Interoperability Test Generation and Minimization for Communication Protocols Based on the Multiple Stimuli Principle

Soonuk Seol, Myungchul Kim, Member, IEEE, Samuel T. Chanson, Member, IEEE, and Sungwon Kang

Abstract—This paper presents an automatic test generation and minimization method for testing interoperability of communication protocols such as the asynchronous transfer mode/broadband integrated services digital network (ATM/B-ISDN) signaling protocol and the transmission control protocol (TCP). The method is based on a technique of composing finite-state machines (FSMs). Traditionally, the generation of inputs utilizing FSM was mostly based on sequential test scripting languages and often done manually. Each input is processed on a one-at-a-time basis, which did not allow generating complex scenarios of simultaneous inputs on several interfaces of the system under test (SUT). In practice, there exists the possibility that additional inputs have to be sent to the SUT, while the previous input is still being processed, and/or multiple inputs need to be sent to different interfaces of the SUT at the same time. Our method generates interoperability test cases dealing with these more complex situations. Moreover, it minimizes the number of increased test cases due to the multiple stimuli without jeopardizing transition coverage. Experimental results have shown that compared with the conventional methods, our test generation method generates more interoperability test cases resulting in higher transition coverage (26% higher for TCP and 12% higher for ATM signaling protocol). Also, our test minimization method reduces the interoperability test cases by about 94% in the case of the ATM signaling protocol with the same transition coverage.

Index Terms—Conformance testing, interoperability testing, protocols, test case generation, test minimization, validation.

I. INTRODUCTION

To increase the confidence that protocol products conform to international standards, various protocol testing methodologies (see, for example, [5], [20], [22], and the bibliographies therein) have been developed and deployed. In particular, conformance testing that checks whether an implementation is correct with respect to the relevant standards has been standardized by ISO [1] and ITU-T [12], and applied to protocols such as narrowband integrated services digital network (N-ISDN) and asynchronous transfer mode/broadband (ATM/B-ISDN). Nevertheless, it is well known that conformance testing has limitations in ensuring interoperability. Thus, even two conforming implementations may fail to interoperate [2], [3]. The reasons include ambiguity of protocol standards, incompatible option settings, and incomplete conformance testing [4]. Moreover, in interoperability testing, the degree of interoperation between implementations depends not only on the implementations themselves but also on the environment and, hence, some degree of direct testing of interoperation is considered indispensable [8], [21].

In the field of interoperability testing, most work has concentrated on generating interoperability test cases. A common approach is to select test cases from the composed finite-state machine (FSM), which is constructed from component machines of the system via reachability analysis. Rafiq and Castanet [2] employed this approach to generate interoperability test cases. However, a rigorous definition of interoperability was not used and the work considered only the case where lower testers exist between two implementations under test (IUTs). Arakawa et al. [3] derived a conformance test suite and an interoperability test suite separately, and then manually combined them to reduce the number of conformance test cases. Kang and Kim [6], [8] developed a coherent framework of interoperability testing and a systematic interoperability test suite derivation method based on the framework. Griffeth et al. [21] presented a general method for automatic generation of test cases with various redundancy criteria and applied it to voice-over-Internet protocol (VoIP) systems. For testing the data part of protocols, Seol et al. [18] developed an automatic test generation method, which assigns values to parameterized test cases based on experimental design techniques.

In fact, the approach used in interoperability testing can also be found in some other work in which the notion of interoperability testing is not explicitly mentioned. These include validation of interacting processes modeled by labeled transition systems [15], conformance testing based on communicating FSMs [10], [19], and embedded testing [11].

In the previous studies, however, the single stimulus principle has always been explicitly or implicitly assumed either in the method, model, or architecture. According to the single stimulus principle, only a single stimulus is examined at each stable state when generating a reachability graph. A stable state for concurrent systems or communicating protocols is defined in [9] as the system state 1) that is reachable from the initial state adhering to the single stimulus principle and 2) from which no change can occur without another stimulus. Systems adhering to the single stimulus principle are said to be running in a slow environment [10]. In a slow environment, inputs can be sent from the environment to the system only when all the queues and all the channels
are empty. Therefore, simultaneous stimuli and stimuli during transitions cannot be considered.

When protocol implementations interact with one another, it is possible that additional messages are sent to the implementation(s), while the previous message is still being processed, and/or simultaneous messages are sent to each implementation at the same time. This is known as multiple stimuli. For instance, TCP allows simultaneous open and simultaneous close of a TCP connection. Therefore, checking multiple stimuli in interoperability testing is important because multiple stimuli exist in many protocol behaviors and their implementation seems to be especially error-prone.

Reducing the size of the interoperability test cases is another important concern. In order to reduce the size of relevant state space, the notion of stable state has been introduced in the literature [9], [10]. Unfortunately, as mentioned above, these studies have assumed the single stimulus principle. An alternative is to use a guided random walk approach [19] to trim the state space at the cost of exploring less behavior space. Various redundancy criteria are discussed in the literature [21]. The existing work, however, is not capable of handling complex scenarios of simultaneous inputs on several interfaces of the system under test (SUT).

In this paper, we present a new method that supports concurrent protocol behavior in generating test cases for interoperability testing. Our scheme also minimizes the size of the test cases in order to reduce testing time and cost. The rest of this paper is organized as follows. Section II presents some related work regarding the limitation of the single stimulus principle. In Section III, we define formal models for communication protocols and interoperability test cases. Section IV describes our approach to interoperability testing and the procedure for generating interoperability test suites with applications to the connection establishment phase of TCP and the ATM/B-ISDN signaling protocol. Section V presents test case minimization algorithms and their applications to the interoperability test cases derived from the ATM signaling protocol. Finally, Section VI concludes the paper.

II. RELATED WORK AND MOTIVATION

In this section, we present some existing test generation methods that are closely related to our work and discuss their limitations due to the single stimulus assumption.

Luo et al. [10] proposed a method of generating test sequences for concurrent programs and communication protocols modeled as communicating nondeterministic finite-state machines (CNFSMs). The method first reduces a system of CNFMSMs into a single NFSM by reachability analysis. The NFSM is then transformed into a trace-equivalent observable NFSM from which test sequences are generated. Basically, a global state denotes contents of channels, input queues, and states in each machine. It is obvious that the number of states in a global state machine becomes infinite in case of unlimited channels and/or queues. For that reason, they made the assumptions that queues and channels are bounded, and that the system runs in a slow environment. The first assumption is reasonable since unbounded queues or unbounded channels cannot be implemented in practice. The second assumption, however, is not reasonable for those protocols that allow multiple and simultaneous messages.

