On Fault Management using Passive Testing for Mobile IPv6 Networks

Raymond E. Miller
Department of Computer Science
University of Maryland,
College Park, MD 20742
miller@cs.umd.edu

Khaled A. Arisha
Department of Computer Science
University of Maryland,
College Park, MD 20742
arisha@cs.umd.edu

Abstract

In this paper, we employ the Communicating finite state machine (CFSM) model for networks to investigate fault management 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. We present our new passive testing approach for fault location, fault identification, and fault coverage based on the CFSM model. Examples are given for each fault management function to illustrate our approach. Then, we illustrate the effectiveness of our new technique through simulation of a practical protocol example, a 4-node mobile Ipv6 network. Finally future extensions and potential trends are discussed.

I. 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 [14]. 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) [13]. 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 [8][8]. We illustrate our techniques by applying them to a 4-node mobile IPv6 network.

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 desirable 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 [8][8] 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. Miller and

Arisha [2][3] demonstrated that better fault location is possible. 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. This greater fault location and identification capability by passive testing is demonstrated here for the IPv6 protocol.

We describe the CFSM based specification model, the observer model, and the fault model in section 2 of this paper with necessary assumptions. In section 3, we introduce a brief description of the fault detection algorithm using passive testing. Section 4 presents the results of fault location. In section 5, we describe our work of fault identification for the single FSM model and the CFSM model as well. Section 6 provides an overview of the fault coverage work with an example. Section 7 presents experiments modeling and simulation of the Ipv6 protocol for a 4-node mobile network, as well as simulation results to demonstrate the effectiveness of our approach. Finally, conclusions and possible extensions are discussed in section 8.

II. 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.

A. 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: We assume that if a fault occurs, only one fault occurs during a test cycle. We also assume that our FSMs are deterministic. For all unspecified input transitions, 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. For more detail about justification of these assumptions, refer to [1][2][4][8].

B. The CFSM Model

Our model 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 [15]. 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 $C=\{C_{ij}: i,j \leq r \wedge i \neq j\}$ 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 [2][3].

C. The Observer

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: We assume that the network topology of the implementation is the same as the specification. 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. The node is viewed as a black box FSM for the observer. For more detail about justification of these assumptions, refer to [2][3][6].

D. The Fault Model

Due to our assumptions of the CFSM model used in passive testing, the three types of faults that we can investigate, in terms of the CFSM specification, are:

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.

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.

Channel Fault: This occurs when a channel corrupts a message (i.e. an input and/or output symbol)

Assumptions:

- Only a single fault exists on the network.
- Faults in the nodes are persistent, while faults in the channels are non-persistent.

III. FAULT DETECTION

Passive testing fault detection for a network using the FSM model was first developed in [1]. the fault detection capability of passive testing can be summarized as follows:
As an input/output sequence of the implementation machine B is observed it is compared with the expected behavior of the specification FSM 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 observed input/output sequence. The procedure for detecting this is to first start out with the set L^0 consisting of all states of A, since we do not know what state A is supposedly in at the start of the observed input/output sequence. Then with the first observed input/output i_1/o_1 we compute a new set of states L^1, the successor states of A from states in L^0. This process is continued for each i_j/o_j to produce an L^j set from L^{j-1}. If at some point L^j becomes a singleton set then the sequence up to this point is called a *passive homing sequence*. If at some point k L^k becomes empty, we know that B is faulty since no state A could produce this observed input/output sequence.

The detailed algorithm that describes the above procedure is in [1]. 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.

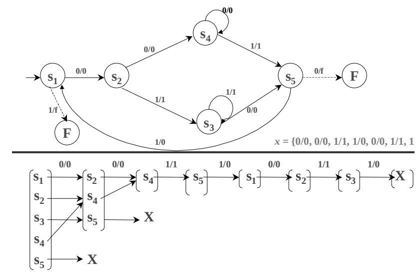

Fig. 1. FSM example for the specification

IV. FAULT LOCATION

Referring to the fault location work on the two-node model done by Miller [8][8], the detected fault can be characterized with respect to its location in the network as follows:
Type A fault: $o_k^1 \, \varepsilon \, O^1$ of m_1 → the fault is in m_1.
Type B_a fault: $o_k^1 = f^{12} \wedge$ no $s^1 \in L^{k-1}$ has $\lambda(s^1, i_k^1)=f^{12}$ → the fault is in m_1.
Type B_b fault: $o_k^1 = f^{12} \wedge \exists\, s^1 \in L^{k-1}$ has $\lambda(s^1, i_k^1)=f^{12}$ → the fault is in m_1 or outside m_1.
Type B_c fault: $o_k^1 \notin O^1 \wedge o_k^1 \neq f^{12}$ → the fault is in m_1.
Type C_a fault: $i_k^1 \notin I \wedge i_k^1 \neq f^{21}$ → the fault is outside m_1.
Type C_b fault: $i_k^1 \notin I^1 \wedge i_k^1 = f^{21}$ → the fault is in m_1 or outside m_1.

