INFORMATICA, 2011, Vol. 22, No. 3, 447–469

A DFSM-Based Protocol Conformance Testing and Diagnosing Method

Xinchang ZHANG[1], Meihong YANG[1], Guanggang GENG[2], Wanming LUO[2]
[1]*Shandong Key Laboratory of Computer Networks, Shandong Computer Science Center Jinan 250014, China*
[2]*Computer Network Information Center, Chinese Academy of Sciences Beijing 100049, China*
e-mail: xinczhang@hotmail.com, yangmh@keylab.net, gengguanggang@cnnic.cn, lwm@cnic.cn

Received: April 2010; accepted: May 2011

Abstract. In the protocol conformance testing, many existing test methods can effectively detect the possible faults of the implementation under test. However, it is difficult to diagnose the found faults in terms of the test results. This paper presents a diagnosable input/output (DIO) sequence, to differentiate a state from other states under a given condition. We further propose a two-tier protocol conformance testing and diagnosing method based on DIO sequences. The proposed method can effectively detect and diagnose the possible faults of the implementation of a protocol.

Keywords: protocol conformance testing, fault detection, fault diagnosis, DFSM model.

1. Introduction

The wide applications of communication services have accelerated the researches of various new or improved protocols (e.g., Liao *et al.*, 2010; Tseng *et al.*, 2010 and Yoon and Yoo, 2010). The correct implementation of a protocol must conform to the corresponding protocol specification, because the protocol conformance is one of basic prerequisites for errorless interactions among communicating entities. Therefore protocol conformance testing is an indispensable part of the development of a communication system. One of the major tasks of the protocol conformance testing is to generate test sequences for the control portion of the protocol. A test sequence is the concatenation of input/output pairs. The input sequence is applied to the implementation being tested, and the corresponding output sequence will be observed. If the output sequence is not coincident with the expected output sequence, then we say that the implementation has some faults.

The control part of a protocol specification is typically modeled as a deterministic Finite State Machine (FSM), and the FSM-based test sequence generation methods for the conformance testing have been widely studied. These methods detect the possible faults of the implementation under test. However, they usually have limited capabilities of locating the faults. Ramalingam *et al.* (1995) analyzed the diagnosis capabilities of some FSM-based test sequence generation methods, and claims that these methods have the capability of locating the fault to a set including more than n transitions, where n indicates the number of the states of the specification FSM.

An efficient fault diagnosis is very helpful to shorten the time of revising the faults found in the protocol conformance testing. However, it is difficult to accurately diagnose the found faults in terms of the test results. The main reasons include that (1) a single test sequence in the test case might not detect the related faults, and (2) the symptoms might not accurately respond to the corresponding faults in the test procedure. Up to now, there have been only limited researches in this area. Ghedamsi *et al.* proposed two methods to provide fault hints through a set of the diagnosis, each of which is formed by a set of transitions that are suspected of being faulty (see Ghedamsi and Bochmann, 1992; Ghedamsi *et al.*, 1993). Vuong and Ko (1990) presented a diagnosis approach which uses considerable additional test sequences to find the FSM model for the implementation under test under some assumptions. Note that the above fault diagnosis methods, as well as most of FSM-based test sequence generation methods, are actually based on the deterministic FSM (DFSM). So far accurately diagnosing the found faults of an implementation is still an open problem.

In this paper, we classify the transfer faults into two categories according to the specified input/output pairs, i.e., IO-wrong transfer faults and IO-correct transfer faults. We propose a diagnosable input/output (DIO) sequence for identifying a given specified state. Under the assumption that an implementation has no fault or only has IO-correct transfer faults, the DIO sequence can differentiate the associated state from other states. We further propose a protocol conformance testing and diagnosing method that includes two fault detection and diagnosis procedures at different levels. The lower-tier fault detection and diagnosis procedure is performed in the development procedure, to find the possible output faults, IO-wrong transfer faults and missing/extra state faults through the local tests. Consequently, the found faults in the lower-tier procedure can be well located. An implementation that past the lower-tier test satisfies the assumption mentioned above. The upper-tier fault detection and diagnosis procedure can accurately locate the IO-correct transfer fault if the starting state and ending state of the transition can be identified based on the DIO sequences.

In Section 2, we present the deterministic finite state machine model for protocol conformance testing and the types of the faults in the implementation under test. The DIO sequence and related concepts are described in Section 3. We introduce the DIO sequence searching algorithm and fault detection and diagnosis procedures for protocol conformance testing in Section 4. In Section 5, the related work will be introduced. Finally, we summarize this paper in Section 6.

2. Preliminaries

2.1. *Deterministic Finite State Machine*

The control portion of the protocol specification and implementation usually can be modeled as a deterministic finite state machine. A deterministic finite state machine is an initialized deterministic Mealy machine that can be formally defined as a 6-tuple $M = (S, X, Y, \delta, \lambda, s_0)$ (Gill, 1962), where S is a finite set of states, s_0 is the initial state, X is a finite set of input symbols, Y is a finite set of output symbols, δ and λ are a pair of characterizing functions given by

$$s_j = \delta(s_i, x),$$
$$y = \lambda(s_i, x),$$

where $s_i, s_j \in S$, x and y are input symbol and output symbol, respectively.

In this paper, we use the notation $s_i \times x \xrightarrow{y} s_j$ to represent a transition, which means that the DFSM M at state s_i responds with an output y and enters the state s_j when the input x is applied, where $s_j = \delta(s_i, x)$ and $y = \lambda(s_i, x)$. State s_i and s_j are the starting state and ending state of the transition $s_i \times x \xrightarrow{y} s_j$, respectively. If we are not interested in the output of a transition $s_i \times x \xrightarrow{y} s_j$, we use the notation $s_i \times x \rightarrow s_j$ to mean the transition. Similarly, we sometimes employ notation $s_i \times x \rightarrow$ to mean a transition of deterministic FSM.

FSM M is said to be a complete FSM if the transition $s_i \times x \rightarrow$ is defined for any state s_i and input x; otherwise, M is said to be a partial FSM (see (Petrenko and Yevtushenko, 2005; Sabnani and Dahbura, 1988). In most cases, only a minority of inputs are specified for a given state of the protocol model, some examples can be seen in Wang *et al.* (2004), Zhang *et al.* (2006).

In the implementation of a partial FSM, an input error is returned when a unspecified input x is applied to a state s_i. For transforming a partial FSM into a complete FSM, some additional transitions are added to the FSM. In Sabnani and Dahbura (1988), for satisfying completeness assumption, there is a self-loop edge, with a null output in its label, from each state corresponding to each input which is ignored. To distinguish the normal protocol behavior from unexpected one, we introduce the following definition.

DEFINITION 1. An exceptional output is a response to the transition which is not specified or is specified only for satisfying completeness assumption.

According to Definition 1, the null output in Sabnani and Dahbura (1988) is an exceptional output. In this paper, we use symbol $\circ$ to represent the exceptional output.

2.2. *Fault Classes*

In the protocol conformance testing, the faults of the implementation can be divided into the following four classes (see Chow, 1978; Fujiwara *et al.*, 1991; Ghedamsi and Bochmann, 1992).

- **Output Fault:** A transition, denoted by $s_i \times x \xrightarrow{y} s_j$, has an output fault if the implementation produces an output different from y when input x is applied to state s_i.

- **Transfer Fault:** A transition, denoted by $s_i \times x \xrightarrow{y} s_j$, has a transfer fault if the implementation enters a state different from s_j when input x is applied to the state s_i.
- **Missing State Fault:** An implementation has a missing state fault, if the implementation cannot enter any state when a transition in the specification is tested.
- **Extra State Fault:** An implementation has an extra state if the implementation cannot enter the state in terms of the specification.