Lee et al. [19] used a guided random walk procedure for conformance testing of protocols specified as communicating FSMs. This procedure attempts to cover all edges of the component FSMs instead of all edges in the global FSM. It uses synchronous interprocess input/output operations to specify interactions between machines. For a message transmission, the input and output operations have to be matched and executed simultaneously. In other words, if a process attempts to send an output, it has to wait until the receiving process is ready to execute the matching input operation and vice versa. This means that the system adheres to the single stimulus principle and, hence, multiple stimuli during transitions between stable states and simultaneous inputs cannot be examined.

Lima and Cavalli [11] proposed a pragmatic approach to generating test sequences for embedded components of complex systems. Embedded testing and interoperability testing are similar in that internal actions are usually invisible and, thus, hard to control. Their method provides means to support multiple outputs (i.e., a signal from the environment brings about several simultaneous outputs) and nondeterminism of components. A drawback of their method is that it limits the number of external inputs between stable states to one. In other words, an input can be given to the system only if the previous input has been completely processed and the system is not undergoing any state change at the time.

In practice, multiple messages between communicating entities often occur. Fig. 1 shows three cases of multiple messages in MSC-like charts. In the example, two machines exchange messages with each other and with the environment. In this paper, we shall call the former type of messages internal messages and the latter external messages. Case (a) involves multiple stimuli from one entity to the peer entity for a given external input, while cases (b) and (c) involve multiple stimuli from the environment to the system of machines. In (a), machine $M_A$ produces two internal messages $x$ and $y$ arising from the external input $a$. Note that multiple output messages can be produced by an external input as well as by an internal input. In (b), external input $b$ is issued, while the previous input $a$ is still being processed. This is an example of multiple stimuli during a transition from one stable state to another. Note that the global states $(A_1, B_1)$ and $(A_4, B_3)$ in Fig. 1(b) are stable states. A similar case is shown in Fig. 1(c), where one input goes to $M_A$ and another input goes to $M_B$ at the same time. This is known as simultaneous stimuli, which is a special case of multiple stimuli. Simultaneous stimuli can be seen in TCP’s simultaneous open and close behaviors. These cases can often happen in the real world and may also occur in faulty implementations. Therefore, interoperability testing should test for these behaviors as well.

III. FORMAL MODELS

This section defines formal models for communicating entities and for interoperability test cases. The specifications and implementations of communicating entities can be modeled using a special type of FSMs known as input output state machine (IOSM). IOSM is defined as follows:
Definition 1: An IOSM is a five-tuple \((S, S_0, L_{\text{in}}, L_{\text{out}}, \mathbf{Tr})\) where:

1) \(S = \{S_0, S_1, \ldots, S_{n-1}\}\) is a set of states;
2) \(S_0 \in S\) is the initial state;
3) \(L_{\text{in}} = \{v_1, v_2, \ldots, v_m\}\) is a set of input symbols;
4) \(L_{\text{out}} = \{u_1, u_2, \ldots, u_k\}\) is a set of output symbols;
5) \(\mathbf{Tr} \subseteq \{S_0 - v'/U \rightarrow S_d| S_r, S_d \in S \land v \in L_{\text{in}} \land U \in \mathbf{P}(L_{\text{out}})\}\) is a set of transitions, where \(L_{\text{out}}^*\) denotes the set of sequences of symbols in \(L_{\text{out}}\) and \(\mathbf{P}(X)\) denotes the power set of the set \(X\).

Bold letters in Definition 1 represent sets. \(L_{\text{in}}\) is divided into the set of external input symbols \(L_{\text{in,E}}\) and the set of internal input symbols \(L_{\text{in,I}}\). Similarly, \(L_{\text{out}}\) is divided into two sets \(L_{\text{out,E}}\) and \(L_{\text{out,I}}\) that are, respectively, the set of external output symbols and the set of internal output symbols. For example, symbol \(a\) in Fig. 1(a) is an external input symbol and \(a'\) is an external output symbol. Symbols \(x\) and \(y\) are internal output symbols for machine \(M_A\) and, at the same time, are internal input symbols for machine \(M_B\). A transition \(tr\) from state \(A\) to state \(B\), an input symbol and a sequence of output symbols. For a transition \(tr\) in \(\mathbf{Tr}\), the ‘.’ notation is used to refer to one of the elements of the following sets such as \(S_r, S_d,\) and \(U\). For example, \(tr|S_r\) represents the starting state of the transition \(tr\). In case of a deterministic IOSM, there is at most one transition in \(\mathbf{Tr}\) for each input symbol, while there is at least one transition in a completely defined IOSM.

Definition 2: For every transition \(tr(\forall tr \in \mathbf{Tr})\) whose starting state is \(S_i(tr, S_0 = S_i)\), the state \(S_i \in S\) is said to be the following:

1) **controllable** iff \(tr.v \in L_{\text{in,E}}\);
2) **uncontrollable** iff \(tr.v \in L_{\text{in,I}}\);
3) **semi-controllable** iff \(\exists tr.v \in L_{\text{in,E}}\) and \(\exists tr.v \in L_{\text{in,I}}\)
   where \(j \neq k\).

We classify states in an IOSM into three types according to Definition 2. If a state can be changed only by external inputs, the state is said to be **controllable**. Such a state is controlled only by the testers’ inputs. If all possible inputs at a state are internal ones, the state is said to be **uncontrollable**. If some possible inputs at a state are internal and others are external, the state is **semi-controllable**. This classification is illustrated in Fig. 2. In IOSM \(M_A\), a part of which is depicted in the box in Fig. 2, state \(A_1\) is controllable, \(A_3\) is uncontrollable, and \(A_2\) is semi-controllable. There are two transitions from state \(A_2\) as depicted in the box in Fig. 2. One transition goes to state \(A_3\) with external input \(y\) and the other goes to state \(A_4\) with internal input \(c\). Let us trace the behavior of multiple stimuli in this system. Suppose an external input \(x\) is given when the system’s global state is \((A_1, B_1)\), which is a stable state. Then, \(M_A\) goes to state \(A_2\), sending an external output \(x'\) and an internal output \(a\) and \(M_B\) replies with an internal message \(c\). Before the message \(c\) reaches \(M_A\), another external message \(y\) (multiple stimuli from the environment) is issued. This is because the state \(A_2\) of \(M_A\) is a semi-controllable state. If \(M_A\) receives the internal message \(c\) in state \(A_2\), it will go to state \(A_4\), but it will go to state \(A_3\) if it receives the external input \(y\). Although the system is in a transient global state \((A_2, B_2)\), the next stable state of the system is unpredictable since the transient global state includes a semi-controllable state. In general, if a machine in a system is in a semi-controllable state and an internal signal comes from a peer communication machine, the existing test generation methods will not be able to examine the external inputs since they adhere to the single stimulus principle.