Fault type B_c is first appended to the fault characterization in [2][3]. More elaboration to generalize the fault location work is given in [2][3]. From this work, analysis done at the observer can be viewed as a node cut through a large network splitting the network into three parts: The cut and the two sides of the cut. To get finer location we can consider multiple node-cuts such that these cuts, together, create relatively small regions for the network. Using our fault location capabilities through each cut, we will be able to locate a fault to a smaller region as follows.

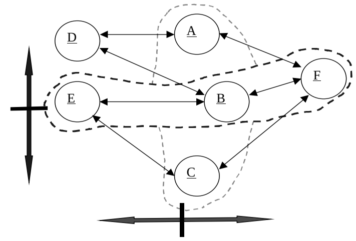

Fig. 2. A multiple-node-cut example

From this point of view, we show how we can obtain more information regarding fault location. Taking into consideration our assumption that there is a central observer to which other local observers can report. For the above figure, the node cut passing through ABC can have 3 observers, one at each node over this node cut. By combining the fault location that is reported from each observer, we can determine whether the fault is located to the left or right side of that node cut. If we look at the other node cut passing through EBF which can also have 3 observers, one at each node of this node cut, we determine whether the fault is above or below that node cut. If we combine the location information from both these edges, we can isolate a region of the network where the fault resides. This leads to more precision in the fault location approach. Subsequent active testing can be applied to the isolated region to determine what fault occurred in that region of the network.

V. FAULT IDENTIFICATION

This section covers the fault identification approach for the single FSM model as described in [4][5][6]. 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.

A. Fault Identification for the single FSM model

In section 3 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.

- We let $(L^k)^R$ be the set of all states of A.
- 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 $(L^j)^R$.

1) 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 —(i_{j+1}/o_{j+1})→$\delta(s_p, i_{j+1})$ could have occurred.

Proof: is given in [4][5].

2) Tail State Fault Identification:

Theorem 2: If L^j has a state s_p with transition s_p —(i_{j+1}/o_{j+1})→ s_q and there is a s_r in $(L^{j+1})^R$, then s_p —(i_{j+1}/o_{j+1})→s_r is a tail state fault that could have occurred.

Proof: is given in [4][5].

Example

Using the same FSM specification as shown in figure 1, we assume an observed input/output sequence: {*0/0, 0/0, 0/0, 0/0, 1/1, 00*}. The forward and backward traces are shown in figure 3 along with "crossovers" shown by dotted arrows.

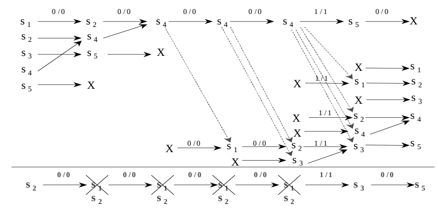

Fig. 3. Example of the Fault Identification

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

Applying theorem 2 we see that this crossover depicts an output fault of transition s_5 —*(0/f)*→ *F* changing to s_5 —*(0/0)*→*F*.

Applying theorem 3 we see that this crossover depicts tail-state faults of the following transitions:

- s_4 —*(1/1)*→s_5 changing to s_4 —*(1/1)*→s_1.
- s_4 —*(1/1)*→s_5 changing to s_4 —*(1/1)*→s_2.
- s_4 —*(1/1)*→s_5 changing to s_4 —*(1/1)*→s_3.
- s_4 —*(1/1)*→s_5 changing to s_4 —*(1/1)*→s_4.
- s_4 —*(0/0)*→s_4 changing to s_4 —*(0/0)*→s_2.
- s_4 —*(0/0)*→s_4 changing to s_4 —*(0/0)*→s_3
- s_4 —*(0/0)*→s_4 changing to s_4 —*(0/0)*→s_1

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

- Do the forward trace analysis for the observed input/output sequence, letting k be the least k such that $L^k=\phi$.
- 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.
- 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 [4][5].