To test the protocol conformance, some test sequences is applied to the implementation. If there are some differences (called "symptoms" as in Ghedamsi and Bochmann, 1992) between expected output sequences and observed output sequences, we can decide that there are some differences between the specification and implementation. However, the symptoms might not accurately respond to the corresponding faults in the test procedure.

EXAMPLE 1. Let us consider a specification DFSM shown in Fig. 1a and its implementation shown in Fig. 1b. Table 1 shows three test sequences and corresponding outputs.

When test sequence "$a,\ b,\ a$" is performed starting from state s_0, the output sequence is equivalent to the expected one through the past state sequence is not equivalent to the

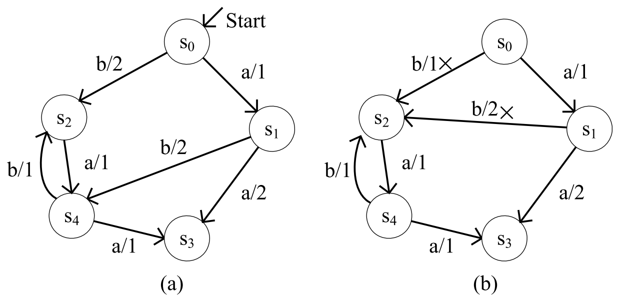

Fig. 1. An example of the specification DFSM and corresponding implementation DFSM.

Table 1

Three test sequences (TSs) and their responses

TS	Excepted output	Output
a, b, a	1, 2, 1	1, 2, 1
$b,\ a,\ a$	2, 1, 1	$\underline{1}$, 1, 1
$a,\ b,\ b$	1, 2, 1	1, 2, $\underline{o}$
TS	Expected state sequence	State sequence
$a,\ b,\ a$	s_0, s_1, s_4, s_3	$s_0, s_1, \underline{s_2, s_4}$
$b,\ a,\ a$	s_0, s_2, s_4, s_3	s_0, s_2, s_4, s_3
$a,\ b,\ b$	s_0, s_1, s_4, s_2	$s_0, s_1, \underline{s_2, null}$

expected state sequence. The input sequence "$a,\ b,\ b$" can detect the fault, but the it cannot reflect the fault in the accurate moment of the fault occurrence.

From the above example, we can see that the symptoms cannot reflect the real faults in some cases. Therefore it is difficult to quickly and accurately diagnose the faults in the conformance testing procedure.

3. Diagnosable Input/Output Sequence

In a given DFSM model, an input/output pair x/y is said to be specified for state s if and only if there exists a transition $s \times x \xrightarrow{y}$. Clearly, two different states might have the same specified input/output pairs.

DEFINITION 2. Some states of a DFSM are said to be IO-equivalent, if and only if the specified input/output pair sets for the states are identical to each other.

For the example shown in Fig. 2a, state s_4 and s_7 are IO-equivalent because the specified input/output pair set for each of them are $\{e/1\}$. Formally, we use $s_i \overset{IO}{=} s_j$ to mean that state s_i and s_j are IO-equivalent. We introduce two types of sets for a given state s, i.e., IO-equivalent set $S_{IO}(s)$ and IO-inclusive set $S^+_{IO}(s)$, $S_{IO}(s) = \{s_i | s_i \overset{IO}{=} s\}$, $S^+_{IO}(s) = \{s_i | IO(s) \subseteq IO(s_i)\}$, where $IO(s)$ represents the set of all specified input/output pairs for state s. Clearly, $S_{IO}(s) \subseteq S^+_{IO}(s)$.

We use IO-inclusive set to classify the transfer faults into two categories, i.e., IO-wrong transfer fault and IO-correct transfer fault, which are defined in following:

DEFINITION 3. The implementation of transition $s_i \times x \xrightarrow{y} s_j$ is said to be IO-wrong if the ending state of the transition in the implementation is not in $S^+_{IO}(s_j)$. If the ending state s_k $(k \neq j)$ of the transition in the implementation is in set $S^+_{IO}(s)$, the transfer fault is said to be IO-correct.

In the example of Fig. 2a, if the ending state of $s_0 \times b \xrightarrow{1} s_1$ in the implementation is state s_8, then we say that the transition has an IO-wrong transfer fault. However, the transition has an IO-correct transfer fault if the ending state is s_2.

In this section, we assume that the implementation has no output fault, IO-wrong transfer fault and missing/extra state fault. The above assumption is called IO-correct assumption, and we will explain how to satisfy it in Section 4. Under the above assumption, we propose an identification sequence, called Diagnosable Input/Output (DIO) sequence, for each state of the specification DFSM if it exists.

DEFINITION 4. The DIO sequence for state s is an input/output sequence which can identify state s in the implementation under the IO-correct assumption.

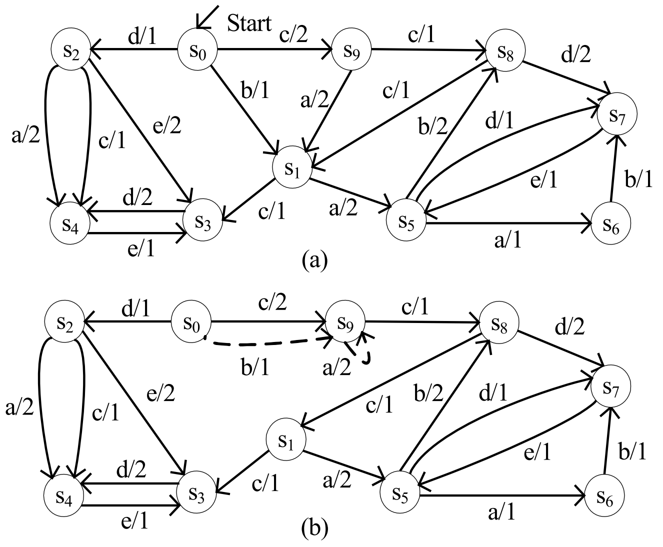

Fig. 2. An example DFSM.

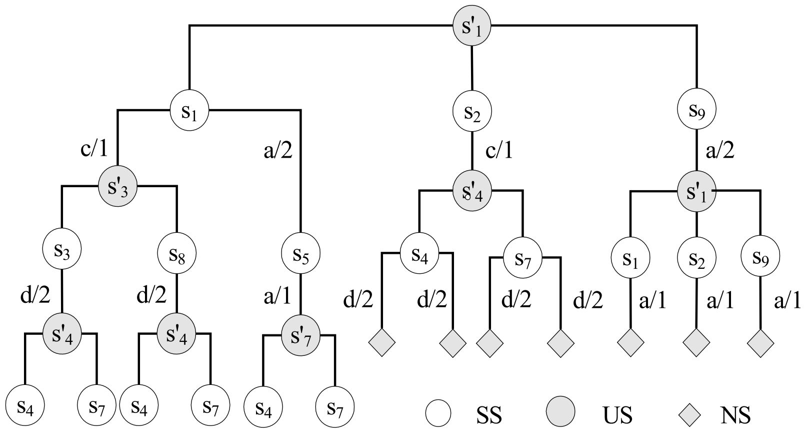

Fig. 3. An example of the transfer tree.

Formally, a DIO sequence for state s_i, denoted $dios(s_i)$, is an input/output sequence $(i_1/o_1)(i_2/o_2)\dots(i_m/o_m)$ such that: $\forall j$ $(j \neq i)$, different output sequences are produced when $dios(s_i)$ is applied to s_i and s_j. Additionally, we employ denotation $dios_i(s_i)$ and $dios_o(s_i)$ to mean the input part and output part of $dios(s_i)$, respectively. For example, if $dios(s_i) = (a/1)(b/2)(c/1)$, then $dios_i(s_i) = abc$, and $dios_o(s_i) = 121$.

