Fault Identification in Networks Using a CFSM Model by Passive Testing

Raymond E. Miller Khaled A. Arisha
Department of Computer Science
University of Maryland, College Park, MD 20742

Abstract

In this paper, we employ the Communicating finite state machine (CFSM) model for networks to investigate fault identification using passive testing. First, we introduce the concept of passive testing. Then, we introduce the CFSM model and the observer model with necessary assumptions and justification. We introduce the fault model and the fault detection algorithm using passive testing. Extending our previous work, we develop the new approach for fault identification based on the CFSM model. A 2-node model example is given to illustrate our approach. Then, we illustrate the effectiveness of our new technique through simulation of practical protocol examples, covering both the 2-node and 3-node models. Finally future extensions and potential trends are discussed.

1 Introduction

Due to the rapid growth in Telecommunication networks and the fast evolution in technology, the need for a more efficient and effective network management approach becomes more urgent. The International Standard Organization (ISO) has defined network management for the Open System Interconnection (OSI) seven-layer model in terms of five functional areas: fault management, configuration management, accounting management, performance management, and security management [7]. A considerable effort has been made to standardize network management protocols and develop network management systems, such as the Simple Network Management Protocol (SNMP) and the Common Management Information Protocol (CMIP) [6]. However, there is much to be done towards formally specifying problems in network management and developing formal techniques to solve these problems. Our work models the network using the formal approaches of Finite State Machine (FSM) as is done in [1] and of CFSM as in [2][3], however we consider the fault identification problem rather than the fault detection or the fault location problems considered in previous papers.

The work presented here focuses on one critical functional area of network management; namely fault management. There are two approaches to test a network for fault management: active testing and passive testing. The most commonly used approach for fault management is active testing, which gathers information actively. By “actively” we mean injecting test messages into the network to aid in finding network faults. In addition to active testing checking for dead links and nodes, active testing has techniques in common with conformance testing of protocols. Conformance testing is used to test protocols off-line to insure that a protocol implementation conforms to its specification. Test sequences are generated from the specification. These input sequences are applied to the implementation to see whether the produced output sequence matches the expected one given by the specification. In contrast, fault management for networks takes place while the network is in use. Because of this, it is desired to keep testing traffic overhead to a minimum. Passive testing simply observes the normal traffic of the network, without adding any test messages. Thus using passive testing enables examining the input-output behavior without forcing the network to any test input sequences. As will be discussed here, quite a bit of fault management can be accomplished using passive testing.

Fault management usually covers the following aspects: detection, location, identification, coverage and correction. The main objective of this research is to see how much fault management information we can obtain using passive testing only. The simplest approaches to passive testing use a FSM specification to model the behavior of the network. Given an implementation of the network under test, it is viewed as a black box where only the input-output behavior is observable. The problem is to determine whether the behavior of the implementation conforms to the behavior of the specification. If it does not conform, this implies the existence of a fault.

Lee et al [1] apply passive testing on a FSM model of the network for fault detection. Their paper demonstrates effective fault detection capabilities of passive testing based on observation of the input/output sequence of the implementation. However, due to the limitation of the single FSM model, no fault isolation or fault location was possible. In [2][3] Miller presents a variant of the CFSM model to specify a network. Using this model he showed that some fault location information could be deduced. As noted above, using passive testing to detect faults eliminates testing overhead normally encountered by other methods that inject special test messages into a network. Of course fault detection is not sufficient. Once a fault is detected, other remedial steps are required to eliminate the fault. Fault location helps by isolating the corrective actions to only a portion of the network. Thus, additional fault location capability by passive testing would be very useful if faults could be isolated to ever-smaller regions. Additionally, if the exact fault that occurred could be limited to a small set of possibilities, this would further simplify the corrective activities. It is precisely this greater fault location and identification capability by passive testing that is being considered here. Miller and Arisha [4][5] demonstrated that better fault location is possible. Here we consider fault identification. Miller and Arisha [8] introduced fault identification based on FSM model. In this paper, we describe the approach used to generalize the fault identification technique to the CFSM model.

We describe the CFSM based specification model and the observer model in section 2 of this paper, with necessary assumptions and justifications. In section 3, we introduce the fault model and give a brief description of the fault detection algorithm using passive testing. Section 4 describes briefly previous work of fault identification for a single FSM model. Then, the new approach for the CFSM model is introduced. It also illustrates the algorithms through a 2-node model example. In section 5, we present experiments modeling practical network protocols for both the 2-node and the 3-node cases. Then, simulation results are given to demonstrate the effectiveness of our approach. Finally, conclusions and possible extensions are discussed in section 6.

2 The Model

In this section we introduce the CFSM model for network specification and the observer model. First, the FSM based model is presented as a description of the single node structure of the CFSM, together with associated assumptions and justifications for the model. Then, the CFSM model is introduced. Finally the observer model is described with assumptions for the whole model.

2.1 The Node model

A single node is modeled as a deterministic finite state machine (DFSM) *M. M* is a six-tuple:

- $M = (I, O, S, s_0, \delta, \lambda)$ where:
- *I*, *O*, and *S* are finite non-empty sets of input symbols, output symbols, and states respectively.
- s_0 is a designated initial state.
- $\delta: S \times I \rightarrow S$ is the state transition function;
- $\lambda: S \times I \rightarrow O$ is the output function.
- When the machine is in state *s* in *S* and receives an input *a* in *I*, it moves to the next state specified by $\delta(s, a)$ and produces an output given by $\lambda(s, a)$.

We denote the number of states, inputs, and outputs by $n = |S|$, $p = |I|$, *and* $q = |O|$, respectively. Also the definition for the transition function δ and the output function λ can be extended from input symbols to strings as well. Starting from initial state s_0, an input sequence $x = a_0,a_1,\ldots,a_k$ takes the machine successively to states $s_{i+1}= \delta(s_i, a_i)$, $i=0,1,\ldots,k$, with the final state $\delta(s_0, x) = s_{k+1}$, and produces an output sequence $y=\lambda(s_0, x)=b_0,\ldots,b_k$, where $b_i=\lambda(s_i,a_i)$, $i=0,1,\ldots,k$.

Assumptions and justifications for the FSM model

A) Single Fault

We assume that if a fault occurs, only one fault occurs during a test cycle. This assumption is important since multiple faults can cause complications such as the hiding of faults. These complications make fault detection by passive testing either more complicated or even impossible. The single fault assumption can be justified however, since we are not assuming a system has frequent faults. If this is so, an effective approach would be to have the one fault in the network detected and corrected before the second fault occurs.

B) Complete Machines

If we define for each state all the possible combinations of inputs on incoming channels, this may lead to a combinatorial explosion. Instead, we show only transitions that would actually take place during correctly specified operation. For all those not specified, a fault should be detected. So all unspecified transitions will lead to an implicitly defined additional fault state with a new output called *"f"* to indicate "fault." This fault state is not an "original state" in the specification; it is used only to allow us to assume that the machines are completely specified. However, in the actual passive testing process, if this faulty state is reached, this would imply that the implementation machine could not produce the observed input/output sequence if it is a correct implementation of the specification.

C) Deterministic Machines

Non-determinism sometimes comes from the lack of complete information during the specification phase. We are assuming here that all "necessary" information is available to indicate deterministically the behavior of the machine. Sometimes non-determinism is introduced in the specification machine to allow different options to be chosen during implementation. We are assuming specific options are chosen in our FSM specification insuring that it is a DFSM.

2.2 The CFSM model

Our model here is based on the node model of DFSM as described above. Representing a huge network by a single DFSM would result in a very large machine, whereas using a machine for each node provides a distributed representation with each machine being relatively simple. So, we choose to propose a variant of the Communicating Finite State Machines (CFSM), where the network is modeled as a set of machines, one for each node of the network, with channels connecting these nodes [9]. This variant uses the Mealy model formulation rather than the send/receive labeling of transitions which is used in the original CFSM model, that is, here we have input/output labeling on transitions.