Notice that in figure 3 as described below the horizontal line, the Tail-state fault {s_2 —*(0/0)*→ s_4 changing to s_2 —*(0/0)*→ s_2} is undetected by the Forward-Backward-Crossover technique. This type of fault is called a "recurrent" fault. Extensions to our fault identification technique to detect such recurrent faults are described in [4].

B. 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.

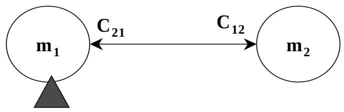

Figure 4: A two-node model

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}.

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 to be 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 $e^2=o_1/i_2, o_2/i_3, \dots, o_{k-2}/i_{k-1}, o_{k-1}/i_k$.

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

This can be expressed as: If $e^1=\{\ i_j^1\ /\ o_j^1 \mid j=1,\dots,k\}$ then $e^2=\{\ i_j^2\ /\ o_j^2 \mid j=1,\dots,k\text{-}1 \text{ and } i_j^2=o_j^1 \text{ and } o_j^2=i_{j+1}^1\ \}$. 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, leads 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.

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.

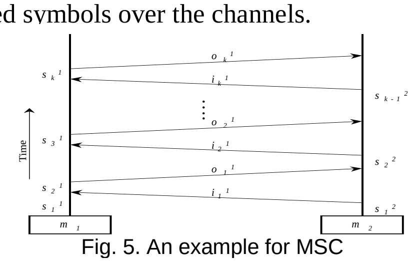

Fig. 5. An example for MSC

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 $e^{1*}=\{\ i_t^1\ /\ o_t^1 \mid t=j,\dots,k\}$ 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 $e^{1**}=\{\ i_t^1\ /\ o_t^1 \mid t=j+1,\dots,k\}$ 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.

VI. FAULT COVERAGE

This section gives an overview of the fault coverage work. During the fault identification description, it is illustrated how we collect information needed for the fault coverage computation. So, it covers the fault coverage process for a single node. It also gives an illustrative example to apply the proposed technique. Finally, we introduce how to generalize our fault coverage functionality to the CFSM model.

A. *Fault Coverage for the single FSM model*

As a product of the fault identification, we can get two sets of faults: Set-I and Set-II. If any of the faults of Set-I occurs, they could produce the observed input/output sequence of the implementation. The faults of Set-II are known not to have occurred since they would have produced a different output sequence than observed, for the observed input sequence. Extending the Forward/Backward Crossover Algorithm such that, during this trace, after passive homing occurs, we can identify potential set-II faults, which are known not to have occurred. These potential set-II faults can be computed as all possible output and tail-state faults that can be produced from traversed transitions during the input/output observations, since these faults if they had occurred would have produced a different output sequence.

For the transitions before passive homing, a transition is considered traversed only after we make sure that it has been executed over all successful paths of transitions from L^0 to the homing state. So, we have to wait until the passive homing occurs to check out the pre-homing transitions to decide whether to add them or not to the potential set-II faults.

At the end of the backward trace, we need to remove from the potential set-II faults, any set-I fault that are detected during the backward phase. Set-I faults are identified during the backward trace after the collection of the potential set-II faults in the forward trace. During the forward trace we cannot predict these set-I faults. After the backward trace, we remove any set-I faults that were collected in the potential set-II faults.

Fault coverage is defined as the percentage of faults found by the test sequence observation versus the total number of faults. First we describe how to calculate the total number of faults. Then, we show how to compute the fault coverage achieved by our observed input/output sequence.

The method used to determine the total number of faults in each class is as follows:

For the total number of output faults, we consider every transition in the specification FSM. To get an output fault we have to alter its output to every possible output symbol other than the output symbol in the transition. Repeating this for every transition leads to:

Total number of Output Faults = |transitions| $\times$ (q - 1)

For the total number of tail state faults, we consider again every transition and alter its tail state to every possible state other than its original tail state. This leads to:

Total number of tail state faults = |transitions| $\times$ (n - 1)

Fault Coverage can be calculated as the percentage of the total number of identified faults (Set-I + Set-II) versus the total number of faults.

Example

The example below shows the application of the algorithm to the FSM network model shown in Figure 1.

