# On Fault Location in Networks 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 a variant of the communicating finite state machine (CFSM) model for networks to investigate fault detection and location using passive testing. First, we introduce the concept of passive testing, then we introduce the model with necessary assumptions and justification. Then, the model for the observer process is described and a 3-node case is studied to show how fault location information can be deduced. Extending this result, we propose a multiple node-cut approach for a general network, applying our technique for fault detection and location. An abstraction of a node-cut shows how the 3-node case can be used in the general case. We then illustrate our technique through a simulation of a practical X.25 example. Finally future extensions and potential trends are discussed.*

## 1. Introduction

Today's networks are becoming larger and more complex, sophisticated, heterogeneous and geographically dispersed. Consequently, network management is becoming more important. In many applications, end-to-end performance should be guaranteed at an acceptable level. The need for automated management tools increases as the complexity of networks rises.

One of the critical aspects of network management is fault management. Fault management includes determining, locating, isolating and correcting faults in the network. Fault management is often approached by devising test messages that can be injected into the network to gather information that will aid in finding network faults. Such "active" testing is also commonly used for determining whether an implementation of a protocol conforms to the protocol specification. For conformance testing test sequences are generated from the known specification and these input sequences are then applied to the implementation to see if the output sequence is identical to the one given by the specification. Once a network is in operation, however, it is desirable to keep the network testing traffic as low as possible so that its effect on normal traffic is minimized. This gives rise to looking at "passive testing." That is, by simply observing the normal traffic on the network, without adding any test messages and seeing what fault management can be accomplished in this way. Passive testing is an approach that enables us to examine the input-output behavior of a network without forcing the system to specialized inputs explicitly for testing. We investigate how passive testing can be used to detect and locate faults in the network.

In the approach, by Lee et al [1], a network is specified by a finite-state machine *A* -which models the desired behavior of the network. Then, given an implementation *B*, which is the network under test, *B* is viewed as a "Black Box", where only its input-output behavior can be observed. The problem is to determine whether *B* has faults, i.e., does the behavior of *B* conform to the behavior of the specification *A*?

The approach in [1] provides fault detection by observing the input/output sequence of *B* and comparing it with the specified behavior of *A*. However, since the model specifies the network as a single finite-state machine, no isolation of a fault to a particular region of the network is possible.

In [2] a variant of the communicating finite-state machine model, used for protocol specification and analysis, was introduced to serve as the specification model for a network. In this paper a characterization of the faults was defined and it was shown that some fault location information could be deduced by passive testing. We describe this model in section 2 of this paper. Then, in section 3, we describe the observer model and its fault detection algorithm, and also show how this provides some fault location capability.

In section 4, we apply this approach to a 3-node system. Then, in section 5, we extend the concept to a multiple node-cut approach that can be applied to any general network. In Section 6, we introduce an abstraction of the node-cut based network to show how it can be reduced to our 3-node case. Section 7 describes an experiment on a practical network example that simulates our model and technique and demonstrates the effectiveness of its fault detection and location capability. Finally section 8 discusses the results and possible extensions.

## 2. The Specification Model

In this section we introduce two models for specification. The first model is defined by [1] where the system is modeled as one Finite State Machine (FSM). The other system is introduced by [2] where the system is modeled as a variant of the Communicating Finite State Machine model (CFSM), where each FSM is the same type as in [1], where input/output labels are on transitions. In this paper we restrict our attention to deterministic machines.

### 2.1. The FSM model

A deterministic finite state machine (DFSM) $M$ 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.
  - $\delta$: $S*I \rightarrow S$ is the state transition function;
  - $\lambda$: $S*I \rightarrow O$ 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 $\delta$ and the output function $\lambda$ 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 $b_i=\lambda(s_i,a_i)$, $i=0,1,\ldots,k$.

### 2.2. The CFSM model

Our model here is based on the 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 nodes, one for each node of the network, with channels connecting these nodes. 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,
- $C=\{C_{ij}: i,j \leq r \wedge i \neq j\}$ 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 $m_j$.