We introduce a special tree, called transfer tree (TT), to illustrate and build the possible DIO sequence for a given state. The transfer tree gives the possible transfer space under the IO-correct assumption. This section explains the structure of the transfer tree, and the corresponding building procedure will be depicted in next section. As Fig. 3 shows, there are the following three types of nodes in the transfer tree:

- **Specified State (SS)** node represents a state of the specification DFSM.
- **Uncertain State (US)** node means an uncertain state which might be any state in an IO-inclusive set.
- **Nonexistent State (NS)** node indicates a nonexistent state. Any NS node is a leaf node.

Each SS node is labeled by s_i if it represents state s_i, and each US is labeled by s'_j, where s_j means some specified state. If a node is labeled by s_i, then we say that the label state of the node is s_i.

Each transfer tree is in association with a certain state s_i. The tree root is a US node, labeled by s'_i. A US node labeled by s'_j has $|S^+_{IO}(s_j)|$ SS children, labeled by the states in $S^+_{IO}(s_j)$, respectively. Only the edge starting from the SS node is labeled by some input/output pair. Assume that an edge labeled by x/y starts from a tree node labeled bys_j, then the ending node of the edge is (1) a SS node labeled by s_k if $s_j \times x \xrightarrow{y} s_k$ and the possible transition is unique (i.e., $|S^+_{IO}(s_k)| = 1$, see line 3–5 of Algorithm 3), or (2) a US node labeled by s'_k if $s_j \times x \xrightarrow{y} s_k$ and the possible transition is not unique, or (3) a NS node if $s_j \times x \xrightarrow{y}$ is not specified. Note that different tree nodes can have the same label state.

In the transfer tree, each node is at some level. Practically, the tree root is at level 0. We use function $lev(n)$ to represent the level of node n other than the root. Function $lev(n)$ is defined as

$$lev(n) = \begin{cases} lev(p(n)), & \text{if } p(n) \text{ is a US,} \\ lev(p(n)) + 1, & \text{otherwise,} \end{cases}$$

where $p(n)$ means the parent node of node n.

In the transfer tree for state s_i, each downstream SS node of the 0-level node which is labeled by s_i is said to be the test point (or node), and other node (other than the tree root) is said to be the reference point (or node). Particularly, the test point and reference point at level 0 are said to be main test point and main reference point, respectively. Additionally, we introduce two types of paths by the following definition.

DEFINITION 5. In a given transfer tree, the path from main test point to one of its downstream leaf nodes is said to be a test path, while the path from a main reference point to one of its downstream leaf nodes is said to be a reference path.

The concatenations of input/output pairs in the test path and reference path are said to be Test Input/Output (TIO) sequence and Reference Input/Output (RIO) sequence, respectively. In the transfer tree, the last input/output pair of each RIO sequence can be represented by $x/\circ$, where x is an input and symbol $\circ$ means the exceptional output. Let a TIO sequence be $(i_1/o_1)(i_2/o_2)\ldots(i_m/y)\ldots$ and a RIO sequence be $(i_1/o_1)(i_2/o_2)\ldots(i_m/\circ)$, we say that the TIO sequence includes the RIO sequence. In addition, we say that a TIO sequence is hidden if it is a prefix of another TIO sequence.

In the transfer tree, the subtree rooted by the main test point is called test subtree, and the subtree rooted by a main reference point is said to be a reference subtree. The transfer tree has one and only one test subtree. Each node of the test subtree has at least one associated reference node, which is defined in the following:

DEFINITION 6. Let the input/output pairs of the path, from the main test point to a test point t, be represented by $(i_1/o_1)(i_2/o_2)\dots(i_m/o_m)$, a reference node r is said to be an associated reference node of t if there is a reference path ending at r such that the input/output sequence in the reference path is $(i_1/o_1)(i_2/o_2)\dots(i_m/o_m)$ or $(i_1/o_1)(i_2/o_2)\dots(i_m/\circ)$.

Formally, we use $RS(s)$ to indicate the set of associated reference nodes (except NS nodes) of test point s. Similarly, suppose that a reference point r is one of associated reference nodes of some test point t, we say that the reference path ending at r is one of associated reference path of the test path ending at t.

We further introduce a special subtree of the test subtree, called matching subtree. A matching subtree can be formed through pruning the test subtree as the following steps:

Step 1. The main test point is the root of the matching subtree; $P \leftarrow \{\text{the root}\}$.

Step 2. For each US node u in P, at least one subtree rooted by the child of u is retained, the subtrees rooted by other child nodes, if have, are pruned; $P \leftarrow \{\text{the current children of the nodes among } P\}$.

Step 3. If P is null, end the pruning procedure. Otherwise, go to Step 2.

Lemma 1. *Let MT be any subtree of a transfer tree. For any reference path, there exists at least one test path (in MT) which includes the reference path.*

Similar to the UIO sequence (Sabnani and Dahbura, 1988), we also use notation $T(s_i,\alpha)$ to indicate the input/output sequence that brings the machine back to s_i from $\delta(s_i,\alpha)$, where α is an input sequence. We also employ an operator $\sharp$ to represent a concatenation of input/output sequences. Let $A_1, A_2, \dots, A_k$ be k input/output sequences, then the concatenation CO is defined by $CO = \sharp_1^k A_i$ (Sabnani and Dahbura, 1988). The DIO sequence $dios(s_i)$ can be generated in terms of the transfer tree for state s_i. We use notation $\Gamma(s_i)$ to represent the set of TIO sequences (for state s_i) except hidden TIO sequences. Let $\Gamma(s_i) = \{ts_1, ts_2, \dots, ts_k\}$, then $dios(s_i) = (\sharp_1^{k-1} ts_i T(s_i, ts_i)) ts_k$. Particularly, if $|S_{IO}^{+}(s_i)| = 1$, then it is unnecessary to build the transfer tree for s_i, because the transition ending at s_i has no fault under the IO-correct assumption.

When the DIO sequence for a certain state s_i is applied to a tested state, the corresponding output sequence is observed. If the practical output sequence of a test sequence ts in $\Gamma(s_i)$ is equal to the output part of ts, we say that ts is matched. All the test paths (including related nodes and edges) which produces a matched TIO are marked with a symbol $pass$. Additionally, all the test paths which are hidden by these test paths with symbol $pass$ are also marked with symbol $pass$. If at least one test path is marked with symbol $pass$ and all the children of each SS node with symbol $pass$ are also marked with

Table 2

The TT test sequences and the observed outputs

Test sequence	Input	Expected output	Observed output
TIO 1	c, d	$1, 2$	$1, 2$
TIO 2	a, a	$2, 1$	$2, \underline{\circ}$

the symbol, we say that the DIO sequence has an expected output sequence. Under the IO-correct assumption, we have the following theorem.

Theorem 1. *If the DIO sequence for s_i has an expected output sequence, then the tested state is s_i. Otherwise, the the tested state is not s_i.*

Proof. We mark all the matched paths and the paths hidden by these paths with symbol $pass$, then all paths with symbol $pass$ comprise a tree T. Clearly, the root of T and any matching subtree is 0-level node labeled by s_i. We can notice that all the children of a SS node are marked with symbol $pass$. Additionally, at least a child of a US node is marked with symbol $pass$ because the US node is marked with symbol $pass$ only if there is a matched test path which passes the US node. Therefore we can prove that at least one matching subtree can be derived from T. Thus the DIO sequence for s_i has an expected output sequence. For any reference sequence, there exists at least one TIO sequence derived from T which includes the reference sequence, i.e., the DIO sequence for s_i has an expected output sequence if and only if the tested state is s_i. Thus this theorem has been proven.

EXAMPLE 2. Consider a specification DFSM shown in Fig. 2a and its implementation shown in Fig. 2b. In the implementation DFSM, dashed lines represent the IO-correct transfer faults. A transfer tree for state s_1 is shown in Fig. 3. For determining whether or not state s_1 is entered when input b is applied to state s_0, input sequence $dios_i(s_1)$ is applied to the tested state. Table 2 gives the TIO sequences and corresponding output sequences. According the observation, we notice that the 0-level node labeled by s_1 is marked with symbol $pass$, but one child s_5 of the node is not marked with symbol $pass$. Thus we can determine the DIO sequence has no expected output sequence, i.e., transition $s_0 \times b \xrightarrow{1} s_1$ has an IO-correct transfer fault. Note that this example is under the IO-correct assumption.