Now, we define a behavior model for two IOSMs communicating with each other and the environment. Let \(\Pi\) be the composition of the two IOSMs, one, \(M_A\), for the specification \(A\) and the other, \(M_B\), for the specification \(B\). Then, \(\Pi\) is defined as follows:

1) \(S_{\Pi_0} \subseteq S_\Pi\) is the initial state. Namely, \(S_{\Pi_0} = (S_{A_0}, S_{B_0})\);
2) \(L_{\Pi_{\text{in}}} = \{\psi_{\Pi_1}, \psi_{\Pi_2}, \ldots, \psi_{\Pi_n}\}\) is a set of input symbols. These messages are external inputs of each machine. Namely, \(L_{\Pi_{\text{in}}} = (L_{A_{\text{in}}} \cup L_{B_{\text{in}}})\);
3) \(L_{\Pi_{\text{out}}} = \{\varphi_{\Pi_1}, \varphi_{\Pi_2}, \ldots, \varphi_{\Pi_k}\}\) is a set of output symbols. These messages are external outputs of each machine. Namely, \(L_{\Pi_{\text{out}}} = (L_{A_{\text{out}}} \cup L_{B_{\text{out}}})\).
4) $\mathbf{Tr}_\Pi \subseteq \{\tau_{10}, \tau_{11}, \ldots, \tau_{l-1}\}$ is a set of sequences of transitions, and a sequence of transitions $\{\tau_i\} \Rightarrow \tau_{i_0}, \tau_{i_1}, \ldots, \tau_{i_{l-1}}$ where $\tau_i \in (\mathbf{Tr}_A \cup \mathbf{Tr}_B$).

5) $Q_\Pi = (Q_A, Q_B)$ is a pair of directed channels where $Q_A$ is the channel from $M_B$ to $M_A$, and $Q_B$ from $M_A$ to $M_B$; both channels are represented by queues.

A global state and directed channels (modeled by queues) in a composite IOSM $\Pi$ determine the system state of the IOSM $\Pi$. Here, the system state is used to describe the state of the system as a whole, while a global state defines each local state of the IOSMs in the system. Note that a global state is defined as one describing the entire system, including information for input queues and/or channels in [8] and [10].

An IOSM receives messages from either the environment or its channel queue. Then, the two IOSMs and the two directed channels constitute a composite IOSM $\Pi$ as shown in Fig. 3. Note that in this model IOSM $\Pi$ does not have any queue for inputs from the environment. This is the same as a single IOSM because its formal model does not define any queue. Since each state in an IOSM consumes one message at a time, messages following the current message do not affect the next state. We assume that internal outputs are instantly placed in the channel queue directed to the receiving IOSM. This implies first-come first-served (FCFS) message queuing and handling.

A global state is classified into one of two types depending on the channel contents.

Definition 4: The global states of two IOSMs are classified into two types:

$\forall S_\Pi = (S_A, S_B) \in S_\Pi$

1) a global state $S_\Pi$ is said to be stable iff $Q_A$ and $Q_B$ are both empty;

2) $S_\Pi$ is in a transient state otherwise.

A system in a stable state will change state due to external inputs only. Both local states in a stable state cannot be uncontrollable states, otherwise the system is in a deadlock state because both machines will wait for messages from each other indefinitely. When a system is in a transient state, the system will change its state soon. However, the existing test generation methods do not allow for the system to receive additional external inputs. Our method, on the other hand, examines all possible external inputs. When one of the two local states in a transient global state is semi-controllable, we examine the cases where the IOSM receives an external input in the semi-controllable state, as well as the cases where it consumes an internal message from its receiving channel.

Finally, we define an interoperability test suite in Definition 5. Each machine in a system consumes one input (either internal or external) at each state, that is, one transition at a time for each IOSM. Therefore, an interoperability test case can be represented by a sequence of transitions in either IOSMs in the system. An interoperability test suite is defined in terms of the interoperability test cases. Note that a test case whose length is 1 is simply a conformance test case.

Definition 5: An interoperability test suite $\text{IOPTS} \subseteq \mathbf{Tr}_\Pi$ is defined as follows.

- $\text{IOPTS} = \{iopt_0, iopt_1, \ldots, iopt_{l-1}\}$ is a set of interoperability test cases where an interoperability test case is defined as a sequence of transitions:
  
- $\{iopt\} \Rightarrow \tau_{i_0}, \tau_{i_1}, \ldots, \tau_{i_{l-1}}$ where $\tau_i \in (\mathbf{Tr}_A \cup \mathbf{Tr}_B$, $0 \leq i \leq l-1$, and $l \geq 2$.

The interoperability test case illustrated in Fig. 2 can be represented as follows. The length of this interoperability test case is 5. The starting stable state is $(A_1, B_1)$ and the destination stable state is $(A_4, B_3)$.

$A_1 \xrightarrow{\{x, a\}} A_2 \xrightarrow{\{y, b\}} A_3 \xrightarrow{\{x, c\}} A_4 \xrightarrow{\{y, b\}} B_2 \xrightarrow{\{b, y\}} B_3 \xrightarrow{\{c, y\}} A_3 \xrightarrow{\{c, y\}} A_4$.

IV. TEST CASE GENERATION

This section describes our approach to generating interoperability test cases based on the formal models defined in Section III. First, we explain the overall procedure of our approach and the associated assumptions. Next, we select the test architecture that achieves the desired test coverage. Then, we discuss how to generate interoperability test cases based on the multiple stimuli principle. For the purpose of analysis, we divide the multiple stimuli into two sets as classified in Fig. 1 and consider them in Sections IV-C and IV-D, respectively. Finally, our test generation algorithm is applied to two protocols, the ATM signaling protocol and TCP, with assessment of the results.

A. Overview

In order to derive test cases, we first need to derive IOSMs from the given protocol specifications. We assume that there is no autonomous transition, e.g., a timer. This is because even though we are able to generate test cases involving time constraints, it is difficult to apply them to real test environments. The interoperability test derivation algorithm generates test cases from the IOSMs by constructing an interaction graph. The algorithm starts from the initial stable state consisting of each IOSMs initial states and at each step examines all possible external inputs even in the transient states, until a stable state is reached. In the meantime, new paths are generated based on the multiple stimuli principle, and newly found stable states are added to the associated state space. This procedure is repeated for every new stable state until all the stable states are covered. Our algorithm can be used to derive interoperability test cases based on the single stimulus principle as well, especially when resources (such as available points of control and observation) are limited. The input-enumeration procedure used in our approach is similar to the concept of state-perturbation [15]. However, the goal of our procedure is to build a reachability tree and then generate interoperability test cases based on its stable states, while the goal of state-perturbation is to build a reachability tree for validating protocols between interacting

Authorized licensed use limited to: Korea Advanced Institute of Science and Technology. Downloaded on March 09,2010 at 02:13:03 EST from IEEE Xplore. Restrictions apply.
processes without the concept of stable state. Moreover, in generating interoperability test cases, we mainly focus on inputs and outputs to and from the systems of machines, which are regarded as black- or grey-boxes. However, the model (labeled transition systems) used in paper [15] is focused on internal interactions between systems of machines only.

