Science in China Series F: Information Sciences

www.scichina.com
info.scichina.com
www.springerlink.com

A formal method to real-time protocol interoperability testing

WANG ZhiLiang[1,3†], YIN Xia[2,3] & JING ChuanMing[2,3]

[1] Network Research Center, Tsinghua University, Beijing 100084, China;
[2] Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China;
[3] Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing 100084, China

Interoperability testing is an important technique to ensure the quality of implementations of network communication protocol. In the next generation Internet protocol, real-time applications should be supported effectively. However, time constraints were not considered in the related studies of protocol interoperability testing, so existing interoperability testing methods are difficult to be applied in real-time protocol interoperability testing. In this paper, a formal method to real-time protocol interoperability testing is proposed. Firstly, a formal model CMpTIOA (communicating multi-port timed input output automata) is defined to specify the system under test (SUT) in real-time protocol interoperability testing; based on this model, timed interoperability relation is then defined. In order to check this relation, a test generation method is presented to generate a parameterized test behavior tree from SUT model; a mechanism of executability pre-determination is also integrated in the test generation method to alleviate state space explosion problem to some extent. The proposed theory and method are then applied in interoperability testing of IPv6 neighbor discovery protocol to show the feasibility of this method.

protocol testing, interoperability testing, real-time testing, test generation, timed input output automata (TIOA)

1 Introduction

Recently, as the emergence of new techniques such as P2P and media steaming, more real-time requirements should be satisfied in Internet applications. Compared with the current Internet, in the next generation Internet, network protocols supporting real-time applications should require stricter timing control capability. In such network protocols with real-time requirements, not only input and output actions, but also time constraints of I/O behaviors, should be carefully considered.

Received June 3, 2008; accepted August 21, 2008
doi: 10.1007/s11432-008-0153-7
†Corresponding author (email: wzl@cernet.edu.cn)
Supported by the National Basic Research Program of China (973 Program) (Grant No. 2003CB314801), and the National Natural Science Foundation of China (Grant No. 60572082)

In order to ensure the quality of network protocol implementation, protocol testing techniques are widely used. Conformance testing is a basic method of protocol testing, which can be used to test whether an implementation conforms to its protocol specification. In the field of conformance testing, many literatures have been published, such as international standard ISO/IEC 9646[1], which has shown the mature protocol conformance testing theory and methodology. As the complement of conformance testing, interoperability testing is often used to test whether two or more protocol implementations can communicate with each other correctly and inter-operate as a whole system to perform functions specified in protocol specifications. Interoperability testing is necessary because 1) it is difficult to perform exhaustive conformance testing, that is, a conformance test suite can hardly ensure 100% test coverage; 2) many optional features may be contained in network protocols, and moreover, network devices vendors perhaps have their own extensions, so if two implementations implement different options, problems on interoperability may occur. As the network scale grows, the version of network protocols will be updated continuously and devices vendors will also release their newer network devices continuously. Because of more differences of various implementations for the same protocol in network, protocol interoperability testing will become more important. Interoperability testing is being performed by many international standardization organizations, e.g., IETF (Internet Engineering Task Force) and ETSI (European Telecommunications Standards Institute) in their process of protocol design.

A lot of work has been done in the area of formal interoperability testing[2−14], and a series of interoperability testing framework and test generation method have been proposed. In most of real-life network protocols, not only the behaviors of input and output, but also their time of occurrence should be considered. In order to test such real-time protocols, we should check if the I/O behaviors act under the specified time constraints. Unfortunately, in the previous studies, no timing factors have been considered in interoperability testing. So these methods are difficult to be applied in interoperability testing of real-life network protocols. In the field of real-time system testing, many methods of conformance testing have been proposed[15−21]. Ref. [22] studied interoperability test generation of time dependent protocols; however, no theoretical framework of real-time protocol interoperability testing was established.

In this paper, we intend to propose a formal framework of real-time protocol interoperability testing. Firstly, a formal model CMpTIOA (communicating multi-port timed input output automata) is defined to specify the system under test (SUT) in real-time protocol interoperability testing; timed interoperability relation is then defined based on this model. After that, a test generation method guided by timed interoperability relation is presented to generate a parameterized test behavior tree, in which parameters are used to represent the relative time intervals between I/O behaviors. At last, the proposed theory and method are then applied in interoperability testing of IPv6 neighbor discovery protocol, which has complex time constraints, to show the feasibility of our method.

The rest of the paper is structured as follows. Section 2 introduces related work. Section 3 describes the whole process of protocol interoperability testing. In section 4, the formal models used in real-time protocol interoperability testing are defined. Section 5 defines the timed interoperability relation based on the formal model. In section 6, a test generation method is then presented. Section 7 applies the proposed theory and method in real-life network protocol testing. Conclusions and future work are given in section 8.

2 Related work

In the research field of formal interoperability testing, the commonly used formal models can be classified into two categories: FSM (finite state machine)[23] and IOTS (input output transition system)[24]. In the aspect of formal framework of protocol interoperability testing, ref. [2] proposed a TTCN2-based framework for interoperability testing; ref. [3] presented a formal framework based on IOTS model and defined several test architectures and interoperability relations to guide test generation.

Interoperability test generation, whose purpose is to generate test sequences from formal model of system under test, is an important issue in this field. The basic idea of test generation method based on FSM is to model each protocol entity as an FSM and the interoperability system under test as a CFSM (communicating FSMs), and then generate test sequences using global state reachability analysis techniques[4]. Based on this idea, a series of test generation techniques have been proposed[5−10]: ref. [5] presented a systematic test suite derivation method for protocol control portion interoperability testing based on single stimulus principle; ref. [6] analyzed the fault coverage of interoperability test suite generated with the method of ref. [5] and proposed an enhanced test generation method to improve fault coverage; ref. [7] developed an automatic test generation method considering both control and data portion of protocols; ref. [8] extended the method in ref. [5] based on multiple stimuli principle to improve test coverage, but this method suffers from state space explosion problem; ref. [9] studied test generation method based on multiple stimuli model more deeply and presented two methods to generate tests of checking live-locks and interoperability; ref. [10] presented an interoperability test generation method based on communicating multi-port FSMs (CMpFSM) model and generated distributed test sequences on distributed test architecture. Different from the above literatures, ref. [11] focused on interoperability testing of VoIP protocol system and proposed an efficient method that only considered the specification of one protocol entity. With this method, there is no need to generate global state reachability graph so that state space explosion problem can be alleviated. But this method, without considering the composition of all protocol entities in interoperability system, is essentially still a conformance testing method. Another category of test generation methods is based on IOTS[12−14], basically following the ideas of ref. [3]. Ref. [12] added "quiescence management" in IOTS model, and used suspensive IOTS as system model. Refs. [13,14] proposed an interoperability test generation method based on test purpose, which avoids state space explosion problem. However, in all these studies of interoperability testing, time constraints in protocols are not considered, so existing theory and methods are difficult to be applied in real-time protocol interoperability testing.

In the field of real-time system testing, many methods of conformance testing have been proposed. In most of these methods, Timed Automaton[25] or its variants have been used to specify real-time system. Refs. [15,16] converted timed automaton to grid automaton, and applied existing test generation methods for FSM to it. But this method suffers from state space explosion problem. Ref. [17] presented a test generation method based on executability decision. Refs. [18−21] defined timed conformance relations, and proposed corresponding test generation methods. But existing work only considered conformance testing of real-time system, and real-time protocol interoperability testing has not been studied.

Compared with the existing work, this paper considers time constraints in protocol interopera-

bility testing, applies real-time system model to specify protocol behaviors and presents a formal method to real-time protocol interoperability testing.

3 Basic process of interoperability testing

As one kind of protocol testing, the basic process of interoperability testing is similar with conformance testing[1]. Figure 1 shows the whole process of interoperability testing. There are m protocol entities in an interoperability system under test. The left branch represents the implementation (development) process of protocol entities. After this process, a set of possible protocol implementations correspond to each protocol specification. Fetching out one implementation from each set, we get a system under test which is composed of m implementation under test (IMP_1, IMP_2, …, IMP_m). The right branch represents the testing implementation process. In the process of test generation, abstract interoperability test suite can be generated from system specification composed of several protocol specifications. Then, in the process of test implementation, abstract interoperability test suite can be converted into executable test suite. Test execution is just the process that test system interacts with system under test based on test suite. At last, test verdict will be obtained from the results of test execution. As the testing basis, interoperability relation between implementations and specifications is the guideline in the whole process. In the process of test generation, the abstract test suite should be generated according to interoperability relation. Corresponding to the above process, section 4 of this paper defines a formal model to specify system under test; section 5 defines timed interoperability relation based on the formal model; section 6 studies interoperability test generation.

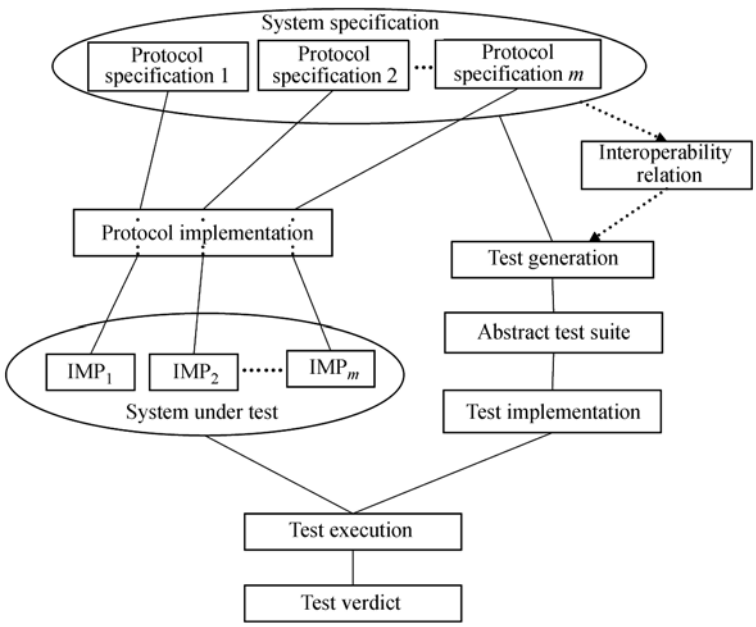

Figure 1 Progress of protocol interoperability testing.

