentially, irrelevant information is removed in order to generate a normalized visible trace (see Definition 3). In particular, PASTE fixes an IP address as the observational host and then it formats all XML-nodes in the log as outputs if the Source value is the fixed IP (PASTE adds the ! character at the beginning of Info). XML-nodes formats as inputs those messages whose Destination field value coincides with our observational host IP (PASTE adds the ? character at the beginning of the Info string).

In Figure 4 we enumerate the most relevant invariants that we used to analyze the disconnection and handshake phases of the SCTP. ψ_1 and ψ_2 were commented in the previous section. ψ_3 and ψ_4 were informally introduced in Section 3 and represent the most usual runs of the handshake and disconnection phases, respectively. ψ_5 represents the idea that if a host finalizes the disconnection phase, by sending ?SHUTDOWN-ACK, then its associated node must perform either !INIT or ?SHUTDOWN-COMPLETE. Finally, ψ_6 represents that if the timed constraint associated with the dsic clock triggers, then the next observed action will be !INIT.

Next we checked the correctness of logs with respect to these invariants. Since we only have logs extracted from correctly implemented systems, we implemented in PASTE

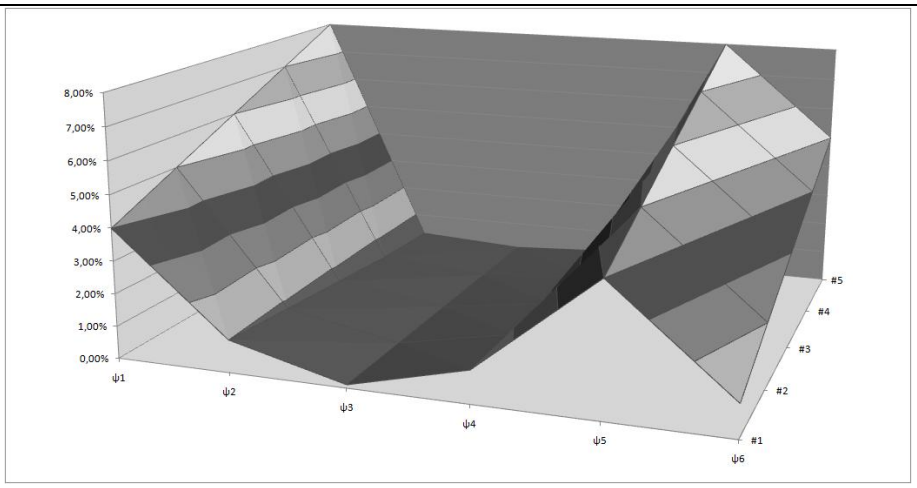


Figure 5. Errors detected by using the invariants $\psi_1, \ldots, \psi_6$.

a module to create *mutated traces*. The underlying idea of mutating traces is to simulate a *common* error (e.g. generated from network delays, white noise, implementation faults, etc). PASTE implements two mutation operators that can be applied to logs. These operators are parameterized so that we derive different mutated traces from the original log.

- *Sensitivity*. It is used to set the percentage of variation to be applied to a temporal value of the trace.
- *Noise threshold*. It simulates the white noise appearing in networks. This noise produces errors in the bites of the trace. For example, when we apply this operator to a log, it may transform an action $!\alpha$ into $?\alpha$, since the bites associated with the IP source and IP destination of this trace could be swaped. Other things that can happen is that an action α is transformed into a different action α'. Finally, it may also happen that the noise produces that the action is completely removed.

So, we consider that when there exists a set of correct timed invariants and a set of logs extracted from a correct implementation, we can simulate common implementation errors by applying *sensitivity* and *noise threshold* over these traces. This approach allows us to classify invariants with respect to their error detection power. Therefore, if we have to perform passive testing against an IUT, then we will exercise first those invariants having a better error detection factor.

In Figure 5 we present our results after experimenting with the set of proposed invariants and the SCTP. In the $X-$axe we include the considered invariants. In the $Y-$axe, the values $\#1, \ldots, \#5$ represent the number of mutations introduced in the log. Finally, the $Z-$axe contains values to represent the percentage of errors found in traces.