## 2.3. Assumptions for the model and justifications

A) Atomic Transition Assumption

1) We assume each node of the network is represented as a machine in *M*, and each one way communication channel from one node to the other is represented by a channel in *C*. A transition for a given node represents reading an input from one or more channels entering the node, and immediately placing an output on one or more of the outgoing channels from that node. So, the transition is viewed as an atomic step.
2) For multiple incoming or outgoing channels for the same node, a numbering scheme will be used to represent each channel. Then, for a node having *k* incoming channels, the input to it is a *k*-tuple with the $i^{th}$ component representing the input being read from the $i^{th}$ incoming channel. Similarly, for the output channels.
3) The atomic step will be recognized as a step by observation. Each machine will have transitions labeled by input-output tuples, where the input tuple represents incoming symbols via input channels, and the output tuple represents outgoing symbols via output channels. Some entries can be represented by a "—", or "none" symbol. This symbol indicates that this step neither reads an input symbol from this input channel, nor writes an output symbol on this output channel for this atomic step. We assume that not all entries of the *k*-tuple are "—". Thus, there are no pure internal steps, which would be unobservable. Thus, an atomic step is always "observable", occurs using "local information" at the node as well as the input tuples incoming from channels and the output tuples outgoing to channels.

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 specification machine could not produce the observed input/output sequence.

C) Deterministic Machines

1) 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.

2) 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 CFSM specification.

# 3. Observers

Each observer will be placed at a certain node in the network. Let *A* represent the machine specification at the node where the observer is placed. Since the observer knows the structure of *A*, it can trace *i/o* tuples with the 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*.

## 3.1. Assumptions

1) We assume the network topology of the implementation is the same as of 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 three-node network we will be considering first, 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.

## 3.2. Fault Detection Algorithm

As described in [1], the fault detection capability of the observer can be summarized as follows:
We compare the observed input/output sequence of the implementation machine *B* (of the node where the observer is placed) with the expected behavior of the specification machine *A* of that node. Notice that we are talking about the machine representing the node where the observer is located. *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 to observe the input/output sequence from *B*.

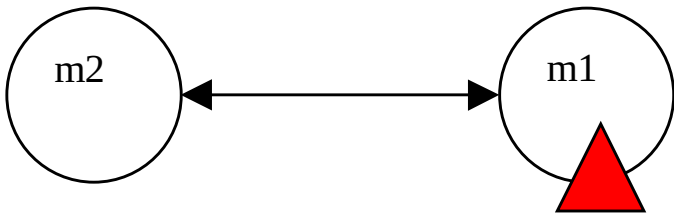



**Figure 1 A two-node model**

Assume that our observer is placed at machine $m_1$, and the rest of the system is viewed as a single machine $m_2$ along with channels $C_{12}$ and $C_{21}$. The observer's fault detection procedure can be described as follows:

a) Since we do not know the initial state of *B*, we assume the observer can be initially in any of the states of *A*. Let $L^0$ designate this initial set of possible states.

b) Once the observer observes the first tuple of inputs $(i_1)$ incoming to the node together with the corresponding output tuple $(o_1)$ outgoing from the node (at the same moment since we already assumed atomic transition), 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 $L^0, L^1, \ldots, L^{j-1}, L^j$ 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 *i/o* tuples will join the observation sequence $e=i_1/o_1, i_2/o_2, \ldots i_{j-1}/o_{j-1}, i_j/o_j$. Consequently, the sequence of sets of possible states $L=L^0, L^1,\ldots, L^{j-1}, L^j$ 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 network behavior conforms to that of the specification up to that point.

g) If set $L^k$ becomes empty, this indicates a fault in the network. 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*.

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 to recover from unobservable transitions and non-deterministic machine specifications are also provided in [1].

## 3.3. Fault Detection Capability

As notation we are going to use the set of symbols $\{f^{ij}\}$ to indicate faulty output symbols from machine $m_i$ to machine $m_j$. This symbol indicates that $m_i$ produced a faulty output symbol that $m_j$ receives as an invalid input (i.e. causing an unspecified transition in the specification machine). 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 $\in \{f^{ij}\}$).

The process to introduce the fault symbols into a specification machine can be described as follows:

Given: a machine $A$ for $m_i$, we form $A'$, a complete deterministic FSM, and the implementation $B$, to be correct has to exhibit the behavior of $A'$.
Steps to produce $A'$ from $A$:

1) If $\delta(s, i_k^j)$ is undefined in $A$,
   then $\delta(s, i_k^j)=F^i$ in $A'$ and $\lambda(s, i_k^j)=f^{ij}$, $\forall_j(s, i_k^j)$ in $A$.
2) $S_{A'}=S_A \cup \{F^i\}$.
3) If $\delta(s, i_k^j)=s'$ in $A$ and $\lambda(s, i_k^j)=o$ in $A$,
   then $\delta(s, i_k^j)=s'$ in $A'$ and $\lambda(s, i_k^j)=o$ in $A'$.