B. Test Architecture

To cater to the multiple stimuli principle, we need a test architecture with a tester between the two IUTs, as well as one at each end as shown in Fig. 4. The arrows in Fig. 4 represent points of control and observation (PCOs) with both observability and controllability functions. It is assumed that there is a communication channel through which the testers coordinate with each other. Tester C should be able to hold messages passing through it for a while. In this way, simulating multiple stimuli is feasible. In the case of TCP/IP, for example, tester C will be configured with two network interface cards for each IUT. The tester duplicates and drops all the messages between the two IUTs using libpcap [14], a library for system-independent packet capture, and then sends the messages according to the interoperability test cases, coordinating with the other two testers in doing so.

C. Multiple Stimuli From One Machine to Another

Fig. 5 shows the channel status (modeled by queues) for every transition execution in deriving an interoperability test case in the given example. Starting with an external input $e_x$, the algorithm pushes every internal output generated by the input into the channel $Q_B$. Next, it moves to the peer machine $M_B$, processes all the messages in the channel, and pushes all internal outputs produced during this period of time into the channel $Q_A$. This procedure is repeated until both channels are empty. Note that the order of messages, a, b, c (or, 0, 1, 2, 3) remains unchanged since first-come-first-served scheduling is used. Moreover, protocol concurrency is still maintained, e.g., $tr_t \parallel tr_1, tr_2 \parallel tr_3$.

D. Multiple Stimuli From the Environment

To support multiple messages from the environment, an additional step is needed. Before the algorithm picks up an element from a channel, it examines whether the current local state of the machine is semi-controllable or not. If so, a new path is generated for all possible external inputs in this semi-controllable state. At this point, information of the current global state (which is a transient state) and all queues will be copied and the algorithm will generate test cases for each path recursively. The procedure of generating interoperability test cases explained so far can be formalized into Algorithm 1. Lines 1 to 10 describe the main (control) part of the algorithm. This part initializes the variables and calls the $interaction_sequences()$ function for every stable state in the $NEW$ set that has been initialized with the initial stable state ($S_{A0}, S_{B0}$). The function described in the rest of the algorithm generates all possible transitions based on the multiple stimuli principle. In lines 14 to 24, the termination condition is checked by examining the queue contents and updating the sets $NEW$, $IOPTS$, and $TI_{II}$, or calling itself recursively. Lines 25 to 37 examine all possible transitions whose inputs are either external inputs or the current message from the queue. The program then calls itself recursively with appropriate variables for this test case. Finally, the rest of the function checks whether or not the current interoperability test case has encountered a specification error, namely, there is no transition for the current message (so-called implicit signal consumption). This algorithm has been implemented in the C language with about 1000 lines of codes. The execution time of the program is negligible (80 ms for generating 1012 test cases from 150 transitions).

Algorithm 1. Interoperability Test Suite Derivation

\begin{algorithm}
\begin{algorithmic}
\STATE $L_{in} := L_{in}^A \cup L_{in}^B$; $S_{in} := (S_{A0}, S_{B0})$; \hspace{1cm} $S_{\pi} := \{ \}$; \hspace{1cm} $IOPTS := \{ \}$; \hspace{1cm} \hspace{1cm} \hspace{1cm} $NEW := S_{in}$
\WHILE {$NEW \neq \emptyset$}
\STATE set $gs$ to any global state in $NEW$
\STATE delete the global state from $NEW$
\FOR {each possible external input symbol $v$ in global state $gs$}
\STATE prepare two empty queues $Q_A$ and $Q_B$, and an empty transition list $iopt$
\STATE push $v$ to corresponding queue, $Q_A$ or $Q_B$
\ENDFOR
\STATE $interaction_sequences(Q_A \text{ or } Q_B, Q_B \text{ or } Q_A, gs, iopt, NEW)$
\ENDWHILE
\PROCEDURE{$interaction_sequences$ (var $Q_1$, var $Q_2$, $gs$, var $iopt$, var $NEW$)}
\IF {$Q_1$ is empty}
\IF {$Q_2$ is empty}
\IF {$iopt$ consists of more than one transition}
\STATE insert this new interoperability test case $iopt$ to $IOPTS$
\ENDIF
\ELSE $interaction_sequences(Q_2, Q_1, gs, iopt, NEW)$
\ENDIF
\RETURN
\ELSE begin
\STATE $in := \text{pop}(Q_1)$; $atLeastOne := \text{false}$
\ENDIF
\ENDPROCEDURE
\end{algorithmic}
\end{algorithm}
Derivation of interoperability test case for multiple internal outputs.

**Fig. 5.** Derivation of interoperability test case for multiple internal outputs.

**TABLE I**

<table>
<thead>
<tr>
<th>TCP’s connection establishment phase (Each IOSM consists of 19 transitions)</th>
<th>Single stimulus principle</th>
<th>Multiple stimuli principle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total test cases</td>
<td>20</td>
<td>74</td>
</tr>
<tr>
<td>Test cases due to specification error</td>
<td>4</td>
<td>54</td>
</tr>
<tr>
<td>Conformance test cases</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Interoperability test cases</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ATM signaling protocol (Each IOSM consists of 75 transitions)</th>
<th>Single stimulus principle</th>
<th>Multiple stimuli principle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total test cases</td>
<td>70</td>
<td>1012</td>
</tr>
<tr>
<td>Test cases due to specification error</td>
<td>0</td>
<td>6 (0) *</td>
</tr>
<tr>
<td>Conformance test cases</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Interoperability test cases</td>
<td>54</td>
<td>990 (996) *</td>
</tr>
</tbody>
</table>

*There is no test case showing an error when the fixed IOSM is applied by adding a transition, “0:0 0:0 \_REL.” Instead, the number of interoperability test cases increases by 6.

27: **for** each possible transition **tr** from the current global state **gs**
28: change **gs** according to the input in the transition, **tr**
29: copy **Q1**, **Q2**, and **iopt** to new**Q1**, new**Q2**, and new**iopt**, respectively
30: **if** the input **tr** (≠ **in**) is external **then**
31: insert current input **in** to the back of new**Q1**
32: **if** the transition **tr** has internal outputs **then**
33: push all the outputs to new**Q2**
34: append the transition **tr** to new**iopt**
35: interaction_sequences(new**Q1**, new**Q2**, **gs**, new**iopt**, NEW)
36: **atLeastOne** := true
37: **end for**
38: **if** **atLeastOne** := false **then** begin
39: report there is a specification error
40: append an error code to **iopt** to stop expanding the current path
41: interaction_sequences(**Q1**, **Q2**, **gs**, **iopt**, NEW)
42: **end**
43: **end procedure**

**E. Applications**

We apply our test generation algorithm (Algorithm 1) to the connection establishment phase of TCP and the ATM/B-ISDN signaling protocol. Their IOSMs can be found in [7] and [8], respectively. These IOSMs may lack completeness compared to real specifications of the protocols. An example is illustrated later.