A CFSM consists of a set of machines M, and a set of channels C. We specify our network $N=(M, C)$, where

$M= \{m_1,m_2,\ldots,m_r\}$ is a finite set of r machines, and $C=\{C_{ij}: i,j \le r \wedge i \neq j\}$ is a finite set of channels,

- For $m \in M$, we define the deterministic finite state machine (DFSM) m as a six-tuple; $m=(I, O, S, s_0, \delta, \lambda)$, as defined in section 2.1.
- Each $C_{ij} \in C$ represents a communication channel from m_i to m_j. It behaves as a FIFO queue with m_j taking inputs from the head of the queue and m_i placing outputs into the tail of this queue for messages produced by m_i that are intended for m_j.

According to our completeness assumption, we are assuming that the implementation machine has a transition from every state for every input symbol $i \in I$. We define also a set of fault states $\{F^i\}$ where each F^i defines for each machine m_i a common destination state for each additional transition (whose output label $\in\{f^{ij}\}$). More detail about the completeness assumption implementation can be found in [4][5].

2.3 The Observer model

Each observer will be placed at a certain node in the network. Let *A* represent a machine specification at a node where the observer is placed. The observer is assumed to know the structure of *A*, so it can trace the input/output tuples observed with the specified state transitions of *A*. For the implementation machine *B* the observer sees the input/output behavior of the FSM representing this node as a black box, and the observer compares *B*'s input/output sequence with the specified sequence of *A*.

Assumptions

1) We assume that the network topology of the implementation is the same as the specification.
2) When more than one node of the network has an observer, we assume that there is some way to gather the information from these observers for fault analysis. Various ways to envisage such a central observer exist, but we do not dwell on these issues here. In particular, for the two-node and three-node networks we will be considering here, we will be using only one observer.
3) The placement of an observer at a node allows it to see inputs and outputs but not other local information like the states of the machine at that node. That is, the node is viewed as a black box FSM.
4) The local observer complexity is limited to the complexity of the FSM representing the node observed. The central observer complexity will vary depending on how much information needs to be sent from local observers.

3 The Fault Model

In this section we introduce the fault model together with a broad overview of the fault detection procedure. The fault model describes the assumption about fault types expected in the network. The fault detection procedure discussed in this section briefly covers the algorithm mentioned in [1].

3.1 Fault Types

Due to our assumptions of the CFSM model used in passive testing, the two types of faults that we can investigate, in terms of the CFSM specification, are:
1) Output Fault: This occurs when a transition has the same head and tail states and the same input as in the specification FSM, but the output is altered.
2) Tail State Fault: This occurs when a transition has the same head state and input/output symbols as specified, but the tail state is altered.

3.2 Fault Detection Overview

As described in [1], the fault detection capability of passive testing can be summarized as follows:
We compare the observed input/output sequence of the implementation machine *B* with the expected behavior of the specification machine *A*. *B* is considered "faulty" if its behavior is *different* than that of *A*. That is, there is no state in *A* that would display the input/output sequence observed from *B*. Here we consider only Observational equivalence. We do not consider any structural isomorphism or equivalence by bi-simulation since all we can do is observe the input/output sequence from *B* and we do not know the structure of *B*.

We define the observer as the entity possessing passive testing fault management functionality. The fault detection procedure can be described as follows:

a) Since we do not know the state of *B* when the observation starts. We assume that *B* could be in any state representing a state of *A*. Let L^0 designate this initial set of possible states.
b) Once the observer observes the first input (i_1) together with the corresponding output (o_1), transitions in *A* lead from states of L^0 to another set of possible states L^1.

c) As each input/output pair (i_j/o_j) is observed, a new set L^j is produced similarly from the set L^{j-1}. The sequence L^0, L^1, …, L^{j-1}, L^j is formed such that, with every observation (i_j/o_j), only the subset of states from L^{j-1} that can still accept an input i_j and produce an output o_j will survive, and they are going to be replaced by their successor states after applying the transition (i_j/o_j). The $\{L^i\}$ sequence is monotonically non-increasing in size, due to the assumption that the machine is deterministic.

d) As the *i/o* sequence progresses, more observed input/outputs will join the observation sequence $e=i_1/o_1$, i_2/o_2, … i_{j-1}/o_{j-1}, i_j/o_j. Consequently, the sequence of sets of possible states $L=L^0$, L^1,…, L^{j-1}, L^j is obtained

e) At some point of observing the input/output sequence, the set of possible current states can lead to a singleton or to an empty set. In case of a singleton set L^k, all subsequent L^j, where $j>k$, will be a singleton sets, or become empty. This process of obtaining a singleton set is called the *passive homing sequence*. And from that point on the current state is determined exactly.

f) As long as L^k is not an empty set, the observation is not contradicting the expected behavior of the specification machine. In this case passive testing concludes that there is no fault yet observed. The implementation machine behavior conforms to that of the specification up to that point.

g) If set L^k becomes empty, this indicates a fault in the implementation. There is no state in *A* that could produce the observed *i/o* sequence *e*. Assuming that the implementation *B* is deterministic and complete, it will end up in some state and be able to continue, but no such state continuation is specified in *A*.

An example of a FSM model and the passive testing fault detection algorithm is shown in Figure 1- where *x* is the observed input/output sequence.

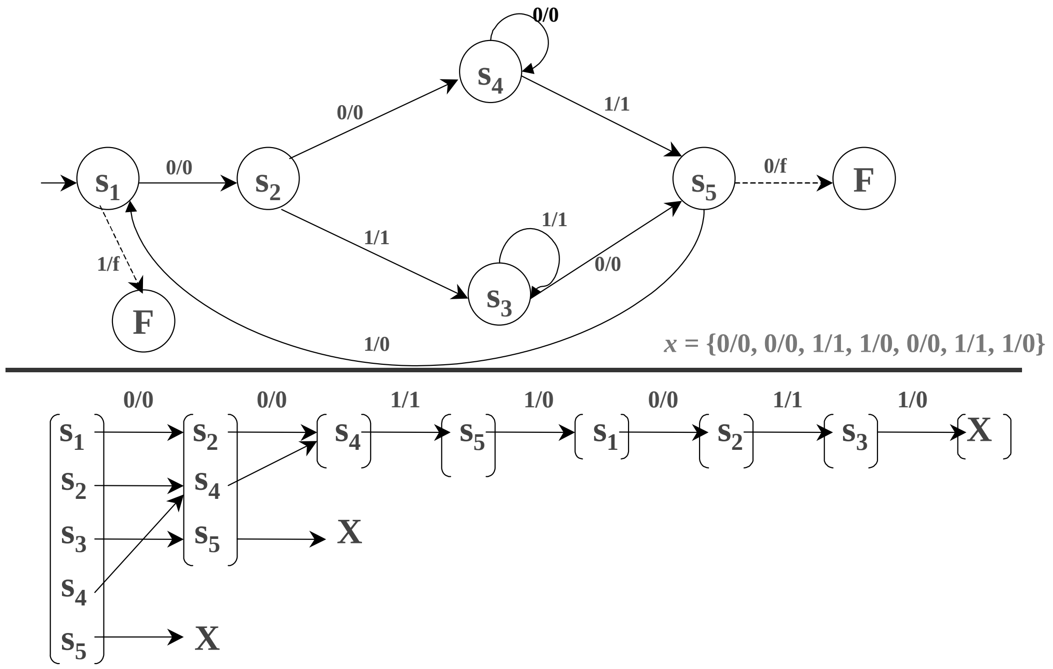


Figure 1: an example FSM for the specification

The detailed algorithm that describes the above procedure is in [1] together with extensions to the algorithm to recover from missing at most *k* observations. Other extensions dealing with unobservable transitions and non-deterministic machine specifications are also provided in [1].