4 Formal model

This section presents the formal models used in real-time interoperability testing. We specify the

interoperability system under test as a system of CMpTIOA (communicating multi-port timed input output automata). A simple real-time communication protocol is then given and specified using CMpTIOA model as the example in the whole paper.

4.1 Multi-port TIOA

Timed automaton[25] is a widely-used model of real-time system. TIOA (timed input output automata)[15] is a variant of timed automaton, which distinguishes whether an action is input or output. To specify an entity interacting with one or more other entities, we extend TIOA to multi-port TIOA as follows.

Definition 1. Multi-port timed input output automaton (MpTIOA)

A timed input output automaton with n ports (denoted as np-TIOA) M has n ports communicating with environment, which are denoted as P_1, P_2, …, P_n respectively. M is a 5-tuple (L, Act, l_0, C, T), where,

- L is a finite set of locations; $l_0 \in L$ is the initial location;
- $Act=I \cup O$: $I=I_1 \cup I_2 \cup \ldots \cup I_n$ is the set of input action symbols, where I_k (k=1,2,…,n) is the set of input action symbols on port P_k. An input action symbol occurring on port P_k can be denoted as $P_k?a$ ($a \in I_k$). $O=O_1 \cup O_2 \cup \ldots \cup O_n$ is the set of output action symbols, where O_k (k=1,2,…,n) is the set of output action symbols on port P_k. An output action symbol occurring on port P_k can be denoted as $P_k!b$ ($b \in O_k$).
- C is a finite set of clocks $\{t_1, t_2, \ldots, t_{|C|}\}$, where, $|C|$ is the number of clocks; $v_i \in \boldsymbol{R}^+$ (non-negative real numbers) is the clock value of t_i; $\boldsymbol{v}=(v_1, v_2, \cdots, v_{|C|})$ denotes a clock valuation.
- T is the set of transitions: $(l, a, P, R, l') \in T$, where, $l, l' \in L$ are the source and destination locations; $a \in Act$ is an input or output action symbol; P is the time constraint, which is a Boolean conjunction over linear inequalities $P(\boldsymbol{v})$. The subset $R \subseteq C$ specifies the clocks to be reset to 0. The transition $(l, a, P, R, l') \in T$ can be also denoted as $l \xrightarrow{a[P]/R} l'$.

In the model, we assume that time constraints of transitions are all in the format of $\wedge(v_i \sim d)$, where, $\sim \in \{<, >, \leqslant, \geqslant, =\}$, and $d \in \boldsymbol{R}^+$. We distinguish two urgency types[26] of transitions implicitly: 1) *Lazy*, for transitions with input actions, means that input actions may not be taken because they are controlled by environment (such a property is also called "*Unforced Inputs*"); and 2) *Delayable*, for transitions with output actions, means that the corresponding output action must be taken during such transitions' enabling time.

The semantics of MpTIOA M=(L, Act, l_0, C, T) can be defined as a TIOTS (timed input output transition system) $(S, s_0, Act, \rightarrow)$, where S is the set of states, state $s=(l, \boldsymbol{v}) \in S$, where $l \in L$ is a location, $\boldsymbol{v}=(v_1, v_2, \cdots, v_{|C|})$ is a clock valuation; $s_0=(l_0, \boldsymbol{v}_0)$ is the initial state; $\rightarrow \subseteq S \times (Act \cup \boldsymbol{R}^+) \times S$ is the set of transitions. There are two types of transitions: *Timed transitions* and *Discrete transitions*. Timed transitions $(l, \boldsymbol{v}) \xrightarrow{d} (l, \boldsymbol{v}')$ model time progress, where $d \in \boldsymbol{R}^+$ is the delaying time, $\boldsymbol{v}'=\boldsymbol{v}+\boldsymbol{d}=\boldsymbol{v}+(d, d, \cdots, d)$, i.e., in this period, no input or output transitions occur and the location is unchanged. Discrete transitions $(l, \boldsymbol{v}) \xrightarrow{a} (l', \boldsymbol{v}')$ correspond to execution of the transition (l, a, P, R, l'), where P is satisfied by $\boldsymbol{v}$($P(\boldsymbol{v})$=*true*) and $\boldsymbol{v}'$

is obtained by updating $\boldsymbol{v}$ according to R. We define the clock valuation updating function $Update_R(\boldsymbol{v})$ as follows.

Definition 2. Clock valuation updating function

$R \subseteq C$ specifies the clocks to be reset to 0. For $\boldsymbol{v} = (v_1, v_2, \cdots, v_{|C|})$, $\boldsymbol{v}' = (v_1', v_2', \cdots, v_{|C|}')$, the clock valuation updating function is defined as

$$\boldsymbol{v}' = Update_R(\boldsymbol{v}) =_{\text{def}} v_i' = \begin{cases} v_i (t_i \notin R), \\ 0 (t_i \in R). \end{cases}$$

In MpTIOA model, timed traces are used to describe the external observable behaviors of the model. We now define some related operators about timed traces.

Definition 3. Timed traces in MpTIOA[1)]

Assume the semantics model of a MpTIOA M is TIOTS $A = (S, s_0, Act, \rightarrow)$, let $\mu_i \in Act \cup \boldsymbol{R}^+$, $s, s', s_i \in S$, $\sigma \in (Act \cup \boldsymbol{R}^+)^*$,

(1) $s \xrightarrow{\mu_1 \cdots \mu_n} s' =_{\text{def}} \exists s_0, s_1, \cdots s_n, s = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} s_n = s'$.

(2) $s \xrightarrow{\sigma} s' =_{\text{def}} \exists s_0, s_1, \cdots s_n, s = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} s_n = s', \sigma = \mu_1 \mu_2 \cdots \mu_n$.

(3) $s \xrightarrow{\sigma} =_{\text{def}} \exists s', s \xrightarrow{\sigma} s'$. σ is a timed trace.

(4) s after $\sigma =_{\text{def}} \{s' \mid s \xrightarrow{\sigma} s'\}$ is the set of states which can be reached from s after the sequence of actions σ.

(5) A after $\sigma =_{\text{def}} s_0$ after σ is the set of all states reachable from the initial state s_0 of A.

(6) $out(s) =_{\text{def}} \{\alpha \in O \cup \boldsymbol{R}^+ \mid \exists s', s \xrightarrow{\alpha} s'\}$ is the set of all possible outputs from the state s.

(7) $out(S) =_{\text{def}} \bigcup_{s \in S} out(s)$ is the set of all possible outputs from the state set S.

(8) $ttraces(s) =_{\text{def}} \{\sigma \in (Act \cup \boldsymbol{R}^+)^* \mid \exists s', s \xrightarrow{\sigma} s'\}$ is the set of possible observable timed traces from s.

In a timed trace, two consecutive delaying actions can be combined, that is, if $s_1 \xrightarrow{d_1 d_2} s_2$, then $s_1 \xrightarrow{(d_1 + d_2)} s_2$. Such property is called "time additivity". So a timed trace can be reduced to the format of $\sigma = d_0 \alpha_0 d_1 \alpha_1 \cdots d_n \alpha_n (\alpha_i \in Act, d_i \in \boldsymbol{R}^+)$. In fact, d_i is just the relative time interval between observable actions α_{i-1} and α_i. In the rest of this paper, we only consider such timed traces.

To test interoperability, we make an assumption that both specifications and implementations are input-complete, that is, they can accept any inputs at any locations, formally, $\forall s \in S$, $\forall i \in I, s \xrightarrow{i}$. To make an MpTIOA model input-complete, some transitions of self-loop can be added to it, which indicates that the specification ignores the unspecified input actions.

Projection of timed traces is defined as follows.

Definition 4. Projection of timed traces

1) In this paper, we do not intend to consider unobservable internal actions in models. To consider the unobservable internal actions, some of these definitions can be only simply modified by replacing " $\rightarrow$ " with " $\Rightarrow$ ".

For a TIOTS, let a timed trace $\sigma \in (Act \cup \boldsymbol{R}^+)^*$, $\alpha \in Act$, $d \in \boldsymbol{R}^+$, an I/O action symbols set X.

(1) The projection of σ on X is denoted as σ/X. σ/X is defined as follows recursively (where ε represents null).

$$\sigma / X = \begin{cases} \varepsilon(\sigma = \varepsilon), \\ d \cdot (\sigma' / X)(\sigma = d \cdot \sigma'), \\ \alpha \cdot (\sigma' / X)(\sigma = \alpha \cdot \sigma' \text{ and } \alpha \in X), \\ \sigma' / X(\sigma = \alpha \cdot \sigma' \text{ and } \alpha \notin X). \end{cases}$$

(2) $ttraces_X(S) =_{\text{def}} \{\sigma' \mid \forall \sigma \in ttraces(S), \sigma' = \sigma / X\}$,

(3) $out_X(S) =_{\text{def}} out(S) \cap X$ is the projection of $out(S)$ on X.

In the above definition, intuitively, the projection of a timed trace on an I/O action symbols set is just to retain action symbol elements in the set and get rid of the other elements not in the set.

4.2 Communicating multi-port TIOA

To specify an interoperability system under test including two or more entities, we introduce a formal model communicating multi-port timed input output automata (CMpTIOA). In the model, MpTIOA can model the single entity, and these entities can communicate with each other via channels between different MpTIOAs.

Definition 5. Communicating multi-port timed input output automata (CMpTIOA)

A Communicating multi-port timed input output automata is a 2-tuple (M, Ch), where,

(1) $M=\{M_1, M_2, ..., M_m\}$ is a finite set of m MpTIOAs;

(2) $Ch = \{C_{ij} \mid i, j = 1,2,\ldots, m \wedge i \neq j\}$ is a finite set of channels between MpTIOAs: $C_{ij} \in Ch$ represents the communicating channel from MpTIOA M_i to M_j.