4. Testing and Diagnosing Method

In this section, we present a two-tier testing and diagnosing method for protocol conformance testing, which includes two fault detection and diagnosis procedures. The method is used to detect and diagnose the possible faults of the implementation which is developed in terms of the specification modeled by deterministic FSM. The lower-tier part of the method is performed in the development procedure, to find the possible output fault, IO-wrong transfer fault and missing/extra state fault through local test, while the upper-tier part tries to use DIO sequences to detect and diagnose the potential IO-correct transfer faults. Through the above hierarchical diagnosis, the proposed method can well locate the found faults.

4.1. *DIO Sequence Searching Algorithm*

The proposed testing and diagnosing method is based on the DIO sequence. As mentioned previously, the DIO sequence for a state can identify the state under the assumption that there is no output fault, IO-wrong transfer fault and missing/extra state fault. Section 3 explains the DIO sequence generation approach based on the transfer tree, this part presents the transfer tree building algorithm, as Algorithm 1 shows.

Given a specification DFSM, the specified transitions is denoted by $e_0, e_1, \ldots, e_{n_e-1}$, where n_e is the number of transitions of the DFSM. Array $M_E[n_e]$ saves the verification status of each transition. The initial value of $M_E[i]$ ($0 \leqslant qi < n_e$) is F. If a transition e_i is proven to be implemented without any fault, then $M_E[i] = T$. More details can be seen in Section 4.2.

In Algorithm 1, equation $comp = T$ holds if and only if the current constructed tree can produce a DIO sequence, i.e., each downstream leaf node of the main reference points is a NS node. Expression $r(p)$ means the SS node (except p) sequence in the path from the root to node p. Denotation $s(c,t)$ means the label state of the tail node of tree edge which starts from c and is labeled by the input/output pair of t. Similarly, $s(c)$ denotes the label state of tree node c. Procedure $Init\,S(B, r(p))$ creates a stack B and then pushes the node of $r(p)$ into B in certain order (i.e., first SS node first enter). The purpose of the $Next\,P(X)$ procedure is to find a new test path, as Algorithm 2 explains. Procedure $Next\,T(c)$ chooses a child a of c, such that a has the most specified inputs among all the children which are not visited for extending the tree. We take the above prior order because it is a good heuristics to reduce the possible tra nsfers. Let d mean the tail node of the tree edge, which starts from c and is labeled by the input/output pair of transition t, then expression $nextss(c,t)$ indicates (1) one random child of d if d is a US node, or (2) d if d is a SS node.

The test path extending procedure $TST(,c,t,A)$ can be depicted by Algorithm 3. In the $RST(v, ref(v,t))$ procedure, $ref(v,t)$ represents the transition that starts from $s(v)$ and has the same input/output of transition t. $RST(v, ref(v,t))$ is similar to $TST(c,t,A)$ except that the former (1) creates new tree nodes and edges in terms of $ref(v,t)$ instead of t, and (2) has no $Push(A,\cdot)$ operation.

Clearly, it is impractical to provide a too long DIO sequence. Therefore the height of the tree produced by Algorithm 1 is limited, as the following theorem shows.

Theorem 2. *The height of the tree produced by Algorithm* 1 *is at most* $2n_s + 2$, *where* n_s *is the number of the specified states.*

Algoritm 1. Build the Transfer Tree for State s_i

```
1:  procedure TTB$M, s_i, M_E[n_e]$
    // $M$ is the specification DFSM.
2:    create a root $r$ labeled by $s_i'$; $\forall s_u$ $(s_u \in S_{IO}^{+}(s_i))$, create a new node labeled by $s_u$ and connect
    the node to the root; $comp = F$;
3:    create a stack $A$ for saving created US nodes;
4:    while $A$ is not null
5:      $c \leftarrow Next\,P(A)$;
6:      $InitS(B, r(c))$;
7:      while $comp = F$ do
8:        if $\neg\exists a((a \in B) \wedge (s(a) = s(c))$ then
9:          $Push(B, c)$;
10:       end if
11:       $t \leftarrow Next\,T(c)$;
12:       if $t \neq null$ then
13:         $TST(c, t, A)$;
14:         for all $v \in RS(c)$ do
15:           $l \leftarrow RST(v, ref(v, t))$;
16:           add $s(l)$ to set $S_n$;
17:         end for
18:         if $S_n = \{s(c, t)\}$ then
19:           $c \leftarrow Next\,P(B)$;
20:         else
21:           $c \leftarrow nextss(c, t)$;
22:         end if
23:       else
24:         $c \leftarrow Next\,P(B)$;
25:       end if
26:       If $c = null$, then return failure.
27:     end while
28:   end while
29:   prune the tree nodes of the test subtree that has no reference node, and prune the edges connecting
    to these nodes.
30: end procedure
```

Proof. According to the algorithm, we can see that the height of the test subtree is limited by stack B. In stack B, the label states of tree nodes are different from each other (see line 8–10). We can also notice that (1) the number of US node is not larger than that of the SS nodes in any test path, and (2) the height of any reference subtree is not larger than that of the test subtree. Additionally, the leaf node and one of its upstream nodes might have the same label state in a test path. Thus the max height of the tree produced by algorithm is $2n_s + 2$.

Algoritm 2. Search New Test Path

```
 1: procedure NextP $X$
    // $X = A$ or $B$, $r$ is the tree root.
 2:   while $X \neq null$ do
 3:     $t \leftarrow top(X)$;
 4:     if $t = r$ then
 5:       Pop(X); return the main test point;
 6:     else if $\exists d((d \in C(t)) \wedge (!vd(d)))$ then
    // $C(t)$ means the set of $t$'s children; $!vd(d)$ indicates that $d$ has not been visited for extending the tree.
 7:       if $d$ is SS node then
 8:         return $d$;
 9:       else
10:         return one random child of $d$;
11:       end if
12:     else
13:       $Pop(X)$;
14:     end if
15:   end while
16:   return null;
17: end procedure
```

Algorithm 3. Extend the Test Path

```
 1: procedure TST$c, t, A)$
    // Assume that $t = s(c) \times x \xrightarrow{y} s_d$, and $t = e_i$.
 2:   if $\neg\exists h((h \in C(c) \wedge (s(h) = s_d))$ then
 3:     if $|S^{+}_{IO}(d)| = 1$ or $M_E[i] = T$ then
 4:       create a SS node $m$ labeled by $s_d$;
 5:       created edge $(c, m)$, labeled by $x/y$;
 6:     else
 7:       create a US node $u$, labeled by $s'_d$;
 8:       created edge $(c, u)$, labeled by $x/y$;
 9:       $Push(A, u)$;
10:       for all $v \in S^{+}_{IO}(s_d)$ do
11:         create a SS node $k$, labeled by $v$;
12:         create edge $(u, k)$;
13:       end for
14:     end if
15:   end if
16: end procedure
```

In a given DFSM $M = (S, X, Y, \delta, \lambda, s_0)$, we use $d_{\max}$ to mean the largest out-degree of the states and $d'_{\max}$ to indicate the largest out-degree of the US nodes, $d'_{\max} = \max\{|S^{+}_{IO}(s)||s \in S\}$. Then we have the following theorem.