4 Fault Identification

This section covers the fault identification approach for the single FSM model as described in [8]. Then, it describes how to generalize it to the new fault identification technique for the CFSM model. It also gives illustrative examples demonstrating the proposed technique on a 2-node model.

4.1 The Fault Identification for the single FSM model

In section 3.2 we discussed how we obtained the sequence of sets L^0, L^1, …, L^{j-1}, L^j from the observed input/output sequence. Now, let us assume we have an observed input/output sequence i_1/o_1, i_2/o_2,…i_{k-1}/o_{k-1}, i_k/o_k and the resulting sequence of sets L^0, L^1, …, L^{k-1}, L^k where $L^k=\phi$ and $L^{k-1}\neq\phi$. That is, at step k we have just detected that a fault has occurred. We will call this process "forward trace" since it can be computed step-by-step as each input/output pair is observed. Now, for fault identification purposes we analyze this input/output sequence, in terms of the specification, by another process that we call the "backward trace", to produce a second sequence of sets of states.

1) We let $(L^k)^R$ be the set of all states of A.
2) In a backward manner we form set $(L^{j-1})^R$ from $(L^j)^R$ as follows: $(L^{j-1})^R$ contains all states that are head states of transitions with input/output i_j/o_j with tail states being members of $(L^j)^R$.

Output Fault identification:
Theorem 1: If L^j has a state s_p that under i_{j+1} has an output $\neq o_{j+1}$ and $(L^{j+1})^R$ has $\delta(s_p, i_{j+1})$ as an element, then the output fault $s_p \xrightarrow{(i_{j+1}/o_{j+1})} \delta(s_p, i_{j+1})$ could have occurred.
Proof: is given in [8].

Tail State Fault Identification:
Theorem 2: If L^j has a state s_p with transition $s_p \xrightarrow{(i_{j+1}/o_{j+1})} s_q$ and there is a s_r in $(L^{j+1})^R$, then $s_p \xrightarrow{(i_{j+1}/o_{j+1})} s_r$ is a tail state fault that could have occurred.
Proof: is given in [8].

Example

Using the same FSM specification as shown in figure 1, we assume an observed input/output sequence: {*0/0, 1/1, 1/0, 0/0, 1/0*}. The forward and backward traces are shown in figure 2 along with "crossovers" shown by dotted arrows.

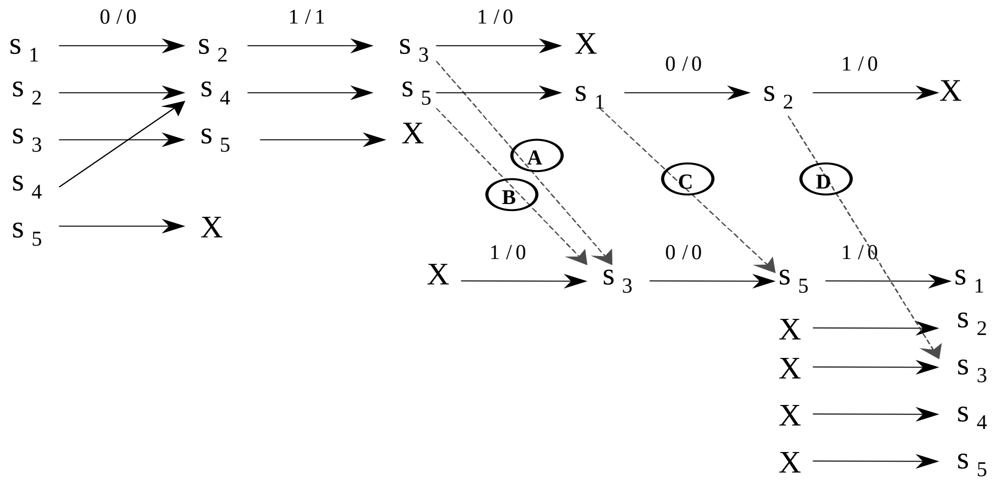


Figure 2: Example of the Forward and Backward Traces

The four "crossovers" arrows are applications of theorems 2 and 3 as described below:

A) Applying theorem 2 we see that this crossover depicts an output fault of transition $s_3 \xrightarrow{(1/1)} s_3$ changing to $s_3 \xrightarrow{(1/0)} s_3$.
B) Applying theorem 3 we see that this crossover depicts a tail-state fault of transition $s_5 \xrightarrow{(1/0)} s_1$ changing to $s_5 \xrightarrow{(1/0)} s_3$.
C) Applying theorem 3 we see that this crossover depicts a tail-state fault of transition $s_1 \xrightarrow{(0/0)} s_2$ changing to $s_1 \xrightarrow{(0/0)} s_5$.
D) Applying theorem 2 we see that this crossover depicts an output fault of transition $s_2 \xrightarrow{(1/1)} s_3$ changing to $s_2 \xrightarrow{(1/0)} s_3$.

This example should provide insight over how single faults that could possibly occur, and would cause the implementation to produce that observed input/output sequence, can be found using the forward and backward traces along with crossovers. The algorithm is thus:

Forward/Backward Crossover Algorithm

1. Do the forward trace analysis for the observed input/output sequence, letting k be the least k such that $L^k=\phi$.
2. Do the backward trace analysis for the observed input/output sequence. Note: This can only be done after the complete input/output sequence has occurred.
3. Add crossover arrows by applying theorems 2 and 3.

Output faults (theorem 1) can arise in this analysis from states, that under some observed input/output have no next state (i.e. an X) in the forward trace analysis. In our example we found two such cases where the current tail state for the transition appeared in the backward analysis at the next step in the input/output sequence. On the other hand, tail state faults (Theorem 2) can arise from states in the forward trace analysis that have next states in the forward trace, but whose faulty next states appear in the next step of the backward analysis. More detail about fault identification technique is given in [8].

4.2 The Fault Identification for the CFSM model

For simplicity, we start with a two-node CFSM model as illustrated in the figure below. An observer is located at machine m_1.

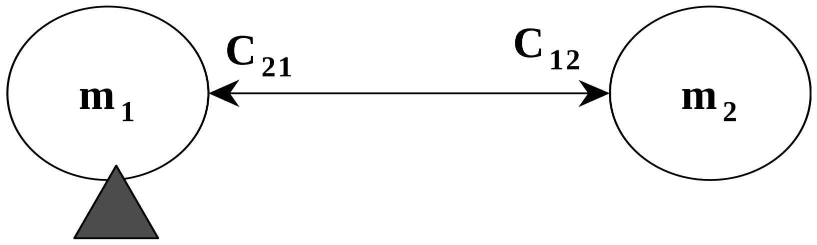


Figure 3 A two-node model

Assumption Only a single fault exists on the network.

Assumption Faults in the nodes are persistent, while faults in the channels are non-persistent. Once a channel has created a fault in a message, that faulty message remains faulty until the receiving machine reads it. However, the channel does not necessarily cause the same fault on a subsequent transmission of the message.

The approach can be described as follows:

1. Fault Detection:

Applying the fault detection technique of passive testing, as in 3.2, the analysis of the observed input/output sequence at m_1 can detect the fault.

2. Fault Location:

Referring to the fault location work on the two-node model done by Miller [2][3], the detected fault can be characterized as follows:

Type A fault:	$o_k^1 \varepsilon O^1$ of $m_1 \rightarrow$ the fault is in m_1.
Type B$_a$ fault:	$o_k^1 = f^{12} \wedge$ no $s^1 \in L^{k-1}$ has $\lambda(s^1, i_k^1)=f^{12} \rightarrow$ the fault is in m_1.
Type B$_b$ fault:	$o_k^1 = f^{12} \wedge \exists\, s^1 \in L^{k-1}$ has $\lambda(s^1, i_k^1)=f^{12} \rightarrow$ the fault is in m_1 or outside m_1.
Type B$_c$ fault:	$o_k^1 \notin O^1 \wedge o_k^1 \neq f^{12} \rightarrow$ the fault is in m_1.
Type C$_a$ fault:	$i_k^1 \notin I \wedge i_k^1 \neq f^{21} \rightarrow$ the fault is outside m_1.
Type C$_b$ fault:	$i_k^1 \notin I^1 \wedge i_k^1 = f^{21} \rightarrow$ the fault is in m_1 or outside m_1.