In the definition of CMpTIOA, channels are all unidirectional. Intuitively, the semantic of channels is that the output of MpTIOA M_i can be transferred via channel C_{ij} to be the input of M_j. To explicitly specify how the entities in CMpTIOA are connected to each other, i.e., the abstract network topology of system under test, we further define port connection relations as follows.

Definition 6. Port connection relations of CMpTIOA

Port connection relation R of CMpTIOA (M, Ch) is an m-tuple: $R=(R_1, R_2,\ldots, R_m)$, where, R_k (k=1,2,…,m) is the set of port connection relations for all ports of MpTIOA M_k: $R_k=\{r_1, r_2,\ldots, r_n\}$, n is the port number of M_k, and r_i (i=1,2,…, n) can be in the format of 1) P_i -> M_j:P_h($j \neq k$), which means that the port P_i of M_k is connected to the port P_h of M_j via the channel C_{kj}; 2) P_i -> env, which means that the port P_i of M_k is connected to external environment of the system.

We further denote the ports communicating with external environment as "external ports"; and others as "internal ports". Inputs/outputs on external/internal ports are "external/internal inputs/ outputs".

Definition 7. Related functions

1. $Port(IO\_symbol, i)$: returns the port index of IO_symbol in the protocol entity M_i;
2. $MI(i, j)$: returns the connected entity index of the other side for the port P_j of the entity M_i;
3. $PI(i, j)$: returns the connected port index of the other side for the port P_j of the entity M_i.

4.3 A simple real-time communication protocol

We specify a simple real-time communication protocol using MpTIOA as an example in this pa-

per. Figure 2(a) shows the MpTIOA specification of such a protocol, which is a 2p-TIOA with two ports (U and l) and two clocks (t_1 and t_2). $I_U = \{A\}$, $O_U = \{B, C\}$, $I_l = O_l = \{a,b,c\}$. The initial location is '0'. The protocol can be specified informally as follows:

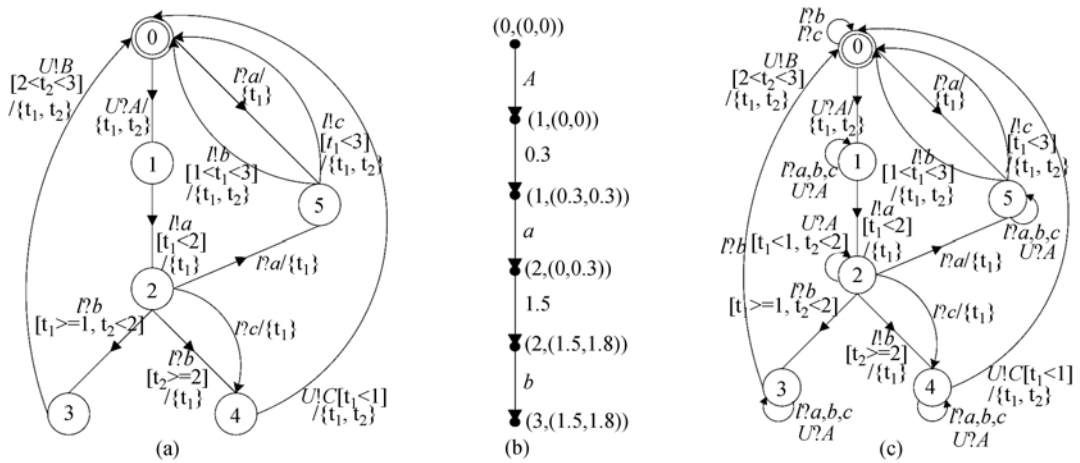

Figure 2 A simple real-time protocol specified by using MpTIOA. (a) Specification of a MpTIOA; (b) a timed trace σ; (c) an input-complete specification.

(1) Initiate a connection to a remote entity actively. If an input 'A' is received from port U in the initial location 0, the protocol entity should initiate a connection to a remote entity actively; In this transition (0,$U?A$, *true*,{t_1, t_2},1), the two local clocks t_1 and t_2 should be reset to 0. Within 2 time units, an output 'a' should be sent from port l to remote entity, and the clock t_1 should be reset to 0 (transition (1,$l!a$,[t_1<2],{t_1},2)). After that, in location 2, three cases should be considered:

a) Receiving an input 'b' from port l in time, i.e., transition (2,$l?b$,[$t_1 \geqslant 1$, t_2<2],{},3), indicates that the connection can be established.

b) Receiving an input 'b' from port l too late (long delay), i.e., transition (2,$l?b$,[$t_2 \geqslant 2$],{t_1},4), indicates that the connection cannot be established.

c) Receiving an input 'c' from port l, i.e., transition (2,$l?c$,*true*,{t_1},4), indicates that the connection cannot be established.

If the connection can be established (i.e., the current location is 3), an output 'B' should be sent to port U, i.e., transition (3,$U!B$,[2<t_2<3],{t_1,t_2},0); else, if the connection cannot be established (i.e., the current location is 4), an output 'C' should be sent to port U, i.e., transition (4,$U!C$, [t_1<1],{t_1,t_2},0).

(2) Respond a connection request from a remote entity passively. If an input 'a' is received from port l in the initial location 0, i.e., transition (0,$l?a$,*true*,{t_1},5), the protocol entity should respond to the connection request from a remote entity passively. Two cases should be considered:

a) Sending an output 'b' to port l, i.e., transition (5,$l!b$,[1<t_1<3],{t_1,t_2},0).

b) Sending an output 'c' to port l, i.e., transition (5,$l!c$,[t_1<3],{t_1,t_2},0).

Considering a timed trace $\sigma = A \cdot 0.3 \cdot a \cdot 1.5 \cdot b$, its corresponding transition sequence in TIOTS is shown in Figure 2(b). We also have s_0 after σ = {(3,(1.5,1.8))}, $out(s_0 \text{ after } \sigma)$=(0.2,1.2) and $out(s_0 \text{ after } \sigma \cdot 0.2) = \{B\} \cup (0,1)$. Figure 2(c) shows an input-complete specification after

adding self-loops to Figure 2(a).

Figure 3 shows an example of a system under test specified by CMpTIOA, which is a real-time communication system containing two real-time protocol entities. M={M_1, M_2}, Ch={C_{12}, C_{21}}. The specifications of M_1 and M_2 are both the MpTIOA of Figure 2(c). We use subscript 1, 2 on ports and actions to distinguish them. Port connection relations are (R_1, R_2), where R_1={U_1->env, l_1->M_2:l_2}, R_2={U_2->env, l_2->M_1:l_1}.

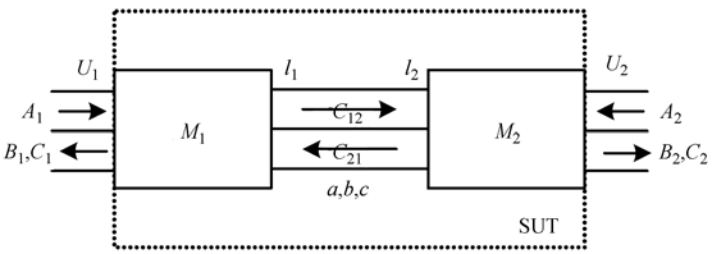

Figure 3 An example of CMpTIOA.

5 Timed interoperability relation

Before discussing interoperability relation, at first, we assume that all channels between protocol entities are reliable, i.e., lossless and non-delayed, and FIFO (first in first out) queues. In interoperability testing studied in this paper, system under test should be isolated and tested in a "clean" and separate environment. Thus, in this situation, if transfer time of I/O symbols in communicating channels can be neglected compared with time constraints in protocols, it is reasonable to assume that channels are lossless and non-delayed. For example, when the lower layer network of protocol entities is Ethernet, in the same LAN (local area network), transfer time of I/O symbols between protocol entities is in the level of milliseconds (ms); thus, if time constraints in protocols are in the level of seconds (s), the above test assumption can be considered reasonable.

Under this assumption, the transfer time of I/O symbols in channels can be approximatively set to 0. When an output symbol occurs on one protocol entity, it will be transferred to the other side immediately via the channel and become an input symbol of the other entity. So communication between the two protocol entities can be considered as synchronous communication. Based on the TIOTS semantics of MpTIOA, the synchronous composition of CMpTIOA is defined as follows.

Definition 8. The synchronous composition of CMpTIOA

In a system of CMpTIOA (M, Ch), the corresponding TIOTS of each MpTIOA is A_i=$(S^i, s_0^i, Act^i, \rightarrow^i)$ (Act^i=$I^i \cup O^i$). The synchronous composition of CMpTIOA (M,Ch) is a TIOTS $(S, s_0, Act, \rightarrow)$, where,

(1) The set of states is $S \subseteq S^1 \times S^2 \times \cdots \times S^m$. The initial state is $s_0 = (s_0^1, s_0^2, \cdots, s_0^m)$.

(2) The set of actions is Act=$Act^1 \cup Act^2 \cup \ldots \cup Act^m$.

(3) The set of transition $\rightarrow$ is defined as follows:

Rule 1. External actions: a is an external action of M_i

$$\frac{(s_i, a, s_i') \in \rightarrow^i}{(s, a, s') \in \rightarrow, s' = s(s_i / s_i')} (a \in Act^i, MI(i, Port(a, i)) = env),$$

where, $s(s_i / s_i')$ represents that in the global state s, the local state s_i is changed to s_i', and other local states keep unchanged.

Rule 2. Internal actions: a is the output action of M_i, and a is also the input action of M_j at the same time, that is, a is in the channel C_{ij}

$$\frac{(s_i,a,s_i')\in\rightarrow^i \wedge (s_j,a,s_j')\in\rightarrow^j}{(s,a,s')\in\rightarrow, s'=s(s_i/s_i',s_j/s_j')}(a\in O^i, a\in I^j, MI(i,Port(a,i))=j),$$

where, $s(s_i / s_i', s_j / s_j')$ represents that in the global state s, the local state s_i is changed to s_i', the local state s_j is changed to s_j', and other local states keep unchanged.