Assume the observed input/output sequence is: *0/0, 0/0, 1/1, 1/0, 0/0, 1/1,* and *1/0*. Tracing the set of possible states according to the observed i/o sequence, we get the forward trace. Then starting from the last observed i/o tuple, we can produce another backward trace of possible states. The forward and backward traces are shown in Figure 6.

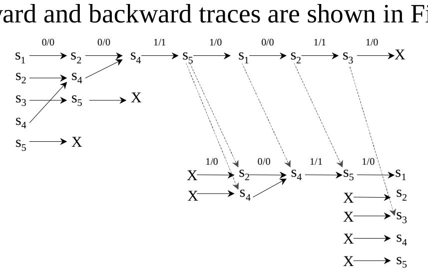

Fig. 6. Example of Forward and backward traces

First, we use the forward phase to calculate potential set-II faults. Every transition visited in the forward trace implies that all associated faults (Output and Tail-State) have been considered, as shown in the table below. To calculate these associated potential set-II faults:

Output faults for a given transition are all possible alterations of the output symbol. Thus, the number of output faults is (q – 1).

Tail-State faults of a given transition are every possible alteration of the tail (destination) state value. Thus, the number of tail-state faults is (n – 1).

Output fault coverage = (set-I + set-II)/total
= (1+4)/8 = 5/8 = 62.5%,

Tail State fault coverage = (set-I + set-II)/total
= (4+12)/32 = 16/32 = 50%,

For more detail about fault coverage calculation, refer to [7][8]. Notice, that it is desired to have set-I fault coverage as small as possible, since the smaller the number of identified set-I faults, the smaller our uncertainty in possible faults to be used in a later fault correction phase. While for set-II and total fault coverage, it is better to have them as large as possible, since this indicates a minimum number of the faults that are left undetected.

B. Fault Coverage for the CFSM model

For fault coverage to be generalized for CFSM model based passive testing, we need to consider the sources of faults. These sources are the nodes and the channels. For the nodes, since each is modeled as a FSM, we can apply the fault coverage procedure as in 4.2 for each node individually to get its own fault coverage. For channels, fault coverage cannot be realistic since according to our assumptions the faults in the channels are transient. So, a symbol transmission through one channel cannot exclude the possibility of occurrence of a related fault for the same symbol transmission later due to the transient nature of faults. Because of this the overall fault coverage is computed as the average of the fault coverage of each individual node.

VII. EXPERIMENTS AND PRACTICAL EXAMPLES

To investigate the effectiveness of the passive testing based fault management approach we have just discussed for our CFM model; we model IPv6 mobility support with a 4-node CFSM model shown in figure 7, and simulate the passive testing techniques we have just described. First we give a brief introduction for the mobility support of the Internet Protocol version 6 (Ipv6), then we discuss the CFSM model, the simulation and the results.

With recent advances in wireless communication technology, mobile computing is an increasingly important area of research. A mobile system is one where independently executing components may migrate through some space during the course of the computation, and where the pattern of connectivity among the components changes as they move in and out of proximity [10]. IETF made efforts to standardize the introduction of the mobility to the Internet Protocol version 6 (IPv6) [11][12]. IP version 4 assumes that a node's IP address uniquely identifies the node's point of attachment to the Internet. Therefore, a node must be located in the network indicated by its IP address to receive datagrams (connectionless packet data units) directed to it; otherwise, datagrams destined to the node would be undeliverable. The alternative mechanisms, proposed by IP version 4, for a node to change its point of attachment without losing its ability to communicate are unacceptable due to the difficulties in maintaining higher-layer connections and server scaling problems. The IPv6 protocol is a new scalable mechanism to accommodate node mobility within the Internet. Mobile IP introduces the following new functional entities:

1) Mobile Node: a host or router that changes its point of attachment from one network or subnetwork to another. A mobile node may change its location without changing its IP address.

2) Home Agent: a router on a mobile node's home network which tunnels datagrams for delivery to the mobile node when it is away from home, and maintains current location information for the mobile node.

3) Foreign Agent: A router on a mobile node's visited network that provides routing services to the mobile node when registered. The foreign agent de-tunnels and delivers datagrams to the mobile node that were tunneled by the mobile node's home agent. For datagrams sent by a mobile node, the foreign agent may serve as a default router for registered mobile nodes.