Fault type B$_c$ is first appended to the fault characterization in [4][5].

3. Fault Identification:

3.1. Fault Identification in m_1:

If by using the results of the above characterization, the fault is determined to be of type *A* or B_a, where the fault is located in m_1, then the fault identification procedure for single FSM model on machine m_1, as described in 4.1 provides the result. This analysis will lead to potential output faults and/or tail-state faults in m_1.

For other types of faults -namely faults of type B_b, C_a and C_b, the fault can be located outside m_1. A further analysis is needed for machine m_2 and both channels C_{12} and C_{21}.

3.2. Fault Identification in m_2:

For machine m_2, we need to extract its expected input/output sequence from the input/output sequence observed at m_1. Since we are assuming only one single fault in the system, and since in this phase we are analyzing potential faults in m_2, machine m_1 and both channels C_{12} and C_{21} are assumed fault free. If we denote the observed input/output sequence at m_1 as $e^1=i_1/o_1,\ i_2/o_2,\dots,\ i_{k-1}/o_{k-1},\ i_k/o_k$, then the expected input/output sequence of machine m_2 should be $e^2=o_1/i_2,\ o_2/i_3,\ \dots,\ o_{k-2}/i_{k-1},\ o_{k-1}/i_k$.

$$i_1/\underbrace{o_1 \quad i_2}_{}/\underbrace{o_2 \quad i_3}_{}/\underbrace{o_3 \quad i_4}_{}/o_4 \quad \dots \quad i_{k-2}/\underbrace{o_{k-2} \quad i_{k-1}}_{}/\underbrace{o_{k-1} \quad i_k}_{}/o_k$$

This can be expressed as: If $e^1=\{\ i_j^1 / o_j^1 \mid j=1,\dots,k\}$ then $e^2=\{\ i_j^2 / o_j^2 \mid j=1,\dots,k-1 \text{ and } i_j^2=o_j^1 \text{ and } o_j^2=i_{j+1}^1\ \}$. Now, we have the expected input/output sequence at node m_2. Applying the fault identification procedure for a single FSM model on machine m_2 only, as described in 4.1. This analysis will lead to potential output faults and/or tail-state faults. Of course, this assumed that the fault was in m_2 and it could have been in C_{12} or C_{21} instead. Thus we have to look at these possibilities also.

3.3. Fault Identification in channels C_{12} and C_{21}:

To analyze potential faults in the channels, according to the single fault assumption, both nodes are assumed fault free for this phase of the analysis. Our approach here is to use the Message Sequence Chart (MSC) to illustrate the scenario of the exchanged symbols over the channels.

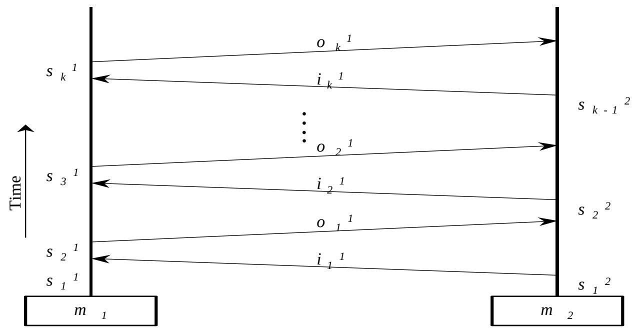


Figure 4 An example for MSC

The analysis goes in the backward direction, i.e. the most recent symbol first. The procedure -that is applied to each symbol- can be described as follows:

- For the symbol i_j^1 over the channel C_{21}, i.e. input to m_1, we check whether this symbol i_j^1 can be a result of alteration of an original symbol o_{j-1}^2 coming from m_2, due to a fault in C_{21}, provided that this alteration will cause the same observed input/output sub-sequence $e^{1*}=\{\ i_t^1 / o_t^1 \mid t=j,\ldots,k\}$ to occur.
- For the symbol o_j^1 over the channel C_{12}, i.e. output from m_1, we check whether this symbol o_j^1 can experience an alteration to some other symbol i_j^2 received by m_2, due to a fault in C_{12}, provided that this alteration will cause the same observed input/output sub-sequence $e^{1**}=\{\ i_t^1 / o_t^1 \mid t=j+1,\ldots,k\}$ to occur.

Notice that we always keep as a reference the observed (m_1 side) input/output symbols and try to assume alteration to occur during transmission over the channels.

Example

For the CFSM model shown below, we consider some examples of the analysis described above to illustrate our approach. This example is a simplified version of the network layer of a connection oriented protocol such as X.25.

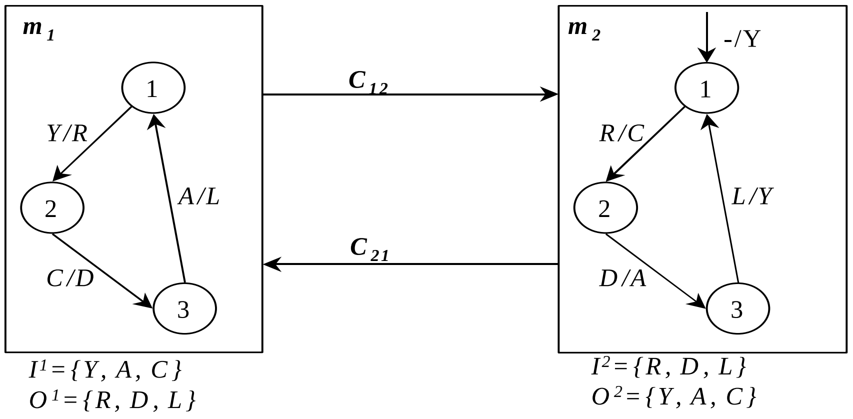
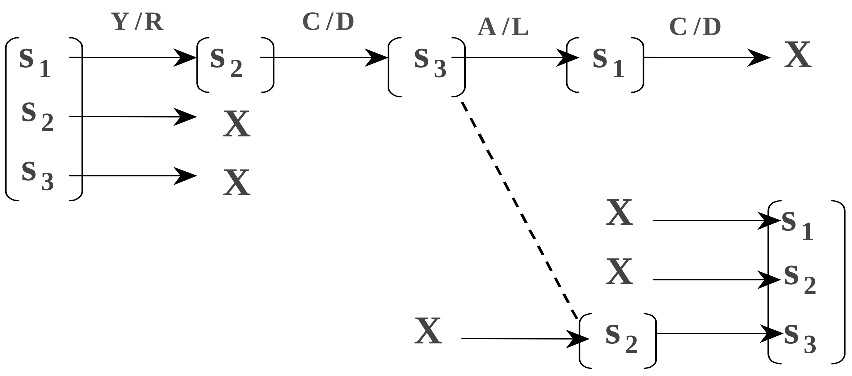


Figure 5 CFSM for Example I

Where the symbol notation is as follows:
Y: Ready (initialized), R: Call Request, C: Call Accept, D: Data, A: Acknowledge, and L: Clear (terminate the call).