Given the IOSMs, the input data needed by our proposed method can be derived systematically. An IOSM is described in a file like that in Appendix I, in which each line represents a transition in the IOSM. For instance, “[−1]0:0 3:6 SETUPa CALL_PROC \_SETUP” in Appendix I is the transition whose starting state is 0:0, destination state is 3:6, input message is SETUPa, and output messages are CALL_PROC and \_SETUP. Each transition has a unique id. The minus sign is used to denote the transitions of one IOSM and the plus sign denotes the transitions of the other IOSM. The postfix is used to specify a specific IOSM, e.g., SETUPb is an external message being sent to IOSM **M** _B_. The prefix “\_” is used to denote an internal message.

The results of applying our method to the two protocols are summarized in Table I. In the case of ATM signaling protocol, a total of 1012 test cases were derived. 990 cases among them are for interoperability testing. Another 16 cases are for conformance testing because they involve one IUT only. The other six cases address specification errors for requesting the IUTs to receive unspecified input messages. Note that our method can also be used in detecting specification errors in the design phase. It is shown that our method generates more test cases than existing...
In order to support multiple stimuli, methods [7], [8] were used to derive interoperability test cases, while only 54 were derived by the method in [7]. Therefore, it is clear the existing methods cannot detect faults due to multiple stimuli. Note that our algorithm can be easily tailored to generate only those interoperability test cases that are related to multiple stimuli.

Fig. 6 shows two examples of the derived interoperability test cases. An interoperability test case consists of several transitions between the IOSMs. Illustrated in Fig. 6(a), the interoperability test case, \( [1] \), involves multiple stimuli. This test case is based on the specification and represents a correct behavior of the protocol. Depicted in Fig. 6(b), the interoperability test case, \( [2] \), reveals a specification error. This leads us to find incompleteness of the IOSM used in [8]. In fact, this case is covered in the error handling clause in ITU-T Q.2931 [23]. By adding a transition to the IOSM to handle the unspecified input "REL", we have derived six more interoperability test cases.

Table II gives the transition coverage for the IOSMs of TCP’s connection establishment phase and the ATM signaling protocol. Note that the traversed transitions include those covered by the conformance test cases. It is shown that our method has higher transition coverage than the existing methods that are based on the single stimulus principle only. Specifically, five more transitions of TCP and nine more transitions of ATM are traversed than the existing methods in [7] and [8]. Our coverage is exactly the same as that covered by validation\(^2\) techniques. A SDL commercial validation tool [13] produces the same results reported in Table II. In other words, our method can generate interoperability test suite whose transition coverage is equivalent to that of validation.

Table III shows the reasons why some transitions are not traversed. First, some transitions are incorrectly specified and will never be executed due to the absence of transitions that send the corresponding signals. For instance, the 38th transition in Appendix I needs an input signal "RESTART", but there is no transition that sends the required signal. Note that a wrongly specified transition may give rise to subsequent untraversed transitions. Also, some transitions are overspecified. Overspecified transitions are superfluous and are unnecessary for our purpose. They include self-transitions whose start state and destination state are the same. Actually, they are sometimes used for error handling. Since checking correct behavior is the objective of our method, such transitions are considered overspecified in this paper. Finally, some transitions are not traversed due to the limitation of the method used. As shown in the table, our method can traverse all the transitions that cannot be handled by the existing methods.

\(^2\)Validation is based on the state space exploration technique. It automatically checks a distributed system’s design for logical errors. Validation executes all possible combination of events that can happen, and reports any indication that something has gone wrong.
It is observed that the multiple stimuli principle incurs a considerable number of interoperability test cases for the ATM signaling protocol. In the following, we provide methods to reduce the number of interoperability test cases, while keeping the same transition coverage.

### V. Test Case Minimization

The remaining 62 transitions were not traversed. We consider only those 88 transitions in the IOSM for minimization. Another factor is the total number of transition executions and it was 6308 times in this case. So the transition execution rate \( r \) is 71.68 (i.e., 6308/88). This implies that each transition would be redundantly executed close to 72 times on average. The minimum cost (or most effective) test is achieved when the transition execution rate \( r \) is 1.

Our derived interoperability test cases originally constituted 990 test cases, which would execute 6308 transitions. In the following, we provide methods to reduce the number of interoperability test cases, while keeping the same transition coverage.

#### B. Definition of the Minimization Problem

Minimizing interoperability test cases with the same transition coverage is similar to the set-covering problem [17]. The set-covering problem is an optimization problem that models many resource-selection problems and is NP-hard.

An instance \((T, TC)\) of the set-covering problem consists of a finite set \(T\) and a family \(TC\) that are subsets of \(T\), such that every element of \(T\) belongs to at least one subset in \(TC\). Fig. 7 gives an example, where \(T = \{t_1, t_2, t_3, t_4, t_5, t_6\}\) and \(TC = \{t_1, t_2, t_3, t_4\}\). The problem is to find a minimum-sized subset \(C \subseteq TC\) whose members cover all of \(T\). With regard to the problem of minimization of interoperability test cases, \(T\) represents a set of transitions in the individual machines of the composite IOSM and \(TC\) is a set of derived interoperability test cases. Each interoperability test case consists of more than one transition in \(T\) and every transition in \(T\) is covered by at least one interoperability test case (a subset) in \(TC\). Note that we do not consider transitions that are not covered by the originally derived interoperability test cases. The minimization of the interoperability test cases can be conducted with two policies:

- **P1**: to reduce the number of interoperability test cases;
- **P2**: to shorten the total length of interoperability test cases.

The length of an interoperability test case is defined as the number of transitions contained in the test case. Note that a smaller number of test cases does not imply shorter total length. However, it is obvious that a long test case is costly in terms of...
testing time and a large number of test cases require effort in

test preparation for each of them. Rigorous analysis on the cost

testing with the interaction effect between these two policies

is not dealt with in this paper.

In essence, policy $P_1$ minimizes the number of sub-

sets of $C$, i.e., $n(C)$, while policy $P_2$ minimizes the total length of subsets of $C$, i.e., $\sum n(c_i)$ where $c_i$ is the $i$th subset of $C$. In the example of Fig. 7, under policy $P_1$, the minimum-sized set cover $C = \{tc1, tc2\}$ or $\{tc1, tc3, tc4\}$, $\{tc1, tc2, tc3, tc5, tc6\}$, while under policy $P_2$, $C = \{tc1, tc3, tc4\}$, or $\{tc1, tc2, tc3, tc4\}, \{tc2, tc5\}, \{tc3, tc6\}$. Although the latter solution has more interoperability test cases—three compared with the former’s two, the total length (8) is shorter than that of the former (9).