Rule 3. Time delay:

$$\frac{(s_i,d,s_i')\in\rightarrow^i\ (i=1,2,\cdots,m)}{(s,d,s')\in\rightarrow, s'=(s_1',s_2',\cdots,s_m')}\ (d\in \boldsymbol{R}^+).$$

The interoperability relation describes the conditions to be satisfied by the implementations that constitute the system under test. If the interoperability relation is satisfied, we consider that components of the system can be interoperable with each other. We define timed interoperability relations based on CMpTIOA model as follows. In this paper, similar to ref. [3], only specification-based interoperability relations will be considered. Firstly, we give the definition of unilateral timed interoperability relation. Intuitively, unilateral timed interoperability relation indicates that in the process of interactions between protocol entities of system under test, projection of the observable behaviors on one protocol entity is foreseen by the specification of this protocol entity. According to the intuitive meaning, we use the semantics of timed traces to define such an interoperability relation. Let the specifications of protocol entities in a system of CMpTIOA be S_1, S_2,…,S_m respectively, and their corresponding implementations be I_1, I_2,…,I_m. We denote the synchronous composition of I_1, I_2,…,I_m as $Com(I_1,I_2,\ldots,I_m)$.

Definition 9. Unilateral timed interoperability relation (for I_i)

$$\boldsymbol{R}(I_i,(I_1,I_2,\cdots,I_{i-1},I_{i+1},\cdots,I_m)) =_{\text{def}} \forall\sigma_i\in ttraces(S_i), \forall\sigma\in ttraces(Com(I_1,I_2,\cdots,I_m)),$$
$$\sigma / Act^{I_i}=\sigma_i \Rightarrow out_{Act^{I_i}}(Com(I_1,I_2,\cdots,I_m) \text{ after } \sigma)\subseteq out(S_i \text{ after } \sigma_i),$$

where Act^{I_i} represents the set of I/O action symbols of protocol implementation I_i. This relation means that during the interactions of implementations in system, after any corresponding traces in S_i, the observable output behaviors of implementation I_i must be foreseen by its specification S_i. If this relation is satisfied, we say that the implementation I_i can be interoperable with other implementations.

To consider the outputs of each port in I_i, this relation can also be denoted as follows.

Definition 10. Unilateral timed interoperability relation (for I_i)—considering ports

$$\boldsymbol{R}(I_i,(I_1,I_2,\cdots,I_{i-1},I_{i+1},\cdots,I_m)) =_{\text{def}} \forall\sigma_i\in ttraces(S_i), \forall\sigma\in ttraces(Com(I_1,I_2,\cdots,I_m)),$$
$$\sigma / Act^{I_i}=\sigma_i \Rightarrow \bigwedge_{j=1}^{n}(out_{O_j^{I_i}}(Com(I_1,I_2,\cdots,I_m) \text{ after } \sigma)\subseteq out_{O_j^{S_i}}(S_i \text{ after } \sigma_i)),$$

In this definition, n is the port number of S_i, $O_j^{I_i}$ ($O_j^{S_i}$) is the output set of $I_i(S_i)$'s port P_j. This means that the possible outputs on each port must be foreseen by its specification. In some test architecture, some ports are possible to be inaccessible. For such test architecture, we can only

check the observable behaviors on the accessible ports for judging interoperability.

For a system under test, all implementations should be interoperable with others. So a full timed interoperability relation can be defined as follows. This relation means that in the process of interactions, each implementation's behaviors must satisfy its corresponding unilateral timed interoperability relation.

Definition 11. Timed interoperability relation **tinterop**

$$\textbf{tinterop}(I_1, I_2, \cdots, I_m) =_{\text{def}} \bigwedge_{i=1}^{m} \boldsymbol{R}(I_i, (I_1, I_2, \cdots, I_{i-1}, I_{i+1}, \cdots, I_m)).$$

6 Test generation

Borrowing the ideas of conformance testing generation method based on region graph[25] (a widely-used technique to analyze real-time systems), real-time interoperability test generation method based on region graph can be proposed. However, this method will lead to state space explosion problem. To solve this problem, this paper proposes an interoperability test generation method based on timed interoperability relation. The starting point of interoperability test generation is the specifications of system under test. According to the definition of timed interoperability relation, the purpose of test generation is to find all possible timed traces on system under test to check whether protocol implementations satisfy the conditions of timed interoperability relation. But due to the continuous model of time, the number of possible timed traces is uncountable. In this paper, our strategy is to represent a subclass of equivalent timed traces as a typical parameterized timed trace. In addition, in order to prevent the unnecessary constructions of inexecutable test cases, we also introduce a mechanism called *Executability Pre-Determination*: after calculating timing conditions of parameterized timed traces, prune the *must-inexecutable* branches in advance. With these methods, state space explosion problem can be alleviated to some extent.

6.1 Algorithm

A test case of real-time protocol interoperability testing is a parameterized test behavior tree. To generate interoperability testing, we start from the initial global state of system GS_0= $(s_0^1, s_0^2, ..., s_0^m)$ to construct the test behavior tree. Leaf nodes of a test behavior tree are the verdict "pass" or "fail". For "fail" verdict, it is also necessary to indicate in which implementation the fault is located. Other nodes represent the tester's knowledge of the SUT's current global states. We define parameterized timed test behavior tree as follows.

Definition 12. Parameterized timed test behavior tree

A parameterized timed test behavior tree can be described as a behavior notation:

$$\text{B} =_{\text{def}} \text{pass} \mid \text{fail} \mid (\alpha,d)[\text{T}(d)];\text{B} \mid \text{B+B} \mid \sum\text{B}.$$

In a parameterized timed test behavior tree, each edge is labeled as a test event $(\alpha,d)[\text{T}(d)]$, in which, α is an input/output action symbol, d is a parameter, which means the relative time interval between this event and its last test event, $\text{T}(d)$ is time constraints of parameters.

Figure 4 shows a typical procedure of constructing a test behavior tree, which extends tests from a global state GS (the reachable global state from GS_0 after a timed trace σ). From it, behaviors of the tester can be 1) observing outputs; 2) applying an external input to SUT; 3) terminating the tests and setting a verdict.

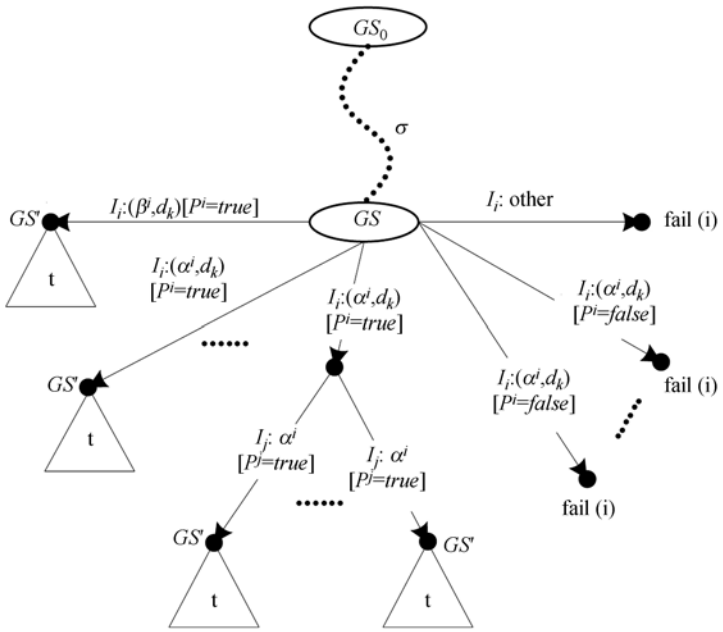

Figure 4 Procedure of constructing a parameterized timed test behavior tree.

The test generation algorithm is shown as follows.

Algorithm 1. Timed interoperability test generation

Input: a system specification described by a CMpTIOA $\{S_1, S_2,\ldots, S_m\}$, $S_i=(L^i, Act^i, l_0^i, C^i, T^i)$ (i=1,2,…,m), where, $Act^i=I^i \cup O^i$.

Output: Parameterized timed test behavior trees t_{GS_0}.

Initial values: current global state $GS=GS_0=(s_0^1,...,s_0^i,...,s_0^m)$. Trace time constraints $Cond(GS_0) = true$.

For current global state $GS=(s^1,...,s^i,...,s^m)$, where $s^i=(l^i,\boldsymbol{v}^i)$ (i =1,2,…,m), apply the following three rules recursively (k is the recursive level):

1. Observing outputs

Consider the implementation I_i, for $\forall \alpha^i \in O^i$:

$$\sum\{I_i:(\alpha^i,d_k)[P^i(\boldsymbol{v}^i+\boldsymbol{d}_k)=true];t_{GS'} \mid \exists(l^i,\alpha^i,P^i,R^i,l'^i)\in T^i, MI(i,Port(\alpha^i,i))=env;$$
$$GS'=UpdateS_i^{(1)}(GS), Cond(GS')=Cond(GS)\wedge P^i(\boldsymbol{v}^i+\boldsymbol{d}_k)\}$$

$$t_{GS} ::= \sum\{I_i:(\alpha^i,d_k)[P^i(\boldsymbol{v}^i+\boldsymbol{d}_k)=false];fail \mid \exists(l^i,\alpha^i,P^i,R^i,l'^i)\in T^i\}$$
$$+\sum\{I_i:(\alpha^i,d_k);fail \mid \neg\exists(l^i,\alpha^i,P^i,R^i,l'^i)\in T^i\}$$
$$+\sum\{I_i:(\alpha^i,d_k)[P^i(\boldsymbol{v}^i+\boldsymbol{d}_k)=true];I_j:(\alpha^i,0)[P^j(\boldsymbol{v}^j+\boldsymbol{d}_k)=true];t_{GS'} \mid$$
$$\exists(l^i,\alpha^i,P^i,R^i,l'^i)\in T^i,\ j=\mathrm{MI}(i,Port(\alpha^i,i))\neq env \text{ and } \alpha^i\in I^j,$$
$$\exists(l^j,\alpha^i,P^j,R^j,l'^j)\in T^j;\ GS'=UpdateS_{i,j}^{(2)}(GS),$$
$$Cond(GS')=Cond(GS)\wedge P^i(\boldsymbol{v}^i+\boldsymbol{d}_k)\wedge P^j(\boldsymbol{v}^j+\boldsymbol{d}_k)\}.$$

Here, d_k is uncontrollable parameters, and k is the recursive level.