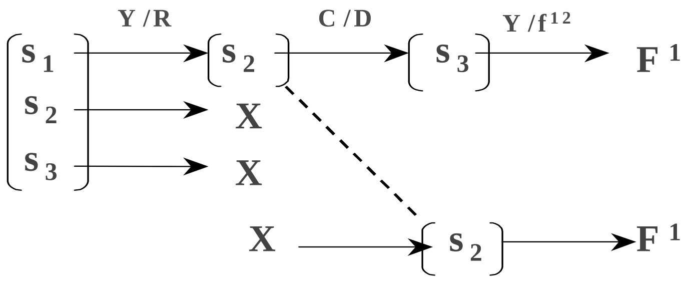
<u>Case 1:</u>
Assume the observed input/output sequence was as follows: {*Y/R*, *C/D*, *A/L*, *C/D*}. We can see that since $o_k^1 \in O^1$, then the fault is of type A^1, i.e. the fault is located in m_1. Analyzing m_1 we get:



Applying the fault identification technique as illustrated above, the potential identified faults are:

Fault ID Class	Original transition	Faulty transition
Output Fault	s_1—(C/f^{12})→F^1	s_1—(C/D)→F^1
Tail State Fault	s_3—(A/L)→s_1	s_3—(A/L)→s_2

Case 2:
Assume the observed input/output sequence was as follows: {Y/R, C/D, Y/f^{12}}. Analyzing m_1 we get:

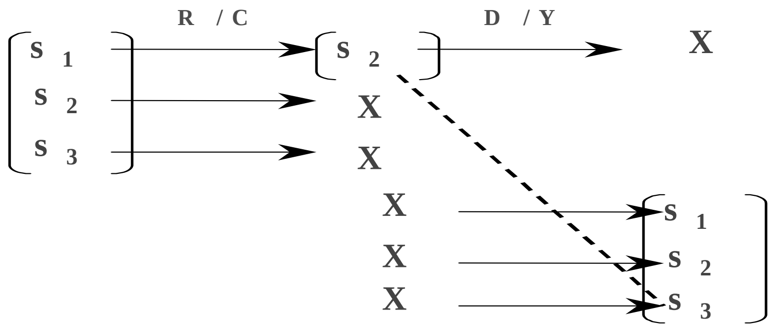


We can notice that since $o_3^1 = f^{12} \wedge \exists\, s_3 \in L^2$ has $\lambda(s_3, Y)=f^{12}$, then the fault is of type B_b^1, i.e. the fault is located either in m_1 or outside m_1. Analyzing m_1 as shown above we get the following potential faults:

Fault ID Class	Original transition	Faulty transition
Tail State Fault	s_2—(C/D)→s_3	s_2—(C/D)→s_2

Of course m_1 might have been implemented correctly reaching s_3 after reading the input/output C/D and then gotten the Y input and correctly produced the f^{12}. This is the case that the single fault is outside m_1, so we need to analyze m_2 and C_{12} and C_{21} to see how a single fault outside m_1 could have caused the observed input/output sequence.

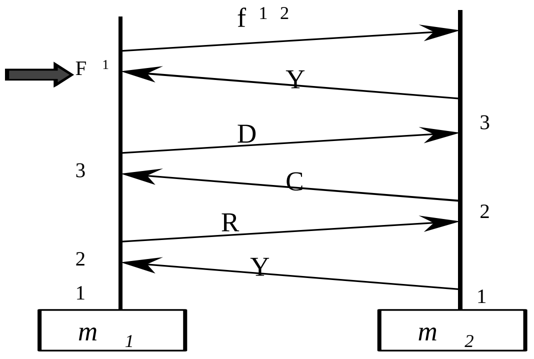
Now looking at m_2, we apply the procedure described above to get the expected input/output sequence at m_2. The sequence is $e^2=\{R/C, D/Y\}$. Analyzing m_2 we get:



Since $o_k^2 \in O^2$, then the fault is of type A^2, i.e. the fault is located in m_2. Applying the fault identification technique as before, we get:

Fault ID Class	Original transition	Faulty transition
Output Fault	s_2—(D/A)→s_3	s_2—(D/Y)→s_3

For the channels we use the MSC notation described above, we get the following figure:



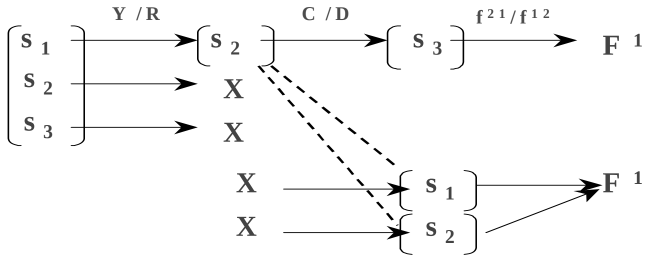
Going through the process of backward checking each exchanged symbol:

- For the symbol f^{12} since it is already a fault symbol and it is going out of m_1 where the observer is located, there is no potential alteration.
- For the symbol *Y*, this symbol can be an alteration of the symbol *A*, which was supposed to be produced by m_2 when it receives *D* while in state s_2^2.
- For the symbol *D*, it is the expected output of m_1 when moving from s_2^1 to s_3^1 under input *C*.
- For the symbol *C*, it can not be a result of an alteration since it is the valid output symbol from m_2 when it moves from s_1^2 to s_2^2 under input *R*.
- For symbol *R*, it is the expected output of m_1 when moving from s_1^1 to s_2^1 under input *Y*.
- For symbol *Y*, it is the only valid symbol for initiation.

So the only potential channel fault is in C_{21} where the symbol *A* could have been altered to *Y*.

<u>Case 3:</u>

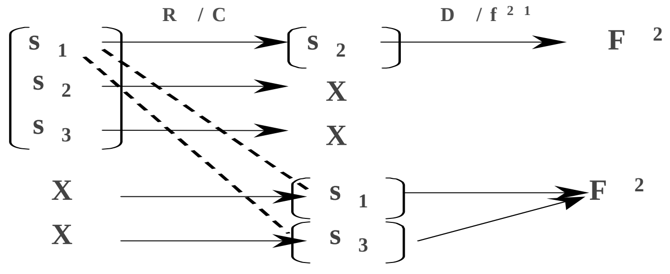
Assume the observed input/output sequence was as follows: {*Y/R, C/D,* f^{21}/f^{12}}. Analyzing m_1 we get:



Since $i_3^1=f^{21}$, then the fault is of type C_b, i.e. the fault can be either in m_1 or outside m_1. Analyzing m_1 as shown above, we get:

Fault ID Class	Original transition	Faulty transition
Tail State Fault	s_2—*(C/D)*→s_3	s_2—*(C/D)*→s_1
Tail State Fault	s_2—*(C/D)*→s_3	s_2—*(C/D)*→s_2

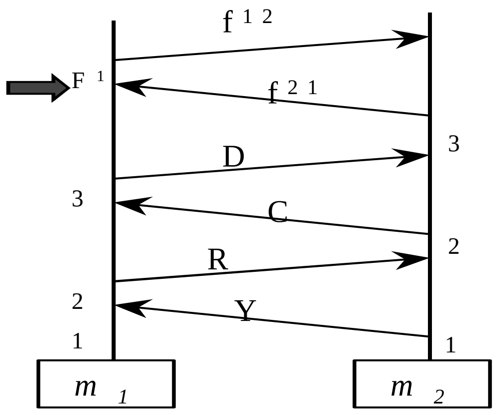
Now for m_2, we apply the procedure described above to get the expected input/output sequence at m_2. The sequence is e^2=*{R/C, D/*f^{21}*}*. Analyzing m_2 we get:



We can notice that since $o_3^2 = f^{21} \wedge \exists$ no $s \in L^2$ has $\lambda(s, D)=f^{21}$, then the fault is of type B_a^2, i.e. the fault is located in m_2. Analyzing m_2, we get:

Fault ID Class	Original transition	Faulty transition
Output Fault	$s_2 \xrightarrow{D/A} s_3$	$s_2 \xrightarrow{D/f^{21}} s_3$
Tail State Fault	$s_1 \xrightarrow{R/C} s_2$	$s_1 \xrightarrow{R/C} s_1$
Tail State Fault	$s_1 \xrightarrow{R/C} s_2$	$s_1 \xrightarrow{R/C} s_3$

For the channels, using the same MSC notation we obtain the figure below:



Analyzing the symbols in the backward direction, we get:

- For the symbol f^{12} since it is already a fault symbol and it is going out of m_1 where the observer is located, there is no potential alteration.
- For the symbol f^{21}, this symbol can be an alteration of the symbol A, that is supposed to be produced by m_2 when it receives D while is state s_3^2.
- For the symbol D, it can not be altered, since there is no other symbol X such that $\lambda(s_3^2, X)=Y$.
- For the symbol C, it can not be a result of alteration it is the only valid output symbol from m_2 when it receives R while in state s_2^2.
- For symbol R, as for D it cannot be altered.
- For symbol Y, it is the only valid symbol for initiation.