$A$ is incomplete only in the way that if a transition is defined, then an *i/o* label is on the transition (both next state and output are define). So we are strictly excluding the cases of only undefined next state, or only undefined output.

For input symbols not specified in the specification machine, the implementation machine will act as if it goes to a fault state and produces an output symbol $f^{ij}$, which we call the fault symbol for machine $m_i$ to machine $m_j$. That fault symbol can appear as follows:

- Output = $f^{ij}$ (the fault symbol for observed node $m_i$; which indicates an unspecified transition in the specified machine that was substituted by a transition to a faulty state with an output fault symbol (for completely specified FSM to describe the behavior desired of the implementation machine)),
- Input = $f^{ji}$ (the fault symbol from external node $m_j$, indicating a faulty output from machine $m_j$),
- Input is not legal at all i.e. $i_k^j \notin I$ *and* $i_k^j \neq f^{ji}$.

Now let us consider again the two-node case. We are assuming that we are going to detect a fault always when we have an empty $L^k$ with a non-empty $L^{k-1}$ with the observed *i/o* sequence $e=i_1/o_1, i_2/o_2,\ldots, i_{k-1}/o_{k-1}, i_k/o_k$. The following results, assuming a single fault occurs in the network, are known:

1. Type A fault:
   $o_k \varepsilon O$ of $m_1 \rightarrow$ the fault is in $m_1$.
2. Type $B_a$ fault:
   $o_k = f^{12} \wedge$ no $s \in L^{k-1}$ has $\lambda(s, i_k)=f^{12} \rightarrow$ the fault is in $m_1$.
3. Type $B_b$ fault:
   $o_k = f^{12} \wedge \exists\, s \in L^{k-1}$ has $\lambda(s, i_k)=f^{12} \rightarrow$ the fault is in $m_1$ or outside $m_1$.
4. Type $C_a$ fault:
   $i_k \notin I \wedge i_k \neq f^{21} \rightarrow$ the fault is outside $m_1$.
5. Type $C_b$ fault:
   $i_k \notin I \wedge i_k = f^{21} \rightarrow$ the fault is in $m_1$ or outside $m_1$.

The results and their proofs appear in [2].

With what we have introduced so far, we should understand that having an observer at any node gives us an indication of whether a single fault has occurred, and if so it provides some fault location information possibilities except when the fault is of type $B_b$ or $C_b$. As is obvious, we are considering modeling the network as a 2-node case, where the observer is located at one of the nodes. In the next section, we apply the same approach to the 3-node case representation of the network. We will show that this 3-node case helps, in the sense that it provides better fault location information.

## 4. The three node case

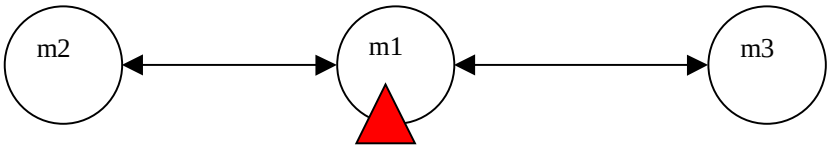
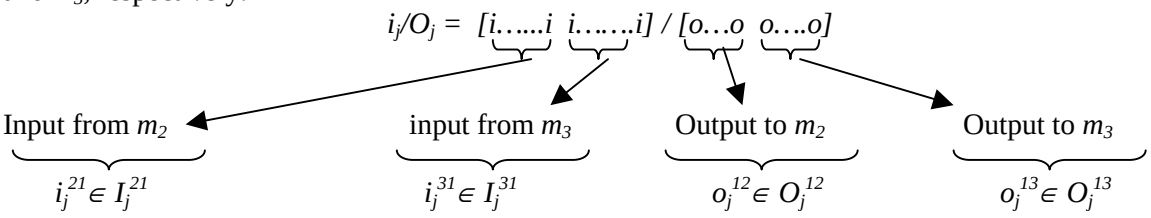



**Figure 2 A three-node model**

For the above configuration, assume $i_j$ and $o_j$ for $m_1$ consist of input tuples and output tuples between $m_2$ and $m_3$, respectively.