2. Applying an external input to SUT.

Consider the implementation I_i, for $\forall \beta^i \in I^i$ and β^i is an external input:

$$t_{GS} := \sum \{I_i : (\beta^i, d_k)[P^i(\boldsymbol{v}^i + \boldsymbol{d}_k) = true]; t_{GS'} \mid \exists (l^i, \beta^i, P^i, R^i, l'^i) \in T^i, MI(i, Port(\beta^i, i)) = env;$$
$$GS' = UpdateS_i^{(1)}(GS),\ Cond(GS') = Cond(GS) \wedge P^i(\boldsymbol{v}^i + \boldsymbol{d}_k)\}.$$

Here, d_k is controllable parameters.

In the cases 1 and 2, if $Cond(GS')$ is evaluated to be *false*, the corresponding branch should be pruned from the test behavior tree immediately.

3. Stop testing at any time and set pass verdict.

t_{GS} := pass

In the above three rules, the global state after extending GS is GS'. We denote GS' as $(s'^1, ..., s'^i, ..., s'^m)$. The global state updating functions are defined as follows:

$$GS' = UpdateS_i^{(1)}(GS) =_{\text{def}} \begin{cases} s'^i = (l'^i, Update_{\boldsymbol{R}^i}(\boldsymbol{v}^i + \boldsymbol{d}_k)), \\ s'^h = (l^h, \boldsymbol{v}^h + \boldsymbol{d}_k)\ (h \neq i). \end{cases} \tag{1}$$

$$GS' = UpdateS_{i,j}^{(2)}(GS) =_{\text{def}} \begin{cases} s'^i = (l'^i, Update_{\boldsymbol{R}^i}(\boldsymbol{v}^i + \boldsymbol{d}_k)), \\ s'^j = (l'^j, Update_{\boldsymbol{R}^j}(\boldsymbol{v}^j + \boldsymbol{d}_k)), \\ s'^h = (l^h, \boldsymbol{v}^h + \boldsymbol{d}_k)\ (h \neq i, j). \end{cases} \tag{2}$$

We illustrate Algorithm 1 in detail. Consider the three cases as follows:

Case 1. Observing outputs. We consider the implementation I_i. When an output action α^i is observed after d_k time units, we label the corresponding branch from GS as (α^i, d_k).

Theorem 1. From global state $GS=(s^1, s^2, \ldots, s^m)(s^i = (l^i, \boldsymbol{v}^i))$, an output action α^i is observed after d_k time units $((\alpha^i, d_k))$. Then,

1) If $\exists (l^i, \alpha^i, P^i, R^i, l'^i) \in T^i$ (i.e., S_i has a transition with the output action α^i from location l^i) and $P^i(\boldsymbol{v}^i + \boldsymbol{d}_k) = true$, then the observation (α^i, d_k) will not violate **tinterop** relation defined in Definition 11, that is, (α^i, d_k) is a valid output.

2) Else, **tinterop** relation will be violated, that is, (α^i, d_k) is an invalid output.

Proof. See Appendix A.

According to Theorem 1, a set of equivalent timed traces can be represented as a typical parameterized timed trace (α^i, d_k). Whether timed traces are equivalent depends on whether timed traces satisfy timed interoperability relation. So we can extend the test behavior tree from GS according to the specification $S_i(i=1,2,\ldots,m)$ as shown in Figure 4: For each transition from l^i: $(l^i, \alpha^i, P^i, R^i, l'^i) \in T^i$, from GS, add a branch of the valid output I_i: $(\alpha^i, d_k)[P^i(\boldsymbol{v}^i + \boldsymbol{d}_k) = true]$; and for all transitions with the same output α^i, add a branch of the invalid output I_i: $(\alpha^i, d_k)[\wedge_j P_j^i(\boldsymbol{v}^i + \boldsymbol{d}_k) = false]$, where $\wedge_j P_j^i(\boldsymbol{v}^i + \boldsymbol{d}_k) = false$ means that time constraints of all transitions with the output α^i are *false*. Moreover, a branch of the invalid output "other" should also be added. For branches of not violating tinterop relation, ending nodes GS' can be calculated by

function $UpdateS_i^{(1)}$. For branches of violating tinterop relation, ending nodes are labeled as verdict "fail(i)".

After adding branches of satisfying tinterop, a further step should be taken: if the output action α^i is an internal output ($j = MI(i, Port(\alpha^i, i)) \neq env$), it will be transferred to be the input of another entity I_j. Since we have made an assumption of lossless and non-delayed channels, the transferring time can be set to 0. For each transition with the input action α^i from location l^j of S_j: $(l^j, \alpha^i, P^j, R^j, l'^j) \in T^j$, we add a branch labeled as I_j: $(\alpha^i, 0)[P^j(\boldsymbol{v}^j + \boldsymbol{d}_k) = true]$. Ending node GS' of such a branch can be calculated by function $UpdateS_{i,j}^{(2)}$. The above calculating procedure corresponds to the rule 2 of Definition 8.

From these ending nodes, we can continue to extend the test behavior tree recursively.

In Case 1, d_k are decided by implementations under test, which are so-called uncontrollable parameters. These parameters represent the relative time interval between one internal or external output action and its last I/O action. Their values will be measured in the process of test execution and they can be used to give a verdict or choose a proper branch of the test behavior tree.

Case 2. Applying an external input to SUT. When we extend the test behavior tree from GS, if there is a transition with an external input action β^i from l^i: $\exists(l^i, \beta^i, P^i, R^i, l'^i) \in T^i$, tester can decide to apply β^i to SUT after d_k time units. We add a branch labeled as I_i: (β^i, d_k) [$P^i(\boldsymbol{v}^i + \boldsymbol{d}_k)$ = *true*] in the test behavior tree. Here d_k must be satisfied with $P^i(\boldsymbol{v}^i + \boldsymbol{d}_k)$ =*true*. After execution of this branch, the ending node representing global state can be calculated by function $UpdateS_i^{(1)}$.

From these ending nodes, we can continue to extend the test behavior tree recursively.

In Case 2, the values of d_k are decided by tester, so d_k are called controllable parameters. These parameters represent the relative time interval between one external input action and its last I/O action. The values of these parameters can be randomly chosen by tester or decided by users. In each test execution, tester can choose different values of d_k.

Case 3. Terminating the tests and setting a verdict. Tester must terminate the tests at any nodes labeled with "fail" verdict. At any other nodes, tester also can terminate the tests and set a verdict "pass".

Until now, we can generate a timed test behavior tree with uncontrollable and controllable parameters. To prevent the unnecessary constructions of inexecutable branches in the test behavior tree, we also introduce the mechanism of executability pre-determination. The basic idea is that: after extending a global state node, for each branch starting from it, we calculate timing conditions of the path from root node to the branch; if the time constraints have no solutions in any cases (the trace is *must-inexecutable*), then prune the branch from the test behavior tree immediately. Such a mechanism prevents the unnecessary generation of *must-inexecutable* traces.

To achieve executability pre-determination, Algorithm 1 records the timing conditions of the path σ from root node GS_0 to the reachable global state node GS: $Cond(GS)$. In Algorithm 1, after extending GS, the update function of timing conditions for a reached global state node GS' can be denoted as follows:

$$Cond(GS') = Cond(GS) \wedge P^i(\boldsymbol{v}^i + \boldsymbol{d}_k), \tag{3}$$

or

$$Cond(GS') = Cond(GS) \wedge P^i(\boldsymbol{v}^i + \boldsymbol{d}_k) \wedge P^j(\boldsymbol{v}^j + \boldsymbol{d}_k). \quad (4)$$

Theorem 2. Let $\sigma = d_0\alpha_0 d_1\alpha_1 \cdots d_n\alpha_n$ ($\alpha_i \in Act^{S_1} \cup Act^{S_2} \cup \cdots \cup Act^{S_m}, d_i \in \boldsymbol{R}^+$); $GS_0 = (s_0^1, s_0^2, ..., s_0^m)$, $s_0^i = (l_0^i, \boldsymbol{v}_0^i)$, $\boldsymbol{v}_0^i = (v_{10}^i, v_{20}^i, \cdots, v_{|C^i|0}^i)$ (i=1,2,…,m); GS=(s_1,s_2,…,s_m), $s^i = (l^i, \boldsymbol{v}^i)$, $\boldsymbol{v}^i = (v_1^i, v_2^i, \cdots, v_{|C^i|}^i)$ (i=1,2,…,m); GS is the global state reached from GS_0 after σ. Then

1) The clock value $v_j^i (i = 1,2,...,m, j = 1,2,...,|C^i|)$ can be represented by a linear combination of v_{j0}^i and d_k(k=0,1,2,...,n);

2) ***Cond***(GS) can be represented by a conjunction over a set of linear inequalities of $v_{j0}^i (i = 1,2,...,m, j = 1,2,...,|C^i|)$ and d_k(k=0,1,2,...,n).

Proof. See Appendix B.

Here, d_k(k=0,1,2,…,n) are just uncontrollable or controllable parameters in the generated test behavior tree. In the process of test generation, we use the mechanism of executability pre-determination to check whether timing conditions $Cond(GS')$ have a solution, which can be reduced to a linear programming problem with solution of polynomial timing according to Theorem 2. In the case of no solution, the corresponding branch should be pruned from the test behavior tree.

6.2 Example

This section applies Algorithm 1 to the system under test in Figure 3. We start from the initial global state GS_0=((0,(0,0)), (0,(0,0))) to construct test behavior tree. A part of result is shown in Figure 5. d_0 is a controllable parameter, and d_1, d_2, d_3 are uncontrollable parameters.