The mobile node is given a permanent IP address on a home network. When away from its home network, a "care-of-address" is associated with the mobile node and reflects the mobile node's current point of attachment. The mobile node uses its home address as source address of all IP datagrams that it sends. The following steps provide an outline of the operation of the mobile IP protocol:

- Mobility agents (i.e. foreign agents and home agents) advertise their presence via agent advertisement messages. In the absence of agent advertisements, a mobile node may optionally solicit an agent advertisement message from any locally attached mobility agents through an agent solicitation message. All mobility agents should respond to agent solicitation.
- A mobile node receives these agent advertisements and determines whether it is on its home network or a foreign network.
- When the mobile node detects that it is located on its home network, it operates without mobility services. If returning to its home network from being registered elsewhere, the mobile node de-registers with its home agent, through exchange of a Registration Request and Registration Reply with it.
- When a mobile node detects that it has moved to a foreign network, it obtains a care-of-address on the foreign network. The care-of-address can either be determined from a foreign agent's advertisement (a foreign agent care-of-address), or by some external assignment mechanism such as the Dynamic Host Configuration Protocol (DHCP) (a co-located care-of-address).
- The mobile node operating away from home then registers its new care-of-address with its home agent through exchange of a Registration Request and Registration Reply message with it, possibly via a foreign agent.
- Its home agent intercepts Datagrams sent to the mobile node's home address, tunneled by the home agent to the mobile node's care-of-address, received at the tunnel endpoint (either at a foreign agent or at the mobile node itself), and finally delivered to the mobile node.
- In the reverse direction, datagrams sent by the mobile node are generally delivered to their destination using standard IP routing mechanisms, not necessarily passing through the home agent.

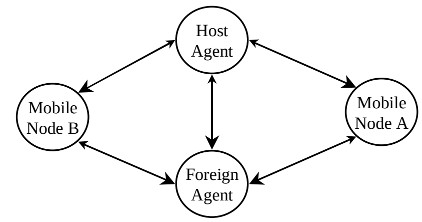

Fig. 7. Architecture of the model of IPv6 mobile network

As shown in figure 7, our model has two mobile nodes (*A* and *B*) moving randomly between the two subnetworks. Each subnetwork has a mobility agent. We call the mobility agent of mobile node *A*'s home network the "home agent", while the mobility agent of mobile node *B*'s home network the "foreign agent". The links connecting the mobile nodes to the agents are wireless, while the link between the mobility agents may not be wireless. Using our CFSM model, we place an observer at each of the mobile agents. Figure 8 illustrates the FSM representing the mobile agent, while figure 9 illustrates the FSM representing the mobile node.

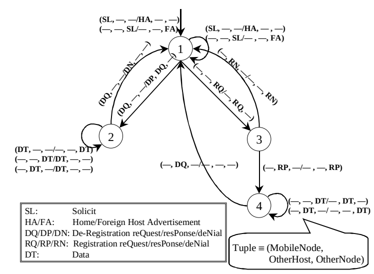

Fig. 8: The FSM representing the mobile agent

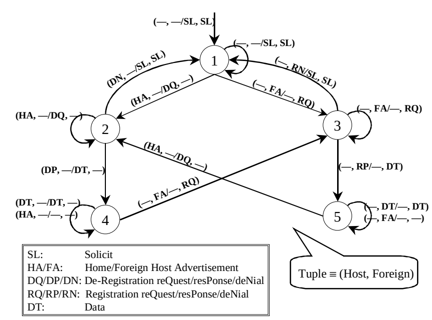

Fig. 9. The FSM representing the mobile node

Placing the observer at the mobile agents (home and foreign agents) we generate faults randomly and inject them in the system. Random generation of faults choose:

- Fault location: whether in home agent, mobile agent, mobile node *A*, and mobile node *B*.
- 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, the fault location information, the set of potential faults identified, and the fault coverage.

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 observers are at the mobile agents and computes the set of possible states $\{L^i\}$ for each observer until the fault is detected $\{L^i = \phi\}$ by at least one of the two observers.

Using the fault characterization, we can get fault location information.

Based on fault location, more analysis in fault location 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. Fault coverage information is computed during the forward and backward traces.

The simulator computes the following results: fault detection time since injection, number of located faulty entities, average count of identified faults. Aggregate analysis, such as histograms and averages of these parameters; are computed for the whole set of tests.

Running the experiment for 50,000 random faults injected into the system and the fault management process simulated, the final results for fault detection, location, identification and coverage are illustrated as follows.