Now, the above fault results for the 2-node case can be applied to both sides the observer is observing. Our original fault characterization works for the 2-node case where the observer is located at one node. For the 3-node case, the observer functionality can be visualized as if the observer is having two modules working in parallel with a third module to collect information from these two modules giving final results from the observer as a single entity. So, we can analyze the system as if there is one module observing the subsystem ($m_1$-$m_2$), and the other observing the subsystem ($m_1$-$m_3$). Each of these modules will look like the original 2-node case. Combining fault location information (if any) from both modules provides more precise location information as explained later.

Assume we have the symbol $f^{ij}$ for the output fault symbol from $m_i$ to $m_j$, and the symbol $f^{ji}$ for fault output symbol of $m_j$ as an input to $m_i$.
Also assume the set $O^{ij}$ as the output set for symbols from $m_i$ to $m_j$, and the set $O^{ji}$ as the output set for symbols from $m_j$ to $m_i$.
Similarly, for the input sets $I^{ij}$ and $I^{ji}$ of input symbols.

For the subsystem ($m_1$-$m_2$), we have the following cases for fault types to be observed:

A) No Fault: i.e. $L^k \neq \varphi$.
B) Transfer (input) faults: $L^k = \varphi$ and one of the following:

1. $i_k \in I^{21}$
2. $i_k = f^{21}$
3. $i_k \notin I^{21} \cup \{ f^{21} \}$

C) Output faults: $L^k = \varphi$ and one of the following:

1. $o_k \in O^{12}$
2. $o_k = f^{12}$
3. $o_k \notin O^{12} \cup \{ f^{12} \}$

Similarly, for the subsystem ($m_1$-$m_3$).

We now map these cases back to our fault characterization types A, $B_a$, $B_b$, $C_a$ and $C_b$. Then we show how the combined results from both modules provide fault location results for the whole system from the *i/o* sequence observed at $m_1$.

## 4.1. Mapping to fault characterization

Here, we map the fault types described above with the previously referenced fault characterization [2]. As a notation, the symbol $(m_1 — m_2)$ denotes the subsystem including $m_1$, $m_2$, $c_{12}$, and $c_{21}$. Also the symbol $(— m_2)$ indicates $m_2$, $c_{12}$, and $c_{21}$.

Case A: $L^k \neq \varphi$, no fault has been detected in the subsystem $(m_1\text{-}m_2)$.

Case B: implies there is some fault in subsystem $(m_1\text{-}m_2)$ such that:

1. There is no input fault, so we have to check for the output fault characterization.
2. There is a type $C_b$ fault.
3. There is a type $C_a$ fault.

Case C: Implies there is an output fault in subsystem $(m_1\text{-}m_2)$ such that:

1. There is a type A fault.
2. There is a type *B* fault:
   It is of type $B_a$ if there is a state $s \in L^{k-1}$ with $\lambda(s, i_k^2)=f^{12}$.
   It is of type $B_b$ if there is no state $s \in L^{k-1}$ with $\lambda(s, i_k^2)=f^{12}$.
3. The case of $o_k \notin O^{12} \cup \{ f^{12}\}$ was not characterized in [2]. As shown below this case implies that the fault is in $m_1$.

Theorem 1: If the network contains at most one fault and the observer at $m_1$ records $o_k \notin O^{12} \cup \{ f^{12}\}$ then the fault is in $m_1$.

Proof: Since $L^{k-1} \neq \varphi$, no fault has been observed yet, but now with $i_k/o_k$ observed we find $o_k \notin O^{12} \cup \{ f^{12}\}$. Clearly for each state in $L^{k-1}$ there is a transition leaving each state in $L^{k-1}$ under input $i_k$ (in the completed machine) with output defined either in $O^{12}$ or $f^{12}$. Thus, the single fault could be in $m_1$, i.e. the implementation is simply wrong in producing this undefined output. Now consider if the fault could be in $(— m_2)$. If $i_k \in I^{12}$ then with a correct implementation of $m_1$, we would have had an output $o_k \in O^{12} \cup \{ f^{12}\}$. This implies that $i_k \notin I^{12}$ causing $o_k \notin O^{12} \cup \{ f^{12}\}$. However, this $i_k \notin I^{12}$ would be a case of $C_a$ or $C_b$. But since we consider inspection of input faults before output faults, this kind of fault would be already detected as a different case. Thus, the fault cannot be a result of an error in $(— m_2)$. The fault is in $m_1$.