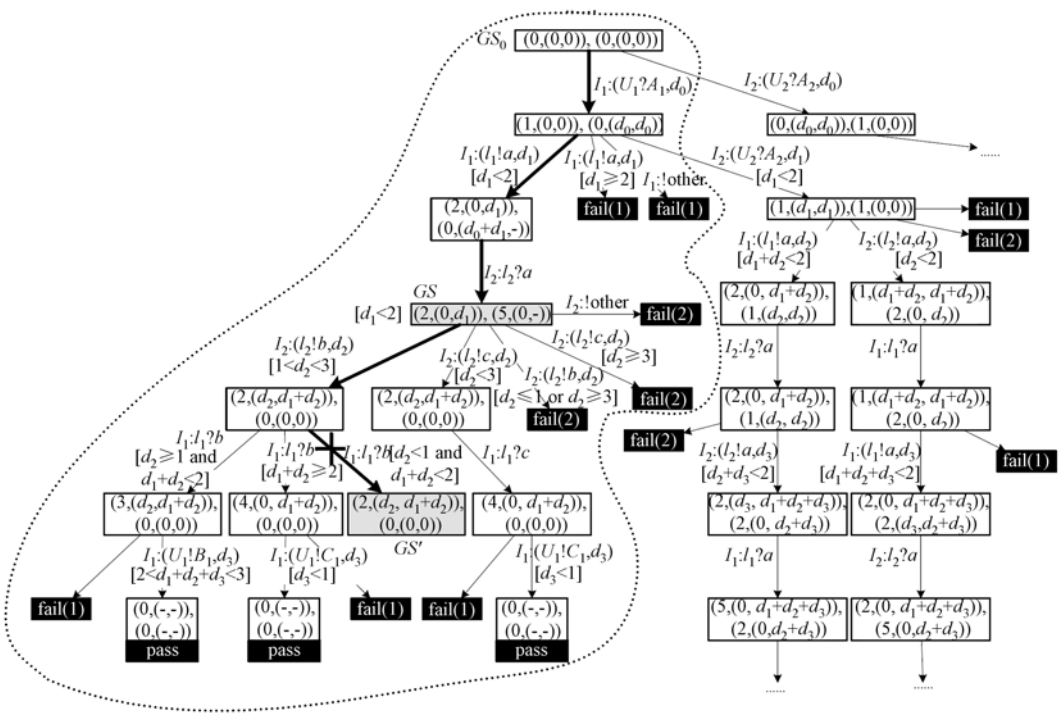

Figure 5 A part of result of test generation for the system under test in Figure 2.

For example, we consider a test sequence from the root node to node GS=((2,(0,d_1)),(5,(0,-))): I_1:(U_1?A_1,d_0); I_1:(l_1!a,d_1); I_2:(l_2?a). Here, d_0 is a controllable parameter, which can be chosen by tester, for example, 1 time unit; and d_1 is an uncontrollable parameter, which will be decided in the process of test execution. The reachable timing conditions of GS is $Cond(GS)$={d_1<2}. The corresponding timed trace is $\sigma = d_0 \cdot (I_1 : U_1?A_1) \cdot d_1 \cdot (I_1 : l_1!a) \cdot (I_2 : l_2?a)$. After σ, possible outputs of I_2 can be I_2:(l_2!b,d_2)[1<d_2<3] or I_2:(l_2!c,d_2)[d_2<3]. All the other behaviors will lead to a verdict fail(2), which indicates that an error occurs in I_2.

The branch from GS to GS'=((2,(d_2,d_1+d_2)),(0,(d_2,-)) is {I_2:(l_2!b,d_2)[1<d_2<3]; I_1:(l_1?b,0)[d_2<1 and d_1+d_2<2]}.So

$$\begin{aligned} Cond(GS') &= Cond(GS) \wedge (1 < d_2 < 3) \wedge (d_2 < 1) \wedge (d_1 + d_2 < 2) \\ &= \{(d_1 < 2) \wedge (1 < d_2 < 3) \wedge (d_2 < 1) \wedge (d_1 + d_2 < 2)\}. \end{aligned}$$

In the process of executability pre-determination, $Cond(GS')$ has no solutions. So this branch will be pruned from the test behavior tree in the test generation process.

6.3 Discussions and comparisons

Region graph[25] is a widely-used technique to analyze real-time systems. In refs. [15, 16], conformance test generation methods have been presented based on region graph. In this section, firstly we give an intuitive interoperability test generation method based on region graph and then compare it with the method based on timed interoperability relation described in Algorithm 1. We describe the intuitive method informally as follows[1)]:

Step 1. Sample the region graph of each MpTIOA in the CMpTIOA system using suitable granularities to construct its grid automaton based on the method in ref. [15].

Step 2. Apply the existing non-timed interoperability testing generation method on the communicating grid automata system (the result of step 1) to generate real-time interoperability test cases.

Now we discuss the state space scale generated by this method. Considering the case of more than one clock, for an MpTIOA (see Definition 1) containing $|C|$($|C|$>1) clocks, the clock sampling granularity is 1/($|C|$+2)[15]. Let the maximal clock value of clock t_i is D_i(i=1, 2,…,$|C|$), then the total number of clock sampling granularity values is D_i*($|C|$+2)+2 (including 0 and +∞). Thus, the maximal number of clock valuations of the $|C|$ clocks is $\prod_{i=1}^{|C|}(D_i*(|C|+2)+2)$, and the maximal number of states of the grid automaton is $|L|*\prod_{i=1}^{|C|}(D_i*(|C|+2)+2)$, i.e., O($|L|$*$|C|^{|C|}$), where, $|L|$ is the number of locations in the MpTIOA (see Definition 1). For a CMpTIOA system (see Definition 5) containing m MpTIOA, let the number of locations of MpTIOA M_j be $|L_j|$(j=1,2,…,m), and the number of its clocks be $|C_j|$, then the maximal state space of CMpTIOA system can be reached to O($\prod_{j=1}^{m}|L_j|*|C_j|^{|C_j|}$). If the system contains more MpTIOA and MpTIOAs contain more locations and clocks, state space explosion problem will be more critical. While in Algorithm 1, via parameterized test behavior tree and parameterized global state nodes in the tree, state space of system can be compressed to some extent. For example, in the test generation result in Figure 5, for global state node GS=((2,(0, d_1)),(5,(0,-))), its time constraint is $Cond(GS)$={d_1<2}. The clock sampling granularity is 1/4 because the MpTIOA

1) Due to the limitation of space, only outline of the method is given in this paper, and details can be found in ref. [27].

contains 2 clocks, so according to *Cond*(*GS*), the global state *GS* contains 8 states in the corresponding communicating grid automata.

We apply the above method based on region graph (steps 1 and 2) to the system under test of Figure 3. In step 1, granularity 1/4 is used to sample each MpTIOA (because the automaton has two clocks). After reduction by combining some equivalent states, the resulting grid automaton has about 191 states. In step 2, as a comparison, we generate a part of test behavior tree that is equivalent to the parameterized test behavior tree in the dashed circle of Figure 5. The resulting test behavior tree contains about 155 nodes (representing global states) and 378 edges (representing test events, containing 225 I/O actions and 153 delay events). As a comparison, the equivalent parameterized timed test behavior tree generated by using Algorithm 1 contains only 21 nodes (including verdict nodes) and 20 edges. From the comparison, we can conclude that the intuitive method based on region graph given in this section suffers from state space explosion problem and Algorithm 1 can alleviate this problem to some extent. In fact, compared with the region graph based method, the test generation method based on timed interoperability relation is a coarse granularity analyzing method, so it can compress the global state space to some extent.

7 Applications

IPv6 protocol is one of the basic protocols of next generation Internet. This section applies the models and methods presented in this paper to interoperability testing of IPv6 neighbor discovery protocol.

7.1 Neighbor discovery protocol

Neighbor discovery (for short, ND) protocol[28] is one of IPv6 core protocols, which corresponds to a combination of ARP protocol, ICMP router discovery and ICMP redirect function in IPv4. In addition, neighbor unreachability detection mechanism is supplied to enhance the robustness of packet transmission. In ND protocol, two types of nodes exist in a link: router and host. Router is the node that forwards IP packets not explicitly addressed to itself, which is the relay node in the link; while host is any node that is not a router, which is the end node. The two types of nodes have different functions in ND protocol.

In this paper, router discovery function in ND protocol is used to illustrate our method. In such function, router nodes and host nodes play different roles: 1) for router node: it sends unsolicited router advertisements (for short, uS_RA) periodically in its interfaces; on the other hand, a router sends solicited router advertisements (for short, S_RA) in response to valid router solicitations (for short, RS) received on an advertising interface; 2) for host node: to obtain router advertisements quickly, when an interface becomes enabled, a host should transmit up to 3 router solicitation periodically. If the host receives a valid router advertisement, it must stop sending additional RS on that interface. Compared with the corresponding function in IPv4, the above function has some real-time requirements, and time constraints are more complicated. In ND protocol, requirements of time constraints are specified in detail[28]: 1) time intervals between periodic packet transmissions; and 2) delay before sending a packet.

7.2 Protocol model

Figure 6 shows the CMpTIOA model of router discovery function in ND protocol. In this model, two nodes, router and host, are modeled by an MpTIOA respectively; the channels between the

two nodes are C_{RH} and C_{HR}.

The MpTIOA model of the router node contains two clocks t_{11} and t_{12}, and its initial location is 0. MinT and MaxT represent the minimal and maximal time intervals of sending uS_RA for router nodes respectively, i.e., parameters MinRtrAdvInterval and MaxRtrAdvInterval specified in protocol. The two parameters are configurable for router nodes and MinRtrAdvInterval should be no less than 3 s (i.e., $3 \leqslant \text{MinT} < \text{MaxT}$). The MpTIOA model of the host node contains two clocks t_{21} and t_{22}, and its initial location is 0, which represents that the host interface is not attached to the link.