So the only potential channel fault is in C_{21} where the symbol A may be altered to Y.

To show effectiveness and efficiency of our fault identification approach using the CFSM model, we introduce two experiments simulating passive testing based fault identification on 2 practical protocols modeled in CFSM.

5 Experiments and Practical Examples

Experiment I

To investigate the effectiveness of the passive fault identification approach we have just discussed for the CFM model, we simulate the 2-node model shown in figure 5. The case here is a simplified version of the network layer of a connection oriented protocol, e.g. TCP/IP and X.25 protocols [10].

Placing the observer at machine m_1, we generate faults randomly and inject them in the system. Random generation of faults choose:

- Fault location: whether in m_1, m_2, C_{12} or C_{21},
- Fault time: when the fault is injected in the system, (i.e. at which step of the input/output sequence),
- Fault class: based of the fault characterization mentioned above,
- Fault identity: if the fault is located inside nodes, it tells which transition and whether it is an output or tail-state fault. If the fault is in channels it tells how the symbol is altered.

Time is measured in atomic steps, where one atomic step is equivalent to the time it takes for a transition to be executed in one FSM (i.e. a node). The simulator reports the fault detection time, location information, and the set of potential faults identified.

The simulator functionality can be summarized as follows:

- First, the simulator generates the fault randomly as explained above, randomly selects a valid input/output sequence, and injects the fault into the system.
- The forward trace analysis is then done assuming that the observer is at m_1 and computes the set of possible states $\{L^i\}$ until the fault is detected $\{L^i = \phi\}$.
- Using the fault characterization, we can get fault location information.
- Based on fault location, more analysis in fault locations is done to give the set of potential faults that can be identified in each of these locations. For the case of node faults we complete the fault identification process by the backward trace and recurrent fault procedure. For the case of channels we analyze the input/output sequence using the MSC approach described above.
- The simulator computes the following results: fault detection time since injection, number of back steps for backward trace(s), count of identified faults. Aggregate analysis -such as histograms and averages of these parameters- are computed for the whole set of tests.

<u>Results and comments:</u>
Running the experiment for 50,000 random faults injected into the system and the fault identification process simulated, we get the following results:

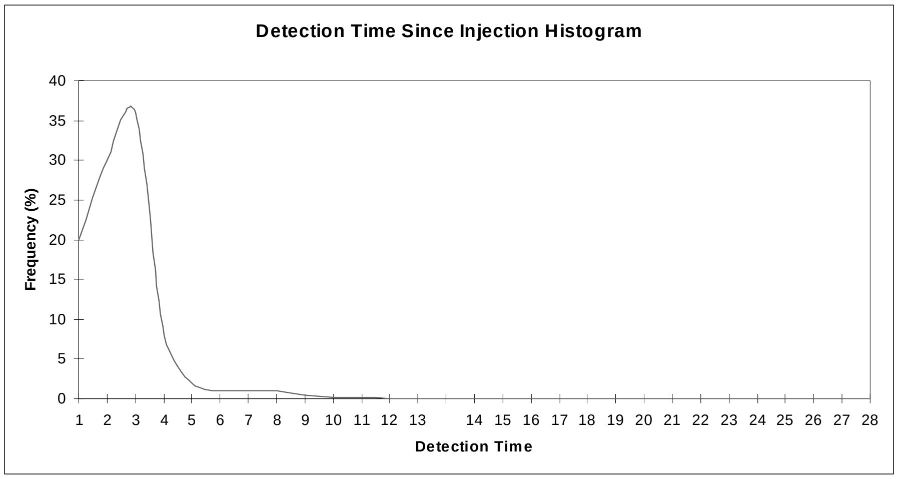


Figure 6 Histogram for Detection Time since Injection

- It can be seen that most of the detection times are between 1 and 3.
- It can be observed that the passive testing based fault identification does not take long to detect the fault once injected.
- Since we are measuring here the detection time since fault injection, the detection time since injection is almost independent of the length of the observed input/output sequence.

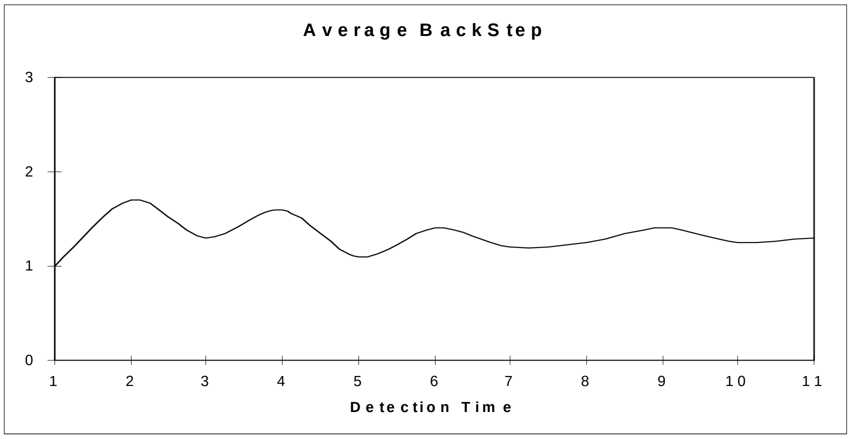


Figure 7 Average Backward Trace Lengths vs. Detection Time

- For statistical significance purposes, we truncated these results at detection time of 11 according to the histogram shown above.
- Average number of back steps is around 1, which demonstrates that this approach is efficient.

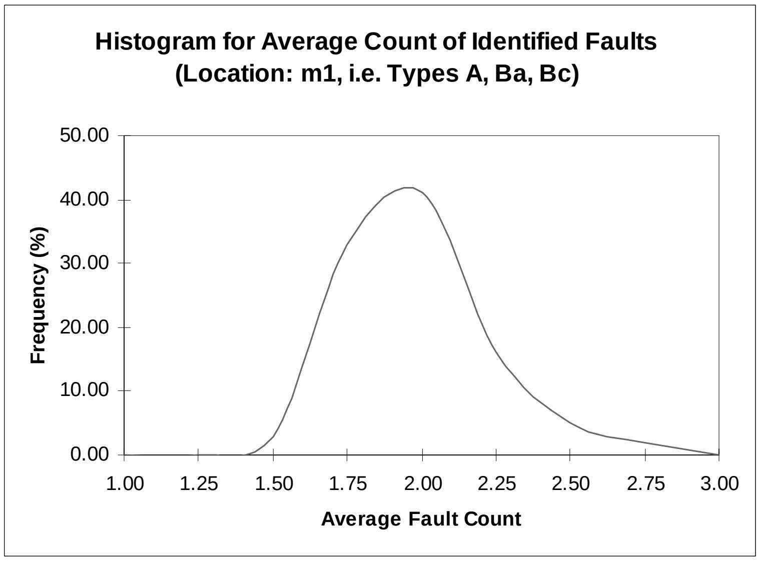


Figure 8 Histogram of average count of faults located in *m_1*

- This diagram shows that when the fault is located inside m_1, the number of identified potential faults is around 2 faults. The smaller the set of identified potential set of faults, the easier it is for any later fault correction process.