According to our experiments, the best invariants of this set are ψ_1 and ψ_5. Our experiments showed that having an empty preface, as these two invariants have, is *good* to increase the error detection power of invariants. The reason is that this type of invariants can detect an error already after observing the occurrence of a message; other invariants need to observe the whole preface before been able to detect errors. However, an empty preface is not a sufficient condition to have a *good* invariant. Let us comment ψ_2 and ψ_6. Both of them have empty prefaces. However, in the case of ψ_2 the occurrence frequency of the !ABORT message is very low. In the case of ψ_6, that contains *usual* actions, we have that the frequency of observing a disconnection immediately followed by a request for connection is also very low. Finally, even though ψ_3 and ψ_4 represent the most common runs of the handshake and the disconnection phases, respectively, the percentage of errors detected by these invariants is relatively low because before checking the last component of the invariant, where we can claim that an error has been detected, we must observe the whole preface.

6 Conclusions and future work

In this paper we have presented a case study to show the application of formal passive testing to a network protocol. We reviewed timed automata and adapted the notion of timed invariants, presented in previous work in the context of timed finite state machines, to deal with timed automata. We provided a complete specification of two stages of the SCTP protocol: The handshake and disconnection

phases. We presented a methodology to classify timed invariants with respect to their error detection power. In our experiments we focused on properties related to temporal behavior. After the performed experiments and the gained experience, we consider that our technique can be used in other complex network protocols.

We have two lines of future work. The first one, with a theoretical component, consists in advancing in the study of the error detection power of timed invariants. In particular, we plan to study the relation between invariants and *usual* test cases that can be generated from them. Moreover, we plan to formally study the relation between the length of invariants and their error detection power. A second line of work, more practical, consists in analyzing other software systems and real protocols. We are specially interested in working with p2p protocols, and tools implementing them, such as MANOLITO, SoulSeek and BitTorrent.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AMM09] C. Andrés, M.G. Merayo, and C. Molinero. Advantages of mutation in passive testing: An empirical study. In *4th Workshop on Mutation Analysis, Mutation'09*, pages 230–239. IEEE Computer Society Press, 2009.

[AMN08] C. Andrés, M.G. Merayo, and M. Núñez. Passive testing of timed systems. In *6th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'08, LNCS 5311*, pages 418–427. Springer, 2008.

[AMN09] C. Andrés, M.G. Merayo, and M. Núñez. Formal correctness of a passive testing approach for timed systems. In *5th Workshop on Advances in Model Based Testing, A-MOST'09*, pages 67–76. IEEE Computer Society Press, 2009.

[BCNZ05] E. Bayse, A. Cavalli, M. Núñez, and F. Zaïdi. A passive testing approach based on invariants: Application to the WAP. *Computer Networks*, 48(2):247–266, 2005.

[BDS+07] A. Benharref, R. Dssouli, M.A. Serhani, A. En-Nouaary, and R. Glitho. New approach for EFSM-based passive testing of web services. In *Joint 19th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'07, and 7th Int. Workshop on Formal Approaches to Software Testing, FATES'07, LNCS 4581*, pages 13–27. Springer, 2007.

[BT01] E. Brinksma and J. Tretmans. Testing transition systems: An annotated bibliography. In *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, pages 187–195. Springer, 2001.

[BU91] B.S. Bosik and M.Ü. Uyar. Finite state machine based formal methods in protocol conformance testing. *Computer Networks & ISDN Systems*, 22:7–33, 1991.

[CGP03] A. Cavalli, C. Gervy, and S. Prokopenko. New approaches for passive testing using an extended finite state machine specification. *Information and Software Technology*, 45:837–852, 2003.

[EDK02] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Wp-method: Testing real time systems. *IEEE Transactions on Software Engineering*, 28(11):1024–1039, 2002.

[HBB+09] R.M. Hierons, K. Bogdanov, J.P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A.J.H Simons, S. Vilkomir, M.R. Woodward, and H. Zedan. Using formal methods to support testing. *ACM Computing Surveys*, 41(2), 2009.

[HBH08] R.M. Hierons, J.P. Bowen, and M. Harman, editors. *Formal Methods and Testing, LNCS 4949*. Springer, 2008.