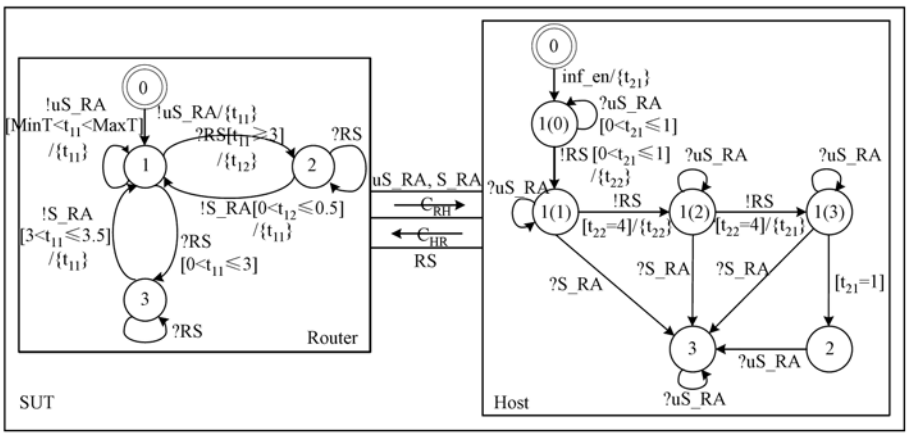

Figure 6 CMpTIOA model of router discovery function in neighbor discovery protocol.

7.3 Test generation

We apply Algorithm 1 to the CMpTIOA model of Figure 6. The generated parameterized test behavior tree is shown in Figure 7: All fail verdict nodes are omitted and gray leaf nodes represent pass verdict. Test architecture shown in Figure 8 can be used in testing: a point of observation PO is used to observe the protocol behavior on the channel C_{RH} and C_{HR}; and PCO1 is used to control the host node, e.g., sending interface enabling command.

In Figure 7, d_1 is controllable parameter, which can be set to an arbitrary value from 0 to positive infinity. In the testing, this value can be assigned in the test case or can be decided by test system randomly in the process of execution. Considering the need of practical testing, this value should have an upper limit. Now we analyze the value of this parameter furthermore. The solutions of set of inequalities $\{(0<d_2\leqslant 1) \wedge (d_1+d_2\geqslant 3)\}$ and $\{((0<d_2\leqslant 1) \wedge (0<d_1+d_2<3))\}$ are $d_1 \geqslant 2$ and $d_1<3$ respectively. Thus the time axis of d_1 can be divided into three areas: 1) $d_1<2$, in which case only the right branch of Figure 7 can be executable; 2) $2\leqslant d_1<3$, in which case both the left and right branches are executable; 3) $d_1\geqslant 3$, in which case only the left branch can be executable. So in practical testing, different representative d_1 values in these three areas can be selected to improve test coverage.

8 Conclusions

In this paper, we have proposed a formal method to real-time protocol interoperability testing. Firstly, a formal model CMpTIOA is defined to model SUT of real-time protocol interoperability testing. Based on the model, we define the timed interoperability relation as the testing basis. A

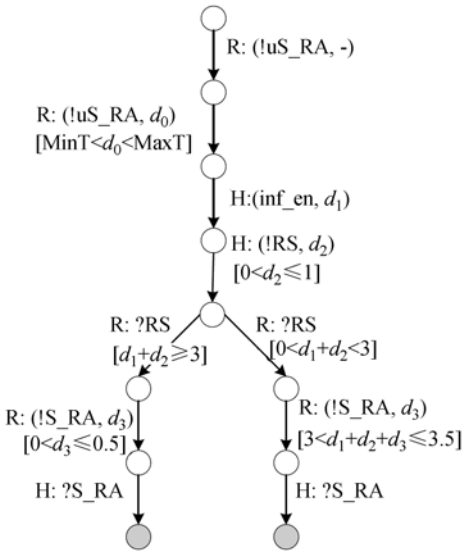

Figure 7 Test generation result: test behavior tree.

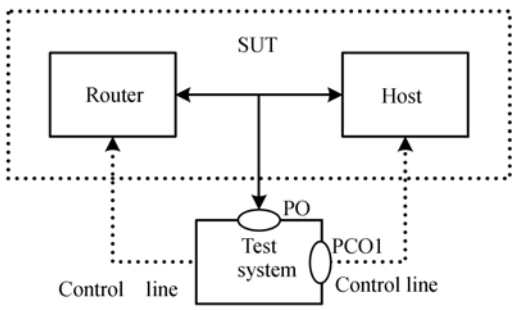

Figure 8 Test architecture.

test generation method based on timed interoperability relation is then proposed. This method starts from the CMpTIOA model of SUT to construct parameterized timed test behavior tree, in which parameters are used to represent the relative time intervals between I/O behaviors and a subclass of equivalent timed traces is represented as a typical parameterized timed trace. A mechanism of executability pre-determination is also integrated in the test generation method to prevent the unnecessary constructions of inexecutable branches in the test behavior tree. Thus, this method can alleviate the state space explosion problems to some extent compared with the method based on region graph. At last, the proposed theory and method are then applied in interoperability testing of IPv6 neighbor discovery protocol, which shows the feasibility of this method.

Test notation is also an important issue in the area of formal testing. Its main purpose is to solve the problem of how to specify test cases by using abstract test suite notations. TTCN-3 (the Testing and Test Control Notation)[29] is a test description language proposed by ETSI (European Telecommunications Standards Institute), and *TIMED*TTCN-3[30] is a real-time extension of TTCN-3. The parameterized test behavior tree can be converted to *TIMED*TTCN-3 test cases easily using the method in ref. [31]. Based on the theory and method proposed in this paper, we have implemented a protocol testing system based on *TIMED*TTCN-3.

The possible future work is to extend the test framework to consider the data portion of protocol and the unreliable communicating channels. The other directions are to study how to select proper values of controllable parameters and to apply distributed test architecture to real-time protocol interoperability testing. We will also plan to apply this method to protocol testing of real-life network protocols furthermore.

Appendix A Proof of Theorem 1

Proof. Let GS be the reachable global state from GS_0 after a timed trace σ, i.e., $\sigma \in ttraces(Com(I_1, I_2, \cdots, I_m))$, and $GS \in GS_0$ after σ. If an output action α^i is observed after d_k time units, then

$$\begin{cases} d_k \in out(GS) \\ \alpha^i \in out(GS \text{ after } d_k) \end{cases}, \text{ i.e., } \begin{cases} d_k \in out_{Act^{I_i}}(GS) \\ \alpha^i \in out_{Act^{I_i}}(GS \text{ after } d_k) \end{cases},$$

thus,

$$\begin{cases} d_k \in out_{Act^{I_i}}(Com(I_1, I_2, \cdots, I_m) \text{ after } \sigma), \\ \alpha^i \in out_{Act^{I_i}}(Com(I_1, I_2, \cdots, I_m) \text{ after } \sigma \cdot d_k). \end{cases} \tag{A1}$$

For $\sigma_i = \sigma / Act^{I_i}$, $s^i \in s_0^i$ after σ_i, i.e., $s^i \in S_i$ after σ_i;

1) If $\exists(l^i, \alpha^i, P^i, R^i, l'^i) \in T^i$ and $P^i(\boldsymbol{v}^i + \boldsymbol{d}_k) = true$, we have $\begin{cases} d_k \in out(s^i) \\ \alpha^i \in out(s^i \text{ after } d_k) \end{cases}$. So,

$$\begin{cases} d_k \in out(S_i \text{ after } \sigma_i), \\ \alpha^i \in out(S_i \text{ after } \sigma_i \cdot d_k). \end{cases} \tag{A2}$$

According to (A1) and (A2), the observed behavior (α^i, d_k) will not violate unilateral timed interoperability relation defined in Definition 9, so it will not violate tinterop relation defined in Definition 11, that is, (α^i, d_k) is a valid output.

2) Consider two cases as follows:

a) For $\forall(l^i, \alpha^i, P^i, R^i, l'^i) \in T^i$, always $P^i(\boldsymbol{v}^i + \boldsymbol{d}_k) = false$:

If $d_k \in out(s^i) \subseteq out(S_i \text{ after } \sigma_i)$, while $P^i(\boldsymbol{v}^i + \boldsymbol{d}_k) = false$, according to the semantics of MpTIOA model, then $\alpha^i \notin out(s^i \text{ after } d_k)$, so

$$\alpha^i \notin out(S_i \text{ after } \sigma_i \cdot d_k). \tag{A3}$$

According to (A1) and (A3),

$$out_{Act^{I_i}}(Com(I_1, I_2, \cdots, I_m) \text{ after } \sigma \cdot d_k) \not\subseteq out(S_i \text{ after } \sigma_i \cdot d_k). \tag{A4}$$

And we also have $\sigma \cdot d_k / Act^{I_i} = \sigma_i \cdot d_k$, so (A4) indicates that in this case, unilateral timed interoperability relation defined in Definition 9 will not be satisfied, and tinterop relation will not be satisfied yet, that is, (α^i, d_k) is an invalid output.

b) If $\neg\exists(l^i, \alpha^i, P^i, R^i, l'^i) \in T^i$:

According to the semantics of MpTIOA model, for $\forall d_k \in \boldsymbol{R}^+$, we have $\alpha^i \notin out(s^i \text{ after } d_k)$. So (A3) and (A4) are satisfied, which indicates tinterop relation will not be satisfied, that is, (α^i, d_k) is an invalid output.

Appendix B Proof of Theorem 2

Proof. Using mathematical induction method:

1) (1) For GS_0, the clock value $v_j^i = v_{j0}^i (i = 1, 2, ..., m, j = 1, 2, ..., |C^i|)$, so the statement is true.

(2) Suppose that the statement is true for GS, that is, $v_j^i (i = 1, 2, ..., m, j = 1, 2, ..., |C^i|)$ can be represented by a linear combination of v_{j0}^i and d_k (k=0,1,2,…,n). After extension of GS in

test generation, we get GS'. According to Algorithm 1, $GS' = UpdateS_i^{(1)}(GS)$ or $GS' = UpdateS_{i,j}^{(2)}(GS)$. By eqs. (1) and (2) in section 6.1, the possible clock valuation in GS' can be

- $(\boldsymbol{v}^i)' = Update_R(\boldsymbol{v}^i + \boldsymbol{d}_{n+1})$ (i=1,2,…,m): according to Definition 2 (clock valuation updating function),

$$(v_j^i)' = \begin{cases} v_j^i + d_{n+1} (t_j^i \notin R) \\ 0 (t_j^i \in R) \end{cases} \quad (i = 1,2,...,m, j = 1,2,...,|C^i|).$$

- $(\boldsymbol{v}^i)' = \boldsymbol{v}^i + \boldsymbol{d}_{n+1}$ (i=1,2,…,m): $(v_j^i)' = v_j^i + d_{n+1}$ $(i = 1,2,...,m, j = 1,2,...,|C^i|)$.