Theorem 3. *The worst-case space-complexity of Algorithm* 1 *is* $O((d'_{\max})^2(d_{\max} \cdot d'_{\max})^{n+1})$*, where* n *is the number of specified states.*

Proof. In the test subtree, Algorithm 1 produces at most $(d_{\max} \cdot d'_{\max})^{n+1}$ SS nodes, and the number of US nodes is not larger than that of SS nodes. Therefore there are at most $O((d_{\max} \cdot d'_{\max})^{n+1})$ test nodes. Except the edge starting from the root, edges of the test subtree are produced in terms of the SS nodes, and the edge generation procedure creates at most $d'_{\max} + 1$ test edges. Therefore there are at most $O((d'_{\max} + 1)(d_{\max} \cdot d'_{\max})^{n+1})$ edges in the test subtree. The worst-case space-complexity of building any reference subtree is not larger than that of building the test subtree. Thus the worst-case space-complexity of Algorithm 1 is $O((d'_{\max})^2(d_{\max} \cdot d'_{\max})^{n+1})$.

For a certain state s_i and any transition $s_i \times x \xrightarrow{y} s_j$, if there exists a transition $s_k \times x \xrightarrow{y} s_l$ $(k \neq i)$ such that $IO(s_j) \subseteq IO(s_l)$, then we say that s_k is able to hide s_i. If s_k is able to hide s_i and there is an input/output pair x/y such that the tail states of transition $s_j \times x \xrightarrow{y}$ and $s_l \times x \xrightarrow{y}$ are different from each other, we say that s_i can be further distinguished from s_k. If the input/output sequences in a test path and a reference path are different, we say that the test path can be distinguished from the reference path. Assume that a given state is hidden by another state with probability α, and can be further distinguished from another state with probability β.

Theorem 4. *A test path including* m *edges can be distinguished from one of its associated reference path with probability* $(1-\alpha)\frac{(\alpha\beta)^m - 1}{\alpha\beta - 1}$.

Proof. Suppose that the test path starts from state s_i. If s_i can not be hidden by its associated reference node s_k, then there exists a transition $s_i \times x \xrightarrow{y} s_j$ such that some input/output pair x'/y' is specified for s_j but not for any tail state of transition starting from s_k. Thus input sequence "x, x'" can distinguish s_i from s_k with probability $(1-\alpha)$. Similarly, a test path that includes n test edges is distinguished from the associated reference path with probability $(\alpha\beta)^{n-1}(1-\alpha)$. Note that the test path passing through u is not able to distinguished itself from the associated reference path passing through u's associated reference node v if u and v are labeled by the same state. Thus this theorem has been proven.

4.2. *Fault Detection and Diagnosis Procedures*

Our proposed testing and diagnosing method for protocol conformance testing includes two fault detection and diagnosis procedures at different levels. As mentioned previously, the lower-tier detection and diagnosis procedure is used to satisfy the IO-correct assumption, which is performed in the protocol development procedure. The purpose of the

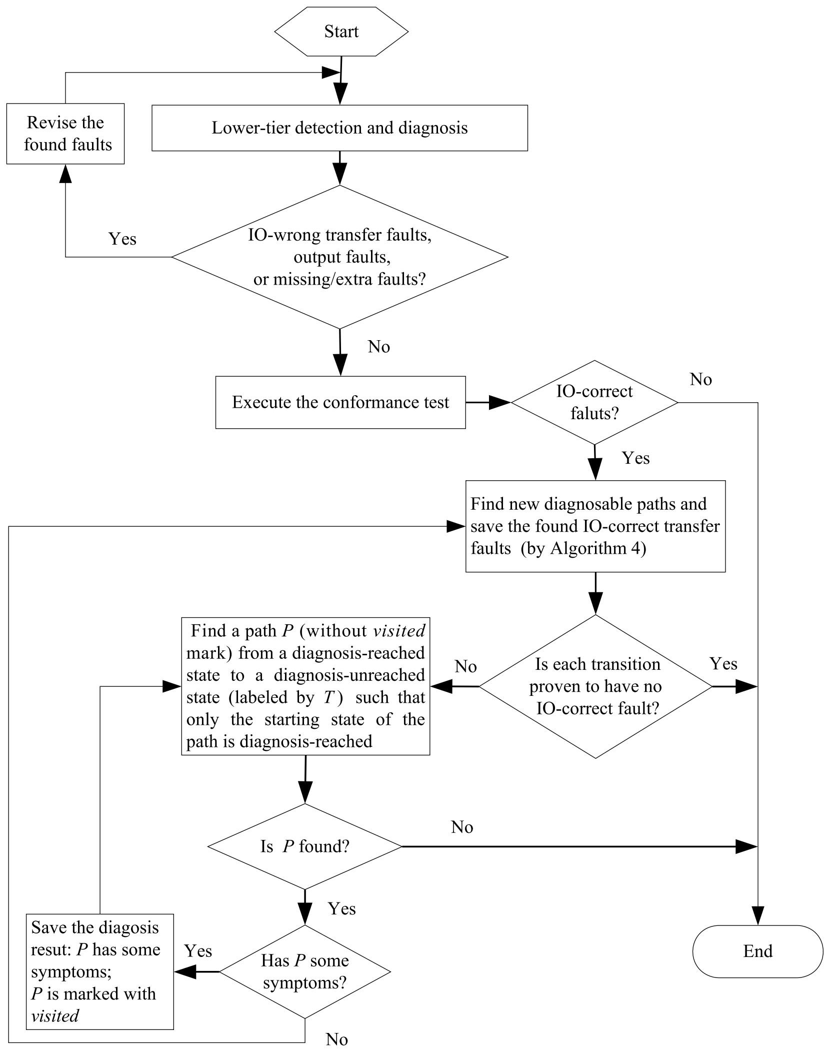
Start
Revise the found faults
Lower-tier detection and diagnosis
Yes
IO-wrong transfer faults, output faults, or missing/extra faults?
No
Execute the conformance test
IO-correct faluts?
No
Yes
Find new diagnosable paths and save the found IO-correct transfer faults (by Algorithm 4)
Find a path *P* (without *visited* mark) from a diagnosis-reached state to a diagnosis-unreached state (labeled by *T*) such that only the starting state of the path is diagnosis-reached
No
Is each transition proven to have no IO-correct fault?
Yes
Is *P* found?
No
Yes
Save the diagosis resut: *P* has some symptoms; *P* is marked with *visited*
Yes
Has *P* some symptoms?
End
No

Fig. 4. The fault detection and diagnosis processes.

upper-tier detection and diagnosis procedure is to detect the potential IO-correct transfer faults of the implementation under test and accurately diagnose the found faults. Fig. 4 describes the two-tier fault detection and diagnosis processes, and we will explain the details in the following parts.

Because the states can be easily identified through messages and communication entities (such as server and client), the lower-tier detection and diagnosis procedure can track each state of the implementation DFSM in the protocol development procedure. The lower-tier detection and diagnosis procedure has the following two objectives:

Objective 1. Ensure that the number of states of the implementation DFSM is equal to that of specified states.

Objective 2. Find and revise the possible output faults and IO-wrong transfer faults.

The first objective can be obtained through counting the different states of the implementation DFSM. The second objective can be gained as follows:

For any specified transition $s_i \times x \xrightarrow{y} s_j$ and specified input/output pair x'/y' for state s_j, an input sequence "x, x'" is applied to the corresponding state (in the implementation) of specified state s_i. If the first observed output is not y, then there is an output fault for the transition. The second observed output is not y' if (1) the tail state of the transition in the implementation is not in $S^{+}_{IO}(s_j)$ and/or (2) the input/output pair x'/y' for state s_j has a wrong implementation. All the found faults will be revised until passing the above test. Because the above tests are within a local area, the found faults can be well located.

REMARK 1. Through the above test and revise procedure, we can say that the implementation has no IO-wrong transfer fault, output fault and missing/extra fault, i.e., satisfy the IO-correct assumption.