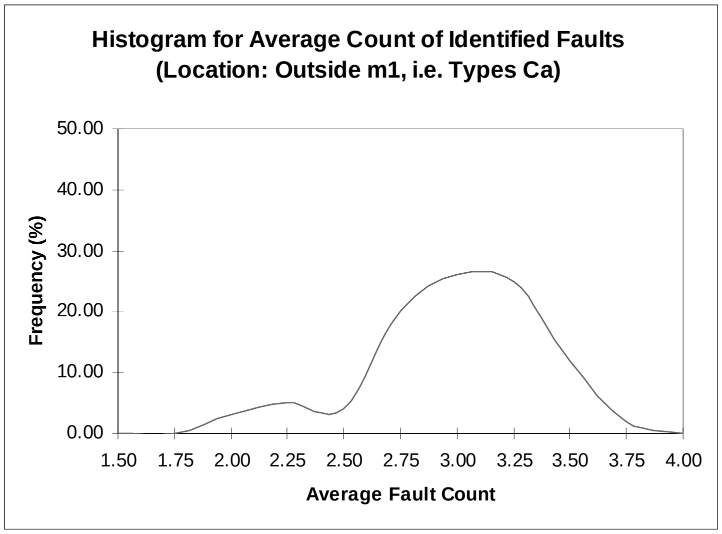


Figure 9 Histogram of average count of faults located outside *m_1*

- This diagram shows that when the fault is located outside m_1 (i.e. m_2, c_{12} or c_{21}), the number of identified potential faults is around 3 faults.

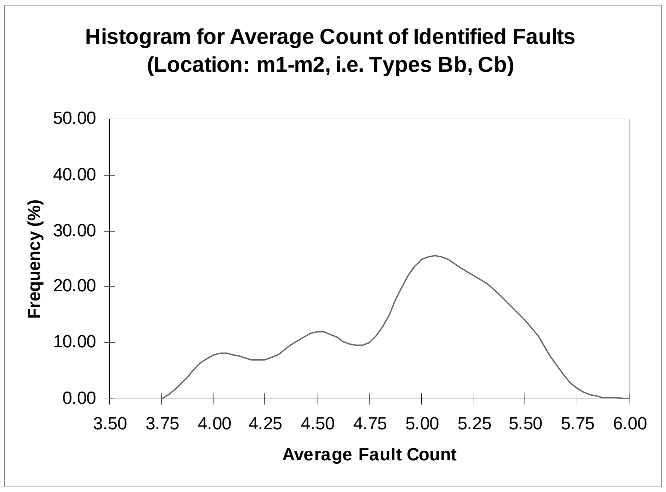


Figure 10 Histogram of average count of faults not located

- This diagram shows that when the fault can not be specifically located (i.e. can be in m_1, m_2, c_{12} or c_{21}), the number of identified potential faults is around 5 faults.

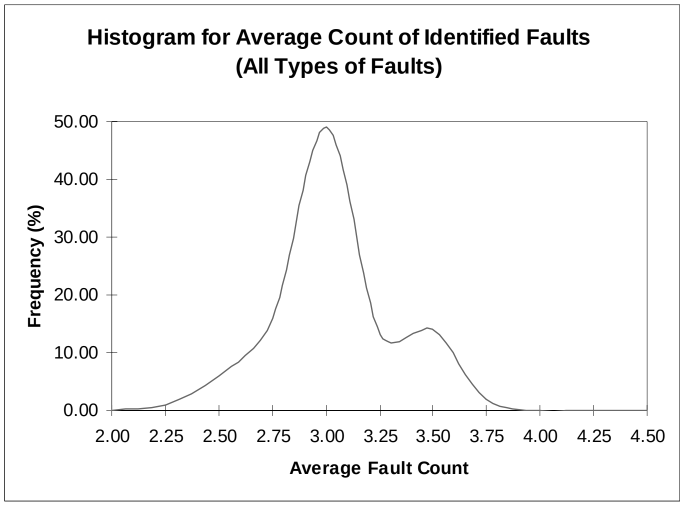


Figure 11 Histogram of average count of all faults

- Finally, this diagram averages the entire above fault location based histograms, assuming equally probable fault types. It can be shown that the average number of identified potential faults is around 3 faults. It shows how efficient and effective the fault identification process is, using our passive testing approach on the CFSM model.
- In order to see how much passive testing based fault identification shrunk the set of faults, for this CFSM model the total number of possible faults is 24. So, our approach reduces the fault space to 1/8th of its original size, i.e. 87.5% reduction in size of the set of faults.

Experiment II

Now we extend our experimentation using a more complex communication protocol and simulating the 3-node model shown in figure 10 below. The experiment is an abstraction of an X.25 network layer protocol, as explained in [4][5], applied to an avionics VHF communication system, where:

- Node *B* represents the aircraft DTE,
- Node *A* represents the ground station,
- Node *C* represents the ground DTE (Air Traffic Controller or Airline office),
- End user *X* represents the pilot for this aircraft application,
- And end user *Y* represents end user at the ground system.

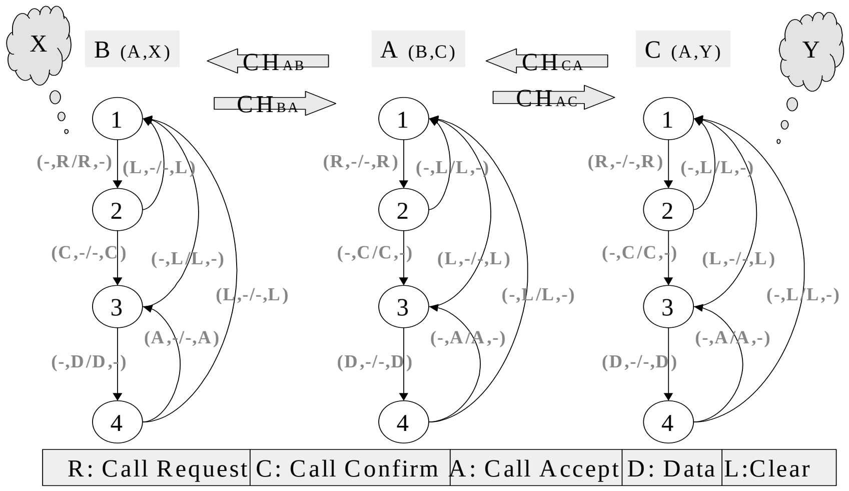


Figure 12 the 3-node CFSM Model

Using the simulator described earlier, we generalize the fault injection procedure to select more fault locations (m_A, m_B, m_C, c_{AB}, c_{BA}, c_{AC}, or c_{CA}) and apply the fault location techniques as in [4][5] for the 3-node model. Then, repeating the same steps as in experiment I, we get the following results.

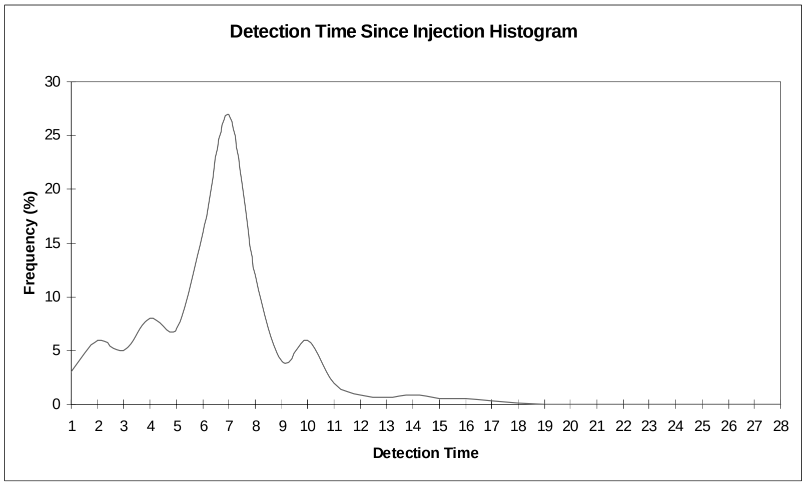


Figure 13 Histogram for Detection Time since Injection

- It can be seen that most of the detection times are between 6 and 8. This confirms our previous conclusion that the passive testing based fault identification does not take long to detect the fault once injected.