To solve the minimization problem, we consider a two-di-

mensional matrix. The minimization problem is modeled so as
to find a subset of the columns of an $M \times N$ zero-one matrix,$

A = \{a_{ki} \in \{0, 1\}; k = 1, \ldots, M; i = 1, \ldots, N\}$, that covers all rows. Here, $M$ is the size of the finite set $T$ and $N$ is the number of subsets in $TC$ in an instance $(T, TC)$ of minimization problem. More precisely, this problem is defined as follows:

\[
\text{Minimize } \sum_{i=1}^{N} s_i \text{ in policy } P_1 \\
or \sum_{i=1}^{N} \sum_{k=1}^{M} a_{ki}s_i \text{ in policy } P_2 \tag{1}
\]

subject to

\[
\sum_{i=1}^{N} a_{ki}s_i \geq 1, \quad k = 1, \ldots, M \tag{2}
\]

with $s_i \in \{0, 1\}, \quad i = 1, \ldots, N. \tag{3}$

Equation (2) states that at least one column must cover a par-
ticular row. Fig. 8 illustrates this matrix model. With policy $P_1$,

we minimize the number of 1’s in $s_i$ subject to the condition

that every row must be set to 1 in at least one of the columns

selected by $s_i$. With policy $P_2$, $s_i$ is set such that the number of selected elements for all rows is minimized.

So far, we have defined the problem of minimizing interoper-

ability test cases. In the following, we will discuss two approaches

for this problem. One is a simple ad-hoc approach that does not

consider either policy explicitly in selecting test cases. The other

approach takes the policies into consideration and is a greedy

heuristics that gives approximate solutions.

C. Ad-Hoc Approach

This approach selects an interoperability test case in an

ad-hoc way until all transitions of the component IOSMs are

covered. For every transition, it simply picks any test case

that covers the transition. This scheme is presented in Algorithm

2. The input is a $M \times N$ zero-one matrix as shown in Fig. 8,

and the output is a set of binary variables $s_i \in \{0, 1\}$ where

$i = 1, \ldots, N$. Note that the input and output are the same

in all the following algorithms we propose. For the ad-hoc

approach, we introduce a temporary variable $v$ to keep the

visited (or covered) elements. From lines 2 through 6, for each

element $v_k$, the algorithm selects any $i$ such that $a_{ki} = 1$ and

sets $s_i = 1$. This means that the $i$th column is selected as an

element of the minimized set. Then, in line 5, it updates $v$ for

the other elements in the $i$th column since the column covers

them as well. The complexity of this algorithm is $O(M)$, i.e., it

is proportional to the number of given transitions to cover in the

component IOSMs. The result obtained by using this ad-hoc

algorithm can be costly because it does not consider redundancy

or optimization. However, this algorithm can be embedded into

our test generation method presented in Algorithm 1 so that the

test cases are minimized at the same time as they are derived.

Algorithm 2. Ad-Hoc Minimization

Ad-Hoc MINIMIZATION

Input. $M \times N$ zero-one matrix $A$. 
Output. Column selector $s$.

1: $v_k = 0, s_i = 0$
2: for $k = 1, \ldots, M$
3: if $v_k = 0$
4: choose any $i$ such that $a_{ki} = 1$
5: update $v_k = a_{ki}$, where $a_{ki} = 1$ and
6: $k = 1, \ldots, M$
7: set $s_i = 1$
8: return $s$

D. Greedy Heuristics

To get smaller size set, we examine greedy heuristics with a

logarithmic ratio bound [17]. We propose two algorithms with

policies $P_1$ and $P_2$. With policy $P_1$, which aims to

reduce the number of interoperability test cases, our greedy

method works by picking the set that covers the largest number

of remaining uncovered elements at each stage. This algorithm

is called quantity minimization and is shown in Algorithm 3.

The difference from the ad-hoc minimization is that it selects,

at each stage, a column that updates $v_k$ at maximum. As a re-

sult, it would cover all rows quicker and, hence, select rela-

tively small number of columns. When there is more than one

choice, it selects the column that has a minimum column sum,

i.e., shortest-length test case.

Algorithm 3. Quantity Minimization (With

Policy $P_1$)

QUANTITY MINIMIZATION

Input. $M \times N$ zero-one matrix $A$. 
Output. Column selector $s$.

1: $v_k = 0, s_i = 0, B = A$
2: until all $v_k$ become 1
3: find $k$ such that $k$th row sum of $B$
4: if row sum = 1 then (there is only
5: find the $s_i$ such that $b_{ki} = 1$
TABLE IV

<table>
<thead>
<tr>
<th>Methods</th>
<th>Without Minimization (Algorithm 1)</th>
<th>Ad-hoc Minimization (without policy) (Algorithm 2)</th>
<th>Quantity Minimization (with policy P1) (Algorithm 3)</th>
<th>Length Minimization (with policy P2) (Algorithm 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test cases</td>
<td>-</td>
<td>Appendix II</td>
<td>Appendix III</td>
<td>Appendix IV</td>
</tr>
<tr>
<td>The number of test cases</td>
<td>990</td>
<td>60</td>
<td>35</td>
<td>38</td>
</tr>
<tr>
<td>Total execution of transitions, $E(M)$</td>
<td>6308</td>
<td>433</td>
<td>182</td>
<td>172</td>
</tr>
<tr>
<td>Execution rate, $r$</td>
<td>71.68</td>
<td>4.92</td>
<td>2.07</td>
<td>1.95</td>
</tr>
</tbody>
</table>

6: else (there is more than one column)
7: select an $s_i$, with priorities:
8: 1st. making max-updates for $v_k$
9: 2nd. minimizing column sum $\sum a_{ki}$, $k = 1, \ldots, M$
10: update $v_k = a_{ki}$, where $a_{ki} = 1$ and $k = 1, \ldots, M$
11: update $b_{kj} = 0$, where $a_{ki} = 1$, $k = 1, \ldots, M$ and $j = 1, \ldots, N$
12: set $s_i = 1$
13: return $s$

With policy P2, which pursues the minimum total length of interoperability test cases, our greedy method works by picking, at each stage, the set that has the fewest number of elements that are already covered. In other words, it tries to keep redundancy as low as possible. This algorithm is called length minimization and is shown in Algorithm 4. Note that the difference between length minimization and quantity minimization is the priority they use. As shown in lines 8 and 9, it first finds the columns, which cover the already covered rows the least, and then chooses the column that updates $v_k$ the most. By doing so, the result approximates the optimal solution that minimizes the total length of test cases because at each stage it picks the column that increases redundancy least. When there is no redundancy in every stage, which means that each column always covers new rows, the total length of the interoperability test cases is the same as the number of rows (i.e., the number of transitions in the given component IOSMs).