As noted above, the DIO sequence can identify a state in the implement under the IO-correct assumption. The upper-tier part of our proposed method employs a test method based on DIO sequences to detect and locate the possible IO-correct transfer faults. Similar to $M_E[n_e]$, array $M_S[i]$ ($0 \leqslant i < n$) is used to indicate whether or not state s_i has a DIO sequence. Specifically, $M_S[i] = T$ if state s_i has a DIO sequence at current moment, while $M_S[i] = F$ if state s_i has no DIO sequence at this moment. State s_i is said to have a T label if $M_S[i] = T$. Next we introduce some related concepts.

DEFINITION 7. In a given DFSM, if there exists a path from the initial state s_0 to state s_i such that each edge e_j in the path is labeled by T or S (i.e., $M_E[j] = T$ or S), then the path is said to be a diagnosable path.

DEFINITION 8. In a given DFSM, a state s_i is said to be diagnosis-reached if there exists a diagnosable path from the initial state s_0 to state s_j. Otherwise, state s_j is said to be diagnosis-unreached.

DEFINITION 9. Let α and β denote the specified input sequence and output sequence of a path (denoted by p) from state s_i to s_j, respectively. We say that path p has the expected output sequence if the output sequence is equal to β when α is applied to state s_i.

We say that a path, from state s_i to state s_j, has no symptom if the expected output sequence is observed and the ending state is s_i when the input sequence of the path is applied in state s_i. If a path has some symptoms, there is at least one fault in the path because the starting state and ending state are both affirmed to be correctly implemented. The path that has some symptoms is said to be a fault-included path. Let transition t be represented by $e_{n(t)}$. Suppose that a specification deterministic FSM M has n_e transitions (denoted by $e_0e_1\ldots e_{n_e-1}$) and n_s specified states (denoted by $s_0s_1\ldots e_{n_s-1}$), then the upper-tier detection and diagnosis procedure does as follows:

Step 1. Perform the conformance test through some existing fault detection method with high fault coverage (by default, the Wp-method). If no fault is found, then end the procedure.

Step 2. Create array $M_E[n_e]$; $\forall i$ $(0 \leqslant i < n_e)$, $M_E[i] \leftarrow F$; Create array $M_S[n_s]$, $\forall i$ $(0 \leqslant i < n_s)$, $M_S[i] \leftarrow F$; Initialize set C_{tf} and C_{pf}, which are used to save the found IO-correct transfer faults and fault-included paths, respectively.

Step 3. Employ Algorithm 4 to attempt to find the new diagnosable paths and save the found IO-correct transfer faults. If $\forall j$ $((0 \leqslant j < n_e) \wedge (M_E[j] = T))$, then end the procedure.

Step 4. Try to find a path from a diagnosis-reached state to a diagnosis-unreached state which is labeled by T such that (1) the path has no symptom and (2) only the starting state of the path is diagnosis-reached. In the searching procedure, if a surveyed path pa (from a diagnosis-reached state to a diagnosis-unreached state which is labeled by T) has some symptoms, then $C_{pf} \leftarrow pa$. If the above path is found, then each edge e_j of the path is labeled by S (i.e., $M_E[j] = S$) and go to Step 3.

Step 5. $C_{sus} = \{e_k | M_E[k] \neq T\} - C_{tf}$, where C_{sus} means a set of the transitions that might have IO-correct transfer faults. All the transitions in set C_{cor} (i.e., $\{e_k | M_E[k] = T\}$) are correctly implemented.

In Algorithm 4, the $DIOSSearch$ procedure generates the possible DIO sequence (for each state s_k) that satisfies $M_S[k] = F$, in terms of $M_E[n_e]$. The DIO sequence generation method can be seen in Section 3. The purpose of the $Validate(t)$ procedure is to validate transition t. The above procedure returns T if (1) $M_E[n_{w,v}] = F$, and (2) transition t has no IO-correct transfer fault. Otherwise, F is returned. To determine whether transition t $(t = s_i \times x \xrightarrow{y} s_j)$ has IO-correct transfer fault or not, input sequence "x, $dios_i(v)$)" is applied to state s_i, and the output sequence is observed. If the output sequence is the expected one, we say that the transition is correct. Otherwise, the transition has a IO-correct transfer fault, related work should be done to revise the fault.

Let T_{pt} be the set of the transitions which are in the paths in set C_{pf}, then $T_{pt} \subseteq C_{sus}$. Additionally, $C_{tf} \bigcap C_{sus} = \Phi$. According to the two-tier fault detection and diagnosis procedures, we have the following theorem.

Theorem 5. *Assume that set C_{sus} produced by the proposed fault detection and diagnosis method is null, then all the faults can be located accurately.*

Proof. Assume that set C_{sus} produced by the proposed fault detection and diagnosis method is null, then all faults can be detected. According to the above description, we can notice that all the found faults occur near the related symptoms (in the lower-tier procedure) or at the tested transition (in the upper-tier procedure). Thus the theorem has been proven.

Additionally, a fault can be located within a limited arrange if the starting state and/or ending state of the transition cannot be identified, because the transitions in set C_{cor} have been confirmed to be correctly implemented.

Algorithm 4. DIO-Based Fault Detection and Diagnosis Procedure

```
1:  **procedure** Diagnose$M, M_S[n_s], M_E[n_e]$
    // $s_0$ is the initial state.
2:    $fin \leftarrow F$;
3:    **while** $fin = F$ **do**
4:      $DIOSSearch(M, M_S[n_s], M_E[n_e])$;
5:      create stack $D$, $Push(D, s_0)$;
6:      create array $V[n]$, $V[i] \leftarrow F$ $(0 \leqslant i < n)$;
7:      $fin \leftarrow T, w \leftarrow s_0$;
8:      **while** $w \neq null$ **do**
9:        $t \leftarrow Next\,(w)$;
10:       **if** $t \neq null$ **then**
11:         **if** $M_E[k] = S$ and $V[v] = F$ **then**
    // $k = n(t)$, $v$ is the tail state of $t$.
12:           $Push(D, v)$; $V[v] = T$; $w \leftarrow v$;
13:         **else if** $M_S[w], M_S[v] = T$ **then**
14:           **if** $Validate(t) = T$ **then**
15:             $M_E[n(t)] = T$; $fin \leftarrow F$
16:           **else**
17:             add $t$ to $C_{tf}$;
18:           **end if**
19:           **if** $V[v] = F$ **then**
20:             $Push(D, v)$;
21:             $V[v] = T$; $w \leftarrow v$;
22:           **end if**
23:     **else**
24:       $w \leftarrow Pop(D)$;
25:     **end if**
26:       **else**
27:         $w \leftarrow Pop(D)$;
28:       **end if**
29:     **end while**
30:   **end while**
31: **end procedure**
```

4.3. *Application Examples*

Given the specification DFSM shown in Fig. 2a and an implementation shown in Fig. 5, we use the proposed testing and diagnosing method to test and diagnose the faults in the implementation.

In the lower-tier detection and diagnosis procedure, the corresponding output sequence "$2, \circ$" is observed when input sequence "e, d" is applied to state s_2. Because "$2, \circ$" is different from the expected output sequence "$2, 2$", transition $s_2 \times e \xrightarrow{2} s_3$ is IO-wrong. Similarly, the procedure also can detect the output fault $e/2$. Note that all found faults in the lower-tier procedure are correctly revised, as Fig. 2b shows.

Next we use the upper-tier detection and diagnosis procedure to test the implementation DFSM shown in the Fig. 2b. The specific steps is explained in Table 3. Note that we do not care about the output sequence β and expected output sequence α, which are in associate with an input sequence $T(,)$ (see Section 3).