[HMN09] R.M. Hierons, M.G. Merayo, and M. Núñez. Testing from a stochastic timed system with a fault model. *Journal of Logic and Algebraic Programming*, 78(2):98–115, 2009.

[HNTC99] T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli. Generating test cases for a timed I/O automaton model. In *12th Int. Workshop on Testing of Communicating Systems, IWTCS'99*, pages 197–214. Kluwer Academic Publishers, 1999.

[How82] W.E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, 8:371–379, 1982.

[HW05] G.-D. Huang and F. Wang. Automatic test case generation with region-related coverage annotations for real-time systems. In *3rd Int. Symposium on Automated Technology for Verifica-*

tion and Analysis, ATVA'05, LNCS 3707, pages 144–158. Springer, 2005.

[Int07] International Engineering Consortium. Stream control transmission protocol: Definition and overview. Available at http://www.iec.org/online/tutorials/sctp/, 2007.

[LCH+02] D. Lee, D. Chen, R. Hao, R. Miller, J. Wu, and X. Yin. A formal approach for passive testing of protocol data portions. In *10th IEEE Int. Conf. on Network Protocols, ICNP'02*, pages 122–131. IEEE Computer Society Press, 2002.

[LNR06] N. López, M. Núñez, and I. Rodríguez. Specification, testing and implementation relations for symbolic-probabilistic systems. *Theoretical Computer Science*, 353(1–3):228–248, 2006.

[LNS+97] D. Lee, A.N. Netravali, K.K. Sabnani, B. Sugla, and A. John. Passive testing and applications to network management. In *5th IEEE Int. Conf. on Network Protocols, ICNP'97*, pages 113–122. IEEE Computer Society Press, 1997.

[LS09] M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[LY96] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[Mil98] R.E. Miller. Passive testing of networks using a CFSM specification. In *IEEE Int. Performance Computing and Communications Conference*, pages 111–116. IEEE Computer Society Press, 1998.

[MMM95] D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real time systems from logic specifications. *ACM Transactions on Computer Systems*, 13(4):356–398, 1995.

[MNR08a] M.G. Merayo, M. Núñez, and I. Rodríguez. Extending EFSMs to specify and test timed systems with action durations and timeouts. *IEEE Transactions on Computers*, 57(6):835–848, 2008.

[MNR08b] M.G. Merayo, M. Núñez, and I. Rodríguez. Formal testing from timed finite state machines. *Computer Networks*, 52(2):432–460, 2008.

[OL94] A.J. Offutt and S.D. Lee. An empirical evaluation of weak mutation. *IEEE Transactions on Software Engineering*, 20(5):337–344, 1994.

[OPTZ99] A.J. Offutt, J. Pan, K. Tewary, and T. Zhang. An experimental evaluation of data flow and mutation testing. *Software: Practice and Experience*, 26(2):165 – 176, 1999.

[Pet01] A. Petrenko. Fault model-driven test derivation from finite state models: Annotated bibliography. In *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, pages 196–205. Springer, 2001.

[RMN08] I. Rodríguez, M.G. Merayo, and M. Núñez. $\mathcal{HOTL}$: Hypotheses and observations testing logic. *Journal of Logic and Algebraic Programming*, 74(2):57–93, 2008.

[SVD01] J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001. Previously appeared as Technical Report CTIT-97-17, University of Twente, 1997.

[SXM+00] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream control transmission protocol: RFC 2960. Available at http://www.ietf.org/rfc/rfc2960.txt, 2000.

[TC99] M. Tabourier and A. Cavalli. Passive testing and application to the GSM-MAP protocol. *Information and Software Technology*, 41:813–821, 1999.

[Tre96] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.

[UX07] H. Ural and Z. Xu. An EFSM-based passive fault detection approach. In *Joint 19th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'07, and 7th Int. Workshop on Formal Approaches to Software Testing, FATES'07, LNCS 4581*, pages 335–350. Springer, 2007.

[WZY01] J. Wu, Y. Zhao, and X. Yin. From active to passive: Progress in testing of internet routing protocols. In *21st IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'01*, pages 101–116. Kluwer Academic Publishers, 2001.