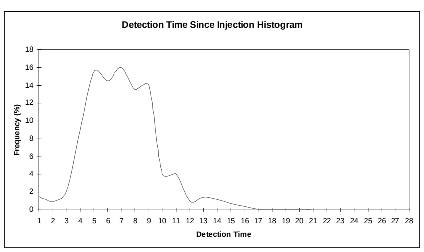

Fig. 10. Histogram for Fault Detection Time

It can be seen that most of the detection times are between 4 and 9. The passive testing based fault management 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.

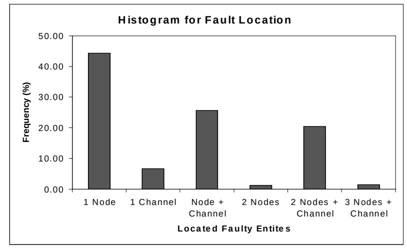

Fig. 11. Histogram for Fault Location Results

It can be seen that almost half the time, the fault is located in just one entity (one node or one channel). About 25% of the time the fault is located in one node and one of its channels, and nearly 20% of the time the fault is located in 2 nodes with their connecting channel. With this observation, we can realize that the fault location can enhance the active corrective process in this 4-node network example. It reduces the uncertainty about fault location from the whole network to only a few entities.

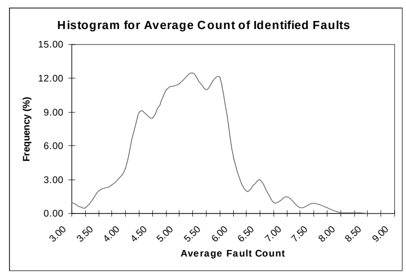

Fig. 12. Histogram for Average Number of identified Faults

This diagram shows that the average number of identified potential faults mostly lies between 5 and 6 faults. The smaller the set of identified potential set of faults, the simpler it is for a later fault correction process. Thus, it shows how efficient and effective the fault identification process is, using our passive testing approach on the CFSM model. In order to evaluate how much passive testing based fault identification shrunk the set of potential faults, for this CFSM model the total number of possible faults is about 420. So, our approach reduces the fault space to $1/70^{th}$ of its original size, i.e. 98.6% reduction in the possible number of faults that would have to be inspected in a later fault correction process.

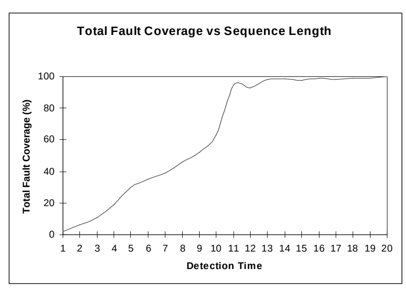

Fig. 13. Fault Coverage vs. Sequence Length

These graphs demonstrate that the fault coverage in general asymptotically approaches 100% coverage as the sequence length increases. Also, for rather short sequence lengths [4, 10], the fault coverage is around [20%, 65%], which is very efficient with such short sequence lengths

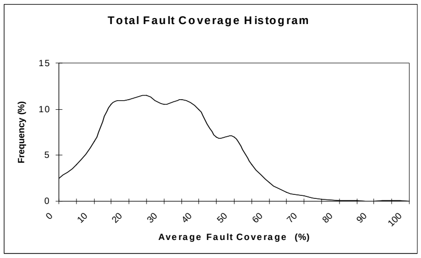

Figure 14: Histogram for Fault Coverage

It can be noticed from the above histogram is that most of the average total fault coverage values lie between 15% and 65%. So, even though we don't have control on the observed input/output sequence, we still can have reasonably good fault coverage.

VIII. CONCLUSIONS AND POSSIBLE EXTENSIONS

A. *Conclusions*

In this paper we have shown how passive testing on a practical example (IPv6 mobility support) can be used in fault management for networks. Fault management includes fault detection, fault location, fault identification, and fault coverage. Previous work by Lee introduced fault detection based on passive testing for a single FSM model. Later work by Miller described how to extend the model to a CFSM model and how to add fault location capabilities based on passive testing. Our contribution is to introduce for both the FSM and the CFSM models an integrated fault management solution based on passive observations. Passive testing could be used first for fault detection, followed by fault location to determine a smaller region of the network containing the fault. Then using fault identification to reduce the number of faults that could have caused a network implementation to display faulty behavior. Finally, fault coverage results would provide some assurances as to how "good" the test was. In this section we summarize our conclusions and remarks on our contributions, first for the single FSM model and then for the CFSM model.