We further use 5 practical models to evaluate the performance of our proposed method. These models include: 4 DFSM models for mobile IPv6 described in Zhang *et al.* (2006), i.e., NS, MN, BCE and BC models; 1 DFSM model for IPv6 neighbor discovery

Table 3

The steps of the IO-correct transfer fault detection and diagnosis for the implementation DFSM shown in Fig. 5. Symbol α and β mean the output sequence and expected output sequence corresponding to input sequence $T(,)$, respectively

Transition	Input sequence	Expected output	Observed output	Diagnosis
$s_0 \times c \xrightarrow{2} s_9$	c, a, c	$2, 2, 1$	$2, 2, 1$	pass
$s_9 \times a \xrightarrow{2} s_1$	$a, c, d, T(s_2, (c, d)), a, c$	$2, 1, 2, \alpha, 2, 2$	$2, 1, 2, \beta, 2, \underline{1}$	error
$s_1 \times a \xrightarrow{2} s_5$	–	–	–	pass
$s_5 \times d \xrightarrow{2} s_7$	d, e, a	$1, 1, 1$	$1, 1, 1$	pass
$s_7 \times e \xrightarrow{1} s_5$	–	–	–	pass
$s_5 \times b \xrightarrow{1} s_6$	a, b, e	$1, 1, 1$	$1, 1, 1$	pass
$s_6 \times e \xrightarrow{1} s_7$	e, e, a	$1, 1, 1$	$1, 1, 1$	pass
$s_5 \times b \xrightarrow{2} s_8$	–	–	–	pass
$s_8 \times d \xrightarrow{2} s_7$	d, e, a	$2, 1, 1$	$2, 1, 1$	pass
$s_8 \times c \xrightarrow{1} s_1$	$c, c, d, T(s_2, (c, d)), a, c$	$1, 1, 2, \alpha, 2, 2$	$1, 1, 2, \beta, 2, 2$	pass
$s_9 \times b \xrightarrow{2} s_8$	–	–	–	pass
$s_0 \times b \xrightarrow{1} s_1$	$b, c, d, T(s_2, (c, d)), a, c$	$1, 1, 2, \alpha, 2, 2$	$1, 1, 2, \beta, 2, \underline{1}$	error
$s_0 \times a \xrightarrow{2} s_2$	–	–	–	pass
$s_2 \times a \xrightarrow{2} s_4$	a, e, d	$2, 1, 2$	$2, 1, 2$	pass
$s_2 \times c \xrightarrow{1} s_4$	c, e, d	$1, 1, 2$	$1, 1, 2$	pass
	$C_{sus} = \{(s_4, s_3), (s_3, s_4), (s_2, s_3), (s_1, s_3)\}$			

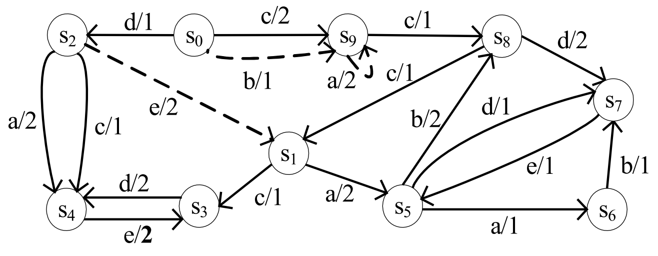

Fig. 5. An example of the implementation.

Table 4

The two-tier fault detection and diagnosis for the multicast tree building procedure of the HCcast protocol (Zhang *et al.*, 2009)

	Fault types in lower-tier part			IO-Correct fault	
	Output fault	IO-Wrong fault	Missing state fault		
SA	1	1	–	1	2
Found faults	5	2	–	3	1

protocol described in Wang *et al.* (2004). Under the IO-correct assumption, each state of 4 DFSM models (i.e., NS, BCE and BC models for mobile IPv6, and the model for IPv6 neighbor discovery protocol) has corresponding DIO sequence, and only 2 states of the MN model (which contains 9 states) have no corresponding DIO sequence. According to the proposed testing and diagnosing method, about 97% potential IO-correct transfer faults of the above DFSM models can be accurately located under the IO-correct assumption. We also applied the two-tier fault detection and diagnosis method in the development procedure of the multicast tree building part of our proposed HCcast protocol. The multicast tree building procedure is the core of the HCcast prot ocol, and the related details can be seen in Zhang *et al.* (2009). Table 4 presents the corresponding fault detection and diagnosis results. In this part, we use suspected arrange (SA) to denote the number of the transitions which a designated found fault might be in. From the table, we can see that the found faults can be accurately diagnosed (i.e., be located in a transition or a small part of all the transitions).

5. Related Work

The deep analysis, especially formal analysis, can effectively improve the system design. For instance, Lunday and Sherali proposed and formulated a multi-objective dynamic network interdiction problem, and further presented a feasible solution of the problem (Lunday and Sherali, 2010). In addition to the effective analysis of the system design, testing is also an indispensable part of developing a high-quality system because it can detect some potential faults of the system implementation.

Testing Finite State Machines is a long-term research subject in hardware and software testing, as well as protocol conformance testing, e.g., Abo *et al.* (1991), Chan *et al.* (1989), Chow (1978), El-Fakih *et al.* (2004), Fujiwara *et al.* (1991), Gonec *et al.* (1970), Naito and Tsunoyama (1981), Petrenko and Yevtushenko (2005), Sabnani and Dahbura (1988), Wang *et al.* (2005) for test derivation methods, and Lai (2002), Lee and Yannakakis (1996) for survey. The test derivation methods for conformance testing can produce effective test sequences to detect the faults of the implementations under test. However, they usually have limited capabilities of diagnosing the faults. Ramalingam *et al.* (1995) analyzed the diagnosis capabilities of some FSM-based test sequence generation methods, including W-method (Chow, 1978), Wp-method (Fujiwara *et al.*, 1991), UIO-method (Sabnani and Dahbura, 1988), UIOv-method (Chan *et al.*, 1989), DS-method

(Gonec, 1970) and T-method (Naito and Tsunoyama, 1981), and claimed that the studied methods have the capability of fault diagnosis when the implementation has only one fault (Ramalingam *et al.*, 1995). Among these methods, the W-method and the Wp-method provide the best resolutions in diagnosing the fault. Additionally, the diagnosis result is a set of transitions which contains the found fault. If an implementation has at most n states and at most one fault, then both W-method and Wp-method locate the fault in a set including more than n transitions.

As mentioned previously, it is difficult to diagnose the faults in terms of the result of conformance test. So far there have been few fault diagnosis methods for the conformance testing based on DFSMs. Ghedamsi *et al.* presented a fault diagnosis method (called SF-method) that can locate single fault in the implementation modeled by DFSMs (Ghedamsi and Bochmann, 1992). The method adapts conflict set to provide the fault information. Ghedamsi *et al.* further proposed a multiple fault diagnosis method (called MF-method) for the case where multiple faults (output and/or transfer fault) might be presented in the transitions of an implementation represented by a deterministic finite state machine (Ghedamsi *et al.*, 1993). The method can provide fault hints through a set of diagnosis, each of which is formed by a set of transitions suspected of being faulty. The above two methods each need additional test to further find the accurate (or somewhat accurate) location of the found fault, and th e corresponding test costs are depended on the results of the initial diagnosis. Vuong and Ko (1990) proposed a method (called CB-method) based on Constraint Satisfaction Problem (CSP). The CSP-based method detects and diagnoses the possible faults of the implementation under test, under the assumption that the specification (1) is minimal, strongly connected and completely specified, and (2) has an upper bound on the number of states. The CSP-based method attempts to locate the faults through enumerating the possible DFSM model of the implementation, which needs considerable additional test sequences.