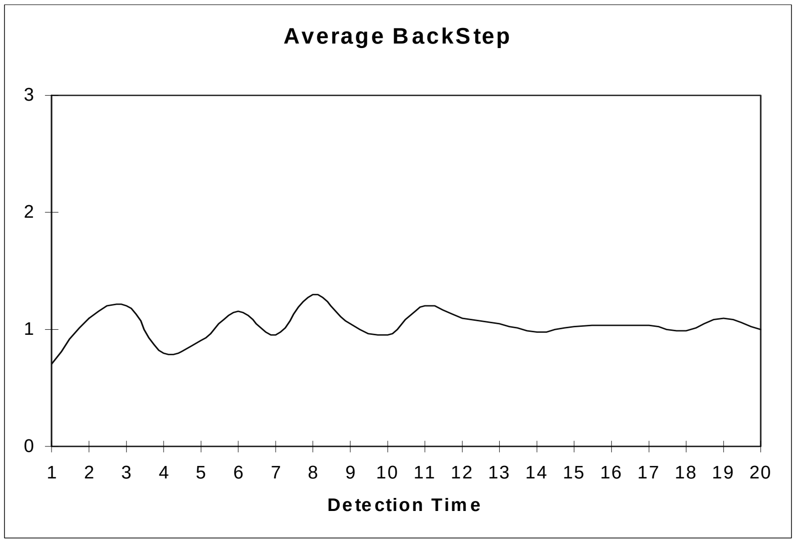


Figure 14 Average Backward Trace Lengths vs. Detection Time

- Average number of back steps is around 1, which demonstrates that this approach is efficient.

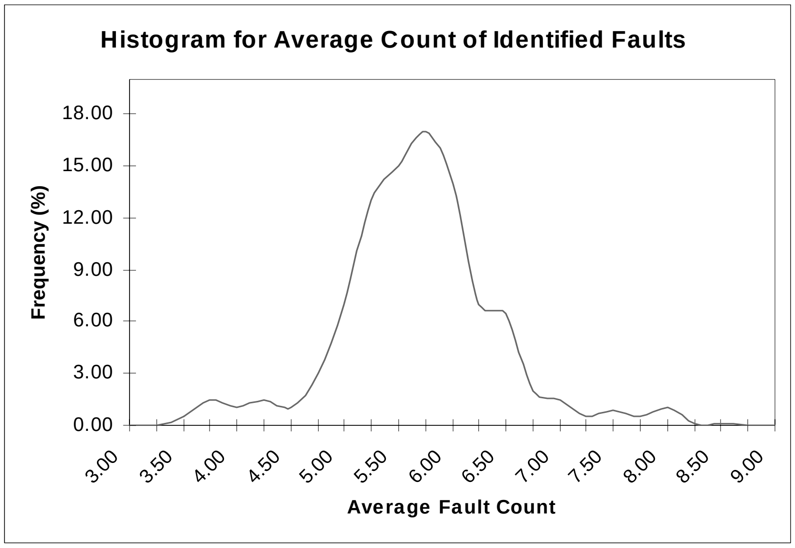


Figure 15 Histogram of average count of identified faults

- Finally, this diagram shows the average count of identified faults, assuming equally probable fault types. It can be shown that the average number of identified faults is around 6 faults. It shows how efficient and effective the fault identification process is using our passive testing approach on the CFSM model.
- In this experiment, this CFSM model has a total number of possible output faults of $7\times14\times3 = 294$, and a total number of possible tail state faults of $7\times3 = 21$, i.e. the total number of possible faults is 315. So, our approach reduces the fault space to less than 1/52nd of its original size, i.e. 98.1% reduction in size of the set of faults.
- Notice that the average fault detection time varies depending on the structure of the finite state machine model as well as the nature of the protocol. Examples of factors of the machine structure are the length of the longest loop and the degree of transition fan-out or branching from each state. Examples for protocol factors are the frequency of visiting each transition and the probability of triggering events. However, in general the fault detection time is reasonably short.

6 Conclusions and Possible Extensions

In this paper we have shown how passive testing can be used to reduce the number of faults that could have caused a network implementation to display faulty behavior. The network specification was assumed to be a CFSM and the faults considered were output fault and tail-state fault deviations from the specification. Thus, once a fault is detected this fault identification approach can be used to narrow the possibilities of what fault occurred, thus simplifying the following tests aimed at uniquely identifying and correcting the fault. The approach was demonstrated through several examples. Practical protocol examples were used to demonstrate the effectiveness of the approach. Extensive simulation was done for both the 2-node and the 3-node examples over many simulation input/output sequences and many random injections of faults. This simulation demonstrated that the set of possible faults could be determined after only a very modest number of steps once a fault was injected, and also showed that considerable reduction in the number of possible faults giving rise to the observed input/output sequence was obtained by our approach. Detection time results were promising, showing that passive testing in CFSM model will detect and identify faults in reasonable time. The few backward steps indicate a small overhead for fault identification using the Forward-Backward-Crossover part of our analysis.

Much remains to be done. First, the fault coverage attained by the observed input/output sequence can also be determined. We have shown that many of the approaches we have already discussed can be used to determine the fault coverage. These results are being included in a forthcoming technical report, and hopefully a future paper. If so, passive testing could be used first for fault detection, followed by fault location and then followed by fault identification in the small region of the network containing the fault. Finally, fault coverage results would provide some assurances as to how “good” the test was.

There are other major challenges, however. One issue is whether the passive testing approach can be extended to include the time dimension. Such an extension could enable the model not only to perform fault management, but also to give some insight about performance management, such as response time. Although this extension may look orthogonal to our work of fault management using passive testing, it would allow one to obtain even more information by passive observation.

Another issue is whether further passive testing, beyond when a fault is detected, could be used to provide a better fault identification and coverage. After all, the implementation would keep running even though it is known to be faulty, and thus some method of continuing the analysis might be possible and useful for gaining more information. Although we have some initial thoughts on how this might proceed we do not have any definite results to report.

The final challenge is to see how the technique that has been developed for passive testing might be applied in the fault management systems of real network management tools. This somewhat formal approach and way of thinking seems to be quite distant from the techniques currently used in actual network management systems.

References

[1] D. Lee, A. Netravali, K. Sabnani, B. Sugla, and A. John, “Passive Testing and Applications to Network Management,” Proceedings of IEEE International Conference on Network Protocols, pp. 113-122, October 1997.

[2] R. E. Miller, “Passive Testing of Networks using a CFSM Specification,” 1998 IEEE International Performance Computing and Communications Conference, pp. 111-116, February 1998.

[3] R. E. Miller, “Passive Testing of Networks Using a CFSM Specification,” Bell Labs Technical Memorandum, BL011345-97-0522-03TM.

[4] R. E. Miller and K. Arisha, “On Fault Location in Networks by Passive Testing,” 2000 IEEE International Performance Computing and Communications Conference, February 2000.

[5] R. E. Miller and K. Arisha, “On Fault Location in Networks by Passive Testing,” Computer Science Dept., University of Maryland College Park, Technical Report #4044, August 1999.

[6] W. Stallings, “SNMP, SNMPv2, and CMIP The Practical Guide to Network-Management Standards,” Addison-Wesley Publishing Company, 1993.

[7] ISO/IEC 7498-1: 1994 | ITU-T Recommendation X.200 (1994) Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model, 1994.

[8] R. E. Miller and K. Arisha, “Fault Identification in Networks by Passive Testing,” 2001 IEEE Advanced Simulation Technologies Conferences, April 2001.

[9] D. Brand and P. Zafiropulo, “On Communicating Finite-State Machines,” JACM, Vol. 30, No. 2, pp. 323-42, April 1983.

[10] ISO/IEC 8208: Information Technology – Data Communications – X.25 Packet Layer Protocol for Data Terminal Equipment, Third Edition, August 1995.