Algorithm 4. Length Minimization (With Policy P2)

**LENGTH MINIMIZATION**

**Input.** M x N zero-one matrix $A$.

**Output.** Column selector $s$.

1: $v_k = 0$, $s_i = 0$, $B = A$
2: until all $v_k$ become 1
3: find $k$ such that $k$th row sum of $B$ is nonzero minimum
4: if row sum = 1 then (there is only one column)
5: find the $s_i$ such that $b_{ki} = 1$
6: else (there is more than one column)
7: select an $s_i$, with priorities:
8: 1st. making $\sum v_k$ smallest for all $k$ such that $a_{ki} = 1$
9: 2nd. making max-update for $v_k$
10: update $v_k = a_{ki}$, where $a_{ki} = 1$ and $k = 1, \ldots, M$
11: update $b_{kj} = 0$, where $a_{ki} = 1$, $k = 1, \ldots, M$ and $j = 1, \ldots, N$
12: set $s_i = 1$
13: return $s$

E. Applications

By applying the three proposed algorithms, namely, ad-hoc, quantity, and length minimizations, to the ATM signaling protocol, we obtained the results shown in Appendices II, III, and IV, respectively, and they are summarized in Table IV.

Basically, it is shown that we can reduce the number of interoperability test cases by about 94% even with the ad-hoc algorithm (from 990 to 60 test cases). Note that the quantity minimization algorithm results in the smallest number of test cases (35) with slightly more transition executions (i.e., total length of test cases) (182) than that of length minimization algorithm (172 transitions in 38 test cases). Both quantity and length minimizations have similar execution rate of about two, i.e., each transition in the given component IOSMs will be executed twice on average when our interoperability test cases are applied. It would be around five times in case of ad-hoc minimization. As stated earlier, ad-hoc minimization can be combined with our test generation algorithm shown in Algorithm 1 so that it is not necessary to explore all the patterns of interoperating behavior.

Although the results of quantity and length minimizations are similar, the cost required in actual testing may be different. For example, suppose the cost required in preparing and driving the machines to a state ready for applying each single interoperability test case is very high, it would be desirable to have a smaller number of test cases with slightly more transition executions. Rigorous analysis of this tradeoff requires further research.

VI. Conclusion

We have proposed a new method for automatically generating and minimizing interoperability test suites that support multiple inputs to the SUT. This means that the method adopts the **multiple stimuli principle** rather than the traditional **single stimulus principle** used in existing methods. As a result, transition coverage obtained by our method covers the same behavior...
space that is managed by the validation technique. A well-designed protocol is supposed to be able to deal with the race condition, i.e., anomalous behavior caused by unexpected relative timing of events. In other words, the protocol designer has incorrectly assumed that a particular event would always happen before another. Our method can generate interoperability test cases that test for the ability of the implementation to handle possible race conditions, and can also be used in the design phase to check whether or not race conditions would occur in the protocol.

To do this, we have developed formal models on which the interoperability test suite derivation algorithm is based. We have applied our test generation method to the ATM signaling protocol and also to a part of the TCP protocol. Experimental results have shown that our method produces test suites with higher transition coverage than the existing methods: 26% higher for TCP and 12% higher for ATM. The additional transition coverage is for possible complex simultaneous interactions allowed by the protocol specifications.

Since executing test cases is expensive, we have also developed three automatic minimization algorithms that can reduce the test size. They are able to reduce the interoperability test cases considerably, while maintaining the same transition coverage.

**APPENDIX I**
TRANSITIONS OF AN IOSM OF THE ATM SIGNALING PROTOCOL

| [1] | 0:4 | SETUP / CALL PROTOCOL / SETUP
| [2] | 0:9 | CALL PROC
| [3] | 10:10 | CALL CONNECT
| [5] | 10:10 | CONNECT / CALL CONNECT / CONNECT
| [6] | 3:3 | REL / COMP
| [7] | 10:10 | CONNECT / CALL CONNECT / REL
| [8] | 4:1 | RESTART / RESTART / ACK
| [9] | 12:0 | RESTART / RESTART / ACK
| [10] | 12:0 | RESTART / RESTART / ACK
| [12] | 12:0 | RESTART / RESTART / ACK
| [13] | 12:0 | RESTART / RESTART / ACK
| [14] | 12:0 | RESTART / RESTART / ACK
| [15] | 12:0 | RESTART / RESTART / ACK
| [16] | 12:0 | RESTART / RESTART / ACK
| [17] | 12:0 | RESTART / RESTART / ACK
| [18] | 12:0 | RESTART / RESTART / ACK
| [19] | 12:0 | RESTART / RESTART / ACK
| [20] | 12:0 | RESTART / RESTART / ACK
| [21] | 12:0 | RESTART / RESTART / ACK
| [22] | 12:0 | RESTART / RESTART / ACK
| [23] | 12:0 | RESTART / RESTART / ACK
| [24] | 12:0 | RESTART / RESTART / ACK
| [25] | 12:0 | RESTART / RESTART / ACK
| [26] | 12:0 | RESTART / RESTART / ACK
| [27] | 12:0 | RESTART / RESTART / ACK
| [28] | 12:0 | RESTART / RESTART / ACK
| [29] | 12:0 | RESTART / RESTART / ACK
| [30] | 12:0 | RESTART / RESTART / ACK

**APPENDIX II**
TEST CASES MINIMIZED BY AD-HOC MINIMIZATION

| [1] | 4:1 | RESTART / RESTART / ACK
| [2] | 5:3 | RESTART / RESTART / ACK
| [3] | 4:1 | RESTART / RESTART / ACK
| [7] | 6:12 | RESTART / RESTART / ACK
| [8] | 6:12 | RESTART / RESTART / ACK
| [9] | 6:12 | RESTART / RESTART / ACK
| [10] | 6:12 | RESTART / RESTART / ACK
| [12] | 6:12 | RESTART / RESTART / ACK
| [14] | 6:12 | RESTART / RESTART / ACK
| [16] | 6:12 | RESTART / RESTART / ACK
| [17] | 6:12 | RESTART / RESTART / ACK
| [18] | 6:12 | RESTART / RESTART / ACK

**APPENDIX III**
TEST CASES MINIMIZED BY QUANTITY MINIMIZATION

| [1] | 1:2 | START / START / ACK
| [4] | 4:8 | START / START / ACK
| [5] | 5:10 | START / START / ACK
| [7] | 7:21 | START / START / ACK
| [8] | 8:28 | START / START / ACK
| [9] | 9:36 | START / START / ACK
| [10] | 10:45 | START / START / ACK
| [14] | 14:92 | START / START / ACK
| [16] | 16:123 | START / START / ACK
| [17] | 17:141 | START / START / ACK
| [18] | 18:160 | START / START / ACK