Table 5

Comparison among some fault detection and/or diagnosis methods and our proposed method

Method	Fault detection capability	Diagnosis capability		Additional test cost	Black-box test
		Accuracy	Coverage		
W-Method	Very high	Very low	Single	None	Yes
Wp-Method	Very high	very low	single	none	yes
UIO-Method	High	Very low	Single	None	Yes
UIOv-Method	High	Very low	Single	None	Yes
DS-Method	High	Very low	Single	None	Yes
SF-Method	–	Low	Single	Low	Yes
MF-Method	–	Low	Large	Low	Yes
CB-Method	High	High	Large	Very high	Yes
Upper-tier part of our method	Very high	High	Large	Low	Yes
Lower-tier part of our method	High	Very high	Large	Low	No

Table 5 presents the comparison among most of the above-mentioned fault detection and/or diagnosis methods and our proposed method. In the table, the coverage measure of the diagnosis capability denotes the number of faults that can be diagnosed, and the additional test represents the test that is used to diagnose the found faults. Note that each of SF-method and MF-method is based on the existing test result obtained by some fault detection method.

6. Conclusions

In this paper, we classified the transfer faults into two categories, i.e., IO-wrong transfer faults and IO-correct transfer faults. For accurately diagnosing the possible faults, we proposed a diagnosable input/output (DIO) sequence. Under the IO-correct assumption, the DIO sequence for a state of the DFSM model can differentiate the state from other states even if the implementation might have some IO-correct transfer faults. Based on the DIO sequence, we proposed a two-tier fault detection and diagnosis method for protocol conformance testing. The lower-tier fault detection and diagnosis procedure is performed in the development procedure, to find the possible output faults, IO-wrong transfer faults and missing/extra state faults. The implementation past the lower-tier test satisfies the IO-correct assumption. Therefore the upper-tier part of our proposed method can accurately locate the IO-correct transfer fault if the starting state and ending state of the transition can be identi fied. Additionally, the faults which are not accurately diagnosed can be located within a limited area.

Acknowledgments. This work was supported by the National Natural Science Foundation of China under Grant No. 61070039, the National Basic Research Program of China under Grant No. 2009CB320502, and the Fund for the Doctoral Research Foundation of Shandong Academy of Sciences (in China) under Grant No. 2010-12. The authors would like to thank the executive editor and anonymous referees for their helpful comments on the revisions of this paper.

References

Aho, A.V., Dahbura, A.T., Lee, D., Uyar, M.U. (1991). An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours. *IEEE Transactions on Communications*, 39(11), 1604–1615.

Chan, Y.L., Vuong, S.T., Ito, M.R. (1989). An improved protocol test generation procedure based on UIOS. In: *Proceedings of ACM SIGCOMM'89*, pp. 283–294.

Chow, T.S. (1978). Testing software design modelled by finite-state machines. *IEEE Transactions on Software Engineering*, 3(4), 178–187.

El-Fakih, K., Yevtushenko, N., Bochmann, G.V. (2004). FSM-based incremental conformance testing methods. *IEEE Transition on Software Engineering*, 30(7), 425–436.

Fujiwara, S., Bochmann, G.V., Khendek, F., Amalou, M., Ghedamsi, A. (1991). Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6), 591–603.

Ghedamsi, A., Bochmann, G.V. (1992). Test result analysis and diagnostics for finite state machines. In: *Proceedings of the 12th IEEE International Conference on Distributed Computing Systems*, pp. 244–251.

Ghedamsi, A., Bochmann, G.V., Dssouli, R. (1993). Multiple fault diagnostics for finite state machines. In: *Proceedings of INFOCOM'93*, pp. 782–791.

Gill, A. (1962). *Introduction to the Theory of Finite-State Machines*. McGraw Hill, New York.

Gonec, G. (1970). A method for the design of fault-detection experiment. *IEEE Transaction on Computer*, 19, 551–558.

Lai, R. (2002). A survey of communication protocol testing. *The Journal of Systems and Software*, 62(1), 21–46.

Lee, D., Yannakakis, M. (1996). Principles and methods of testing finite state machines – a survey. In: *Proceedings of the IEEE*, pp. 1090–1123.

Liao, C.H., Wang, C.T., Chen, H.C. (2010). An improved securer and efficient nonce-based authentication scheme with token-update. *Informatica*, 21(3), 349–359.

Lunday, B.J., Sherali, H.D. (2010). A dynamic network interdiction problem. *Informatica*, 21(4), 553–574.

Naito, S., Tsunoyama, M. (1981). Fault detection for sequential machines by transition tours. In: *Proceedings of the 11th IEEE Fault Tolerant Computing Symposium*, pp. 238–243.

Petrenko, A., Yevtushenko, N. (2005). Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers*, 54(9), 1154–1165.

Ramalingam, T., Das, A., Thulasiraman, K. (1995). Fault detection and diagnosis capabilities of test sequence selection methods based on the FSM model. *Computer Communications*, 18(2), 113–122.

Sabnani, K.K., Dahbura, A.T. (1988). A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15(4), 285–297.

Tseng, Y.M., Wu, T.Y. (2010). Analysis and improvement on a contributory group key exchange protocol based on the Diffie–Hellman technique. *Informatica*, 21(2), 247–258.

Vuong, S.T., Ko, K.C. (1990). Novel approach to protocol test sequence generation. In: *Proceedings of Global Telecommunications Conference*, pp. 1880–1884.

Wang, Z., Yin, X., Wang, H., Wu, J. (2004). Automatic testing of neighbor discovery protocol based on FSM and TTCN. In: *Proceedings of the 10th Asia–Pacific Conference on Communications*, pp. 805–809.

Wang, J., Xiao, J., Lam, C.P., Li, H. (2005). A bipartite graph approach to generate optimal test sequences for protocol conformance testing using the Wp-method. In: *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC2005)*, pp. 307–314.

Yoon, E., Yoo, K.Y. (2010). An entire chaos-based biometric remote user authentication scheme on tokens without using password. *Informatica*, 21(4), 627–637.

Zhang, X., Li, X., Luo, W., Yan, B. (2009). An application layer multicast approach based on topology-aware clustering. *Computer Communications*, 32(6), 1095–1103.

Zhang, Y., Li, Z. (2006). Hierarchical protocol description and test generation method for mobile IPv6 testing. In: *Proceedings of IEEE Conference on Communications (ICC)*, pp. 1959–1964.

X. Zhang received the PhD degree from the Computer Network Information Center of Chinese Academy of Sciences, and the MS degree from the Shandong University of Science and Technology, China. Presently, he is working at the Shandong Computer Science Center, Shandong Academy of Sciences, China. His research interests include network protocols and architectures, multicasting, and software testing. He has over 20 papers in research journals and international conference proceedings in these areas.

M. Yang received the MS degree from the Shandong University, China. Now, she is the associate technology officer and professor of the Shandong Computer Science Center of Shandong Academy of Sciences, China. In the recent years, she has received the scientific and technological progress award four times. Her research interests include software engineering and computer networks.

G. Geng is a research sssistant at the Computer Network Information Center, Chinese Academy of Sciences. He obtained PhD degree in pattern recognition and intelligent system from Institute of Automation, Chinese Academy of Sciences. At present, he is interested in machine learning, adversarial information retrieval on the Web, and Web search.

W. Luo is an associate professor at the Computer Network Information Center, Chinese Academy of Sciences. He obtained PhD degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences in 2001. His current research interests include computer networks, modeling, and mobile computing. He has co-authored over 30 papers in research journals and international conference proceedings in these areas.

DFSM protokolo atitikimo testavimo ir diagnozės metodas

Xinchang ZHANG, Meihong YANG, Guanggang GENG, Wanming LUO

Daugelis žinomų protokolo testavimo metodų gali efektyviai surasti galimus protokolo defektus. Tačiau sunku surastus defektus susieti su gautais testavimo rezultatais ir juos lokalizuoti. Straipsnyje pasiūlyta naudoti diagnozuojamą įėjimo/išėjimo (I/O) seką, kurios pagalba įmanoma atskirti vieną protokolo būseną nuo kitos. Pasiūlytas dviejų procedūrų protokolo atitikimo ir testavimo metodas besiremiantis I/O sekomis. Šis metodas gali efektyviai surasti ir lokalizuoti galimus protokolo realizavimo defektus.