A mobile IPv6 network model was used to demonstrate the effectiveness of the approach on a practical example. Extensive simulation was done for this example over many simulation input/output sequences and many random injections of faults. This simulation demonstrated that:

For fault detection capability the results in section 7 demonstrate that the average time to detect a fault in our experiment is quite low (between 4 and 9 steps). That is, it does not take long for passive testing to detect a fault

For fault location information, the results show that our approach –in most of the cases- reduces the suspected faulty region. Thus, one obtains a reduction in the amount of work required for the active corrective phase.

The set of identified possible faults can be determined after only a very modest number of steps once a fault was injected, and also that considerable reduction in the number of possible faults giving rise to the observed input/output sequence is obtained by our approach. Thus, simplifying the following tests aimed at uniquely identifying and correcting the fault. In fact in some cases the passive testing identifies the unique fault.

For fault coverage, good fault coverage can be achieved in general, although passive testing has no control over the observed input/output sequence. Generally, fault coverage increases as the sequence length increases, since this potentially increases the number of visited transitions.

B. *Possible Extensions*

There are a number of issues and problems that could be investigated further. Some of these are briefly discussed in what follows.

More than one fault: Multiple faults in the system will complicate the process of fault management.

How should a network be cut to provide best fault location?

Non-determinism: This means that the L^k sequence is not necessarily monotonically non-increasing, i.e. it may never converge to a singleton set. However, whenever the condition $L^k = \varphi$ exists the same analysis can be applied.

Another issue is whether further passive testing, beyond when a fault is detected, could be used to provide better fault identification and coverage.

Studying fault coverage for faults identified in the channels can be revisited either with adding more assumptions for faults in channels, or changing the way fault coverage is calculated to include these types of faults as well.

Another challenge is to see how the techniques that have 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.

One last major extension in our passive testing is to include the timing of faults as a new dimension to our model. Although the real-time dimension might appear orthogonal to our fault management work, it could still add robustness to our passive testing results. Real-time measurements in passive testing could provide "changes in performance" rather than "faulty indication". Along with our passive testing suite (fault detection, location, identification and coverage), this might enable one to decide when and where the performance flaw happens and provide some guidance to take corrective actions.

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 and K. Arisha, “On Fault Location in Networks by Passive Testing,” 2000 IEEE International Performance Computing and Communications Conference, February 2000.
[3] R. E. Miller and K. Arisha, “On Fault Location in Networks by Passive Testing,” Technical Report #4044, Computer Science Dept., University of Maryland College Park, August 1999.
[4] R. E. Miller and K. Arisha, “On Fault Identification in Networks by Passive Testing,” Technical Report CS TR#4207/UMIACS TR#2001-03, Computer Science Dept., University of Maryland College Park, January 2001.
[5] R. E. Miller and K. Arisha, “Fault Identification in Networks by Passive Testing,” Advanced Simulation Technologies Conference (ASTC), April 2001.
[6] R. E. Miller and K. Arisha, “Fault Identification in Networks Using a CFSM Model by Passive Testing,” under preparation.
[7] R. E. Miller and K. Arisha, “On Fault Coverage in Networks by Passive Testing,” Technical Report CS TR#4220/UMIACS TR#2001-10, Computer Science Dept., University of Maryland College Park, February 2001.
[8] R. E. Miller and K. Arisha, “Fault Coverage in Networks by Passive Testing,” under preparation.
[9] R. E. Miller, “Passive Testing of Networks Using a CFSM Specification,” Bell Labs Technical Memorandum, BL011345-97-0522-03TM, May 20th, 1997.
[10] P. McCain and G. Roman, “Modeling Mobile IP in Mobile UNITY,” ACM Trans. on Software Engineering and Methodology, Vol. 8, No.2, April 1999, pp. 115-146.
[11] C. Perkins, “IP Mobility Support,” IETF Network Working Group, RFC 2002, October 1996.
[12] C. Huitema, “IPv6, The New Internet Protocol,” Prentice Hall, 2nd Edition, 1998.
[13] W. Stallings, “SNMP, SNMPv2, and CMIP The Practical Guide to Network-Management Standards,” Addison-Wesley Publishing Company, 1993.
[14] ISO/IEC 7498-1: 1994 | ITU-T Recommendation X.200 (1994) Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model, 1994.
[15] D. Brand and P. Zafiropulo, “On Communicating Finite-State Machines,” JACM, Vol. 30, No. 2, pp. 323-42, April 1983.