So $(v_j^i)'$ can be represented by a linear combination of v_{j0}^i and d_k(k=0,1,2,…,n+1).

According to (1) and (2), the statement is true.

2) (1) For GS_0, according to the initial value of Algorithm 1, we have $Cond(GS_0)$=*true*; so the statement is true.

(2) Suppose that the statement is true for GS, that is, $Cond(GS)$ can be represented by a conjunction over a set of linear inequalities of $v_{j0}^i (i = 1,2,...,m, j = 1,2,...,|C^i|)$ and d_k(k=0,1,2,...,n). According to eqs. (3) and (4) in section 6.1, we have

$$Cond(GS') = Cond(GS) \wedge P^i(\boldsymbol{v}^i + \boldsymbol{d}_{n+1})$$

or

$$Cond(GS') = Cond(GS) \wedge P^i(\boldsymbol{v}^i + \boldsymbol{d}_{n+1}) \wedge P^j(\boldsymbol{v}^j + \boldsymbol{d}_{n+1}).$$

According to Definition 1 (MpTIOA), time constraints $P(\boldsymbol{v})$ of a transition is a conjunction over linear inequalities of $v_j (j = 1,2,...,|C|)$. So $P^i(\boldsymbol{v}^i + \boldsymbol{d}_{n+1})$ is a conjunction over linear inequalities of $v_j^i + d_{n+1} (j = 1,2,...,|C^i|)$. From the conclusion of 1), v_j^i can be represented by a linear combination of v_{j0}^i and d_k (k=0,1,2,...,n). Thus $Cond(GS')$ can be represented by a conjunction over a set of linear inequalities of $v_{j0}^i (i = 1,2,...,m, j = 1,2,...,|C^i|)$ and d_k (k=0,1,2,…,n+1).

According to (1) and (2), the statement is true.

The authors thank Shi Xingang and Tian Beihang for their deep discussions.

1 ISO/IEC. ISO/IEC 9646. Information technology, open systems interconnection, conformance testing methodology and framework. Geneva, Switzerland: ISO/IEC, 1991
2 Hao R B, Wu J P. A formal approach to protocol interoperability testing. J Comput Sci Technol, 1998, 13(1): 79－90 [DOI]
3 Viho C, Barbin S, Tanguy L. Towards a formal framework for interoperability testing. In: Kim M, Chin B, Kang S, et al., eds. Proceedings of the 21st IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2001). Cheju Island, Korea: Kluwer, 2001. 53－68
4 Rafiq O, Castanet R. From conformance testing to interoperability testing. In: Davidson I, Litwack D W, eds. Proceedings of the 3rd IFIP International Workshop on Protocol Test Systems. Virginia, USA: Elsevier Science Publishers, 1990. 371－385
5 Kang S, Shin J, Kim M. Interoperability test suite derivation for communication protocols. Comput Netw, 2000, 32(3): 347－364 [DOI]
6 Trenkaev V, Kim M, Seol S. Interoperability testing based on a fault model for a system of communicating FSMs. In: Hogrefe D, Wiles A, eds. Proceedings of the 15th IFIP International Conference on Testing of Communicating Systems (TestCom 2003), Lect Notes in Comput Sci (LNCS) Vol 2644. Sophia Antipolis, France: Springer, 2003. 226－242

7 Seol S, Kim M, Kang S, et al. Fully automated interoperability test suite derivation for communication protocols. Comput Netw, 2003, 43(6): 735－759 [DOI]
8 Seol S, Kim M, Chanson S T, et al. Interoperability test generation and minimization for communication protocols based on the multiple stimuli principle. IEEE J Sel Area Comm, 2004, 22(10): 2062－2074 [DOI]
9 El-Fakih K, Trenkaev V, Spitsyna N, et al. FSM based interoperability testing methods for multi-stimuli model. In: Groz R, Hierons R M, eds. Proceedings of the 16th IFIP International Conference of Testing of Communicating Systems (TestCom 2004), Lect Notes in Comput Sci (LNCS) Vol 2978. Oxford, UK: Springer, 2004. 60－75
10 Wang Z L, Wu J P, Yin X. Protocol interoperability test generation based on communicating multi-port FSMs (in Chinese). Chinese J Comput, 2006, 29(11): 1909－1919
11 Hao R B, Lee D, Sinha R K, et al. Integrated system interoperability testing with applications to VoIP. IEEE/ACM Trans Netw, 2004, 12 (5): 823－836 [DOI]
12 Desmoulin A, Viho C. Quiescence management improves interoperability testing. In: Khendek F, Dssouli R, eds. Proceedings of the 17th IFIP International Conference of Testing of Communicating Systems (TestCom 2005). Lect Notes in Comput Sci (LNCS) Vol 3502, Montreal, Canada: Springer, 2005. 365－379
13 Desmoulin A, Viho C. Formalizing interoperability for test case generation purpose. In: Proceedings of IEEE Nasa ISoLA Workshop on Leveraging Applications of Formal Methods, Verification, and Validation. Columbia, MD, USA: IEEE, 2005
14 Desmoulin A, Viho C. A new method for interoperability test generation. In: Petrenko A, Veanes M, Tretmans J, et al., eds. Proceedings of the 19th IFIP International Conference of Testing of Communicating Systems/the 7th International Workshop on Formal Approaches to Testing of Software (TestCom/FATES 2007). Lect Notes in Comput Sci (LNCS) Vol 4581. Tallinn, Estonia: Springer, 2007. 58－73
15 En-Nouaary A, Dssouli R, Khendek F. Timed Wp-method: testing real-time systems. IEEE Trans Softw Eng, 2002, 28(11): 1023－1038 [DOI]
16 Springintveld J, Vaandrager F, D'Argenio P R. Testing timed automata. Theor Comput Sci, 2001, 254(1-2): 225－257 [DOI]
17 Higashino T, Nakata A, Taniguchi K, Cavalli A R. Generating test cases for a timed I/O automaton model. In: Csopaki G, Dibuz S, Tarnay K, eds. Proceedings of the IFIP 12th International Workshop on Testing Communicating Systems (IWTCS 1999). Budapest, Hungary: Kluwer, 1999. 197－214
18 Khoumsi A, Jéron T, Marchand H. Test cases generation for nondeterministic real-time systems. In: Petrenko A, Ulrich A, eds. Proceedings of the 3rd Workshop on Formal Approaches to Testing of Software (FATES 2003), Lect Notes in Comput Sci (LNCS) Vol 2931. Montreal, Canada: Springer, 2003. 131－146
19 Krichen M, Tripakis S. An expressive and implementable formal framework for testing real-time systems. In: Khendek F, Dssouli R, eds. Proceedings of the 17th IFIP International Conference of Testing of Communicating Systems (TestCom 2005). Lect Notes in Comput Sci (LNCS) Vol 3502. Montreal, Canada: Springer, 2005. 209－225
20 Larsen K, Mikucionis M, Nielsen B. Online testing of real-time systems using Uppaal. In: Grabowski J, Nielsen B, eds. Workshop on Formal Approaches to Testing of Software (FATES 2004), Lect Notes in Comput Sci (LNCS) Vol 3395. Linz, Austria: Springer, 2004. 79－94
21 Briones L B, Brinksma E. A test generation framework for quiescent real-time systems. In: Grabowski J, Nielsen B, eds. Workshop on Formal Approaches to Testing of Software (FATES 2004), Lect Notes in Comput Sci (LNCS) Vol 3395. Linz, Austria: Springer, 2004. 64－78
22 Wang Z L, Wu J P, Yin X. Towards interoperability test generation of time dependent protocols: a case study. In: Proceedings of IEEE Global Telecommunications Conference (GLOBECOM 2004), Vol. 2. Dallas, Texas USA: IEEE Communications Society, 2004. 589－594
23 Lee D, Yannakakis M. Principles and methods of testing finite state machines —— a survey. Proc IEEE, 1996, 84(8): 1090－1123 [DOI]
24 Tretmans J. Test generation with inputs, outputs and repetitive quiescence. Softw-Concepts and Tools, 1996, 17(3): 103－120
25 Alur R, Dill D. A theory of timed automata. Theor Comput Sci, 1994, 126(2): 183－235 [DOI]
26 Bornot S, Sifakis J, Tripakis S. Modeling urgency in timed systems. In: de Roever W P, Langmaack H, Pnueli A, eds. International Symposium of Compositionality —— The Significant Difference (COMPOS 1997), Lect Notes in Comput Sci (LNCS) Vol 1536. Malente: Springer, 1998. 103－129
27 Wang Z L. Distributed protocol interoperability testing based on formal methods (in Chinese). Ph.D. Thesis. Beijing: Tsinghua University, 2006
28 Narten T, Nordmark E, Simpson W. Neighbor Discovery for IP Version 6 (IPv6). IETF RFC 2461, 1998
29 ETSI: ETSI standard ES 201 873-1 V3.2.1(2007-03): The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France. 2007
30 Dai Z R, Grabowski J, Neukirchen H. *TIMED*TTCN-3 —— a real-time extension for TTCN-3. In: Schieferdecker I, et al, eds. Proceedings of the IFIP 14th International Conference on Testing Communicating Systems (Testcom 2002). Berlin: Kluwer, 2002. 407－424
31 Wang Z L, Wu J P, Yin X, Shi X G, Tian B H. Using *TIMED*TTCN-3 in interoperability testing for real-time communication systems. In: Uyar M U, Duale A, Fecko M, eds. Proceedings of the IFIP 18th IFIP International Conference on Testing Communicating Systems (Testcom 2006), Lect Notes in Comput Sci (LNCS) Vol 3964. New York: Springer, 2006. 324－340