APPENDIX II

**APPENDIX II**
TEST CASES MINIMIZED BY AD-HOC MINIMIZATION

| [1] | 1:4 | -12 / 60,45,8, 8
| [2] | 1:4 | -12 / 60,45,8, 8
| [3] | 1:4 | -12 / 60,45,8, 8
| [4] | 1:4 | -12 / 60,45,8, 8
| [5] | 1:4 | -12 / 60,45,8, 8
| [6] | 1:4 | -12 / 60,45,8, 8
| [7] | 1:4 | -12 / 60,45,8, 8
| [8] | 1:4 | -12 / 60,45,8, 8
| [9] | 1:4 | -12 / 60,45,8, 8
| [10] | 1:4 | -12 / 60,45,8, 8
| [11] | 1:4 | -12 / 60,45,8, 8
| [12] | 1:4 | -12 / 60,45,8, 8
| [13] | 1:4 | -12 / 60,45,8, 8
| [14] | 1:4 | -12 / 60,45,8, 8
| [15] | 1:4 | -12 / 60,45,8, 8
| [16] | 1:4 | -12 / 60,45,8, 8
| [17] | 1:4 | -12 / 60,45,8, 8
| [18] | 1:4 | -12 / 60,45,8, 8
| [19] | 1:4 | -12 / 60,45,8, 8
| [20] | 1:4 | -12 / 60,45,8, 8
| [21] | 1:4 | -12 / 60,45,8, 8
| [22] | 1:4 | -12 / 60,45,8, 8
| [23] | 1:4 | -12 / 60,45,8, 8
| [24] | 1:4 | -12 / 60,45,8, 8
| [25] | 1:4 | -12 / 60,45,8, 8
| [26] | 1:4 | -12 / 60,45,8, 8
| [27] | 1:4 | -12 / 60,45,8, 8
| [28] | 1:4 | -12 / 60,45,8, 8
| [29] | 1:4 | -12 / 60,45,8, 8
| [30] | 1:4 | -12 / 60,45,8, 8

APPENDIX III

**APPENDIX III**
TEST CASES MINIMIZED BY QUANTITY MINIMIZATION

| [1] | 1:4 | -12 / 60,45,8, 8
| [2] | 1:4 | -12 / 60,45,8, 8
| [3] | 1:4 | -12 / 60,45,8, 8
| [4] | 1:4 | -12 / 60,45,8, 8
| [5] | 1:4 | -12 / 60,45,8, 8
| [6] | 1:4 | -12 / 60,45,8, 8
| [7] | 1:4 | -12 / 60,45,8, 8
| [8] | 1:4 | -12 / 60,45,8, 8
| [9] | 1:4 | -12 / 60,45,8, 8
| [10] | 1:4 | -12 / 60,45,8, 8
| [11] | 1:4 | -12 / 60,45,8, 8
| [12] | 1:4 | -12 / 60,45,8, 8
| [13] | 1:4 | -12 / 60,45,8, 8
| [14] | 1:4 | -12 / 60,45,8, 8
| [15] | 1:4 | -12 / 60,45,8, 8
| [16] | 1:4 | -12 / 60,45,8, 8
| [17] | 1:4 | -12 / 60,45,8, 8
| [18] | 1:4 | -12 / 60,45,8, 8

Authorized licensed use limited to: Korea Advanced Institute of Science and Technology. Downloaded on March 09,2010 at 02:13:03 EST from IEEE Xplore. Restrictions apply.
APPENDIX IV

TEST CASES MINIMIZED BY LENGTH MINIMIZATION

| [1] | <1,74,27> |
| [2] | <1,7,4,27> |
| [3] | <5,3> |
| [4] | <5,3> |
| [5] | <1,7,5,134.8, 8> |
| [6] | <1,7,5,2, 23,9> |
| [7] | <7,48,26,53> |
| [8] | <7,48,26, 51> |
| [9] | <12, 24, 9> |
| [10] | <18, 24, 9> |
| [11] | <18, 6, 25, 9> |
| [12] | <3, 6, 25, 9> |
| [13] | <1, 4, 4, 14, 8, 52> |
| [14] | <3, 3, 19, 6, 8> |
| [15] | <3, 5, 23, 9> |
| [16] | <1, 4, 13, 9> |
| [17] | <1, 4, 4, 19, 8, 52> |
| [18] | <3, 5, 23, 9> |
| [19] | <15, 23, 9> |

[20] | <20, 23, 9> |
| [21] | <3, 6, 23, 9> |
| [22] | <15, 23, 9> |
| [23] | <20, 23, 9> |
| [24] | <1, 4, 12, 60, 49, 45, 8, 8> |
| [25] | <1, 4, 17, 60, 49, 46, 8> |
| [26] | <1, 4, 32, 60, 49, 45, 8> |
| [27] | <1, 4, 12, 60, 49, 45, 8, 8> |
| [28] | <1, 4, 17, 60, 49, 46, 8, 8> |
| [29] | <1, 4, 32, 60, 49, 45, 8> |
| [30] | <1, 4, 4, 5, 16, 8, 8> |
| [31] | <1, 4, 12, 60, 49, 45, 8> |
| [32] | <1, 4, 17, 60, 49, 46, 8, 8> |
| [33] | <1, 4, 32, 60, 49, 45, 8> |
| [34] | <1, 4, 4, 5, 16, 8, 8> |
| [35] | <1, 4, 12, 60, 49, 45, 8> |
| [36] | <1, 4, 17, 60, 49, 46, 8, 8> |
| [37] | <1, 4, 32, 60, 49, 45, 8> |
| [38] | <1, 4, 4, 5, 16, 8, 8> |

REFERENCES


Samuel T. Chanson (M’76) received the B.Sc. degree in electrical engineering from Hong Kong University, in 1969, and the M.Sc. and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 1971 and 1975, respectively.

He was a faculty member with the School of Electrical Engineering, Purdue University, West Lafayette, IN, for two years. Then, he was with the Department of Computer Science, University of British Columbia, Vancouver, BC, Canada, where he became a Full Professor and Director of the Distributed Systems Research Laboratory. In 1993, he joined the Computer Science Department, Hong Kong University of Science and Technology. He has chaired many international conferences on communication protocols and distributed systems. His research interests include web content delivery technologies, QoS routing, energy-aware scheduling, and Internet security. He has published more than 160 papers in these areas.

Prof. Chanson has served as an Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING, the Journal of Computing and Information, and New Generation Computing. He is a member of the International Federation of Information Processing (IFIP) TC6/WG6.1 Committee.

Sungwon Kang received the B.A. degree from Seoul National University, Seoul, Korea, in 1982, and the M.S. and Ph.D. degrees in computer science from the University of Iowa, Iowa City, in 1989 and 1992, respectively.

From 1993 to 2001, he was a Principal Researcher with the Korea Telecom Research and Development Group, Seoul, Korea. Throughout this period, his main area of research was protocol testing and formal methods. Then, he joined the Information and Communications University, Seoul, Korea, where he is an Assistant Professor. Since 2003, he has been an adjunct faculty member of Carnegie-Mellon University, Pittsburgh, PA, for the Master of Software Engineering Program. Areas of research interest include software engineering, software architecture, software testing, and formal methods.