## 4.2. Merging fault location information from Module$_{12}$ and Module$_{13}$

1. Input faults: $L^k = \varphi$ and one of the following cases:

   a) $\boldsymbol{i_k^{21} \in I^{21} \wedge i_k^{31} \in I^{31}}$

   For this case, both inputs are valid and are elements of the input set. But since $L^k = \varphi$, there must be an error somewhere in the system. To characterize this fault, the output is checked as discussed in 2 below, where various location results are obtained depending upon the outputs observed.

   b) $\boldsymbol{i_k^{21} = f^{21} \wedge i_k^{31} \in I^{31}}$

   Theorem 2: If the network contains at most one fault and the observer at $m_1$ records $i_k^{21} = f^{21}$ and $i_k^{31} \in I^{31}$ then the fault is in $m_1$, $m_2$, $c_{12}$, or $c_{21}$.

Proof: The input from $m_2$ is faulty. As we mentioned before, this is a type $C_b$ fault. The fault can be in sub CFSM $(m_1 — m_2)$. This can be due to one of the following:

- The previous output $o_{k-1}^{12}$ was sent to $m_2$, and for some faulty operation in $m_2$ the machine $m_2$ produces $f^{21}$ to $m_1$.
- $o_{k-1}^{12}$ was sent to $m_2$ and due to a fault in the channel $c_{12}$, it was received faulty at $m_2$. So $m_2$ receives an unexpected input and produces as output $f^{21}$ as a response to that faulty input.
- $o_{k-1}^{12}$ was sent to $m_2$ and $m_2$ reacts appropriately with a valid output, but due to a fault in the channel $c_{21}$ the symbol has been changed to $f^{21}$, $m_1$ receives the input $f^{21}$ rather than the correct input symbol sent from $m_2$.
- For $o_{k-1}^{12}$ machine $m_1$ produced an output in the output set, but different from the specified output. This was received by $m_2$ as a valid input, but $m_2$ was in a state that had no transition specified for this input. Thus $m_2$ responded with $f^{21}$.

Since all above cases are possible, this leads to the conclusion that the fault is in the sub CFSM $(m_1 — m_2)$.

Although we could not determine exactly which machine is faulty in this case, this still provided error location separation to one side of the cut (including the cut itself). That is, the fault is not in $m_3$, $c_{13}$ or $c_{31}$.

c) $\boldsymbol{i_k^{21} \notin I^{21} \cup \{f^{21}\} \wedge i_k^{31} \in I^{31}}$

Theorem 3: If the network contains at most one fault and the observer at $m_1$ records $i_k^{21} \notin I^{21} \cup \{f^{21}\}$ and $i_k^{31} \in I^{31}$ then the fault is in $m_2$, $c_{12}$, or $c_{21}$.

Proof: The input from $m_2$ is invalid. This is a type $C_a$ fault. The fault can be in sub CFSM $(— m_2)$, i.e. error in $m_2$, $c_{12}$, or $c_{21}$. This is because either a faulty $m_2$ can send an invalid input $i_k^{21}$, or the channel $c_{21}$ can modify the valid output of $m_2$ to be not in the input set of $m_1$, or $c_{12}$ could modify a valid output $o_{k-1}^{12}$ of $m_1$ to an invalid input to $m_2$ causing $m_2$ to send the invalid output to $m_1$. Now assume that the fault could be in $m_1$. Since there is at most one fault in the network, this means that $m_2$, $c_{12}$, and $c_{21}$ are all fault free. Since $L^{k-1}$ is not empty, the $o_1^{12}, \ldots, o_{k-1}^{12}$ is a valid output sequence of $m_1$ which is fault free transmitted to $m_2$ as a valid input sequence to $m_2$ for which $m_2$ produces a valid output. This valid output becomes the input $i_k^{21}$ to $m_1$ since it is transmitted fault free by $c_{21}$ to $m_1$. But, since the network is specified correctly, the output set of $m_2$ is the input set of $m_1$ contrary to the assumption that $i_k^{21} \notin I^{21} \cup \{f^{21}\}$. Thus the fault could not be in $m_1$. Also the fault could not be from $(— m_3)$ side since $i_k^{31} \in I^{31}$. Thus we conclude that the fault is in $(— m_2)$.

d) $\boldsymbol{i_k^{21} \in I^{21} \wedge i_k^{31} = f^{31}}$

Theorem 4: If the network contains at most one fault and the observer at $m_1$ records $i_k^{21} \in I^{21}$ and $i_k^{31} = f^{31}$ then the fault is in $m_1$, $m_3$, $c_{13}$, or $c_{31}$.

The proof is analogous to case b).

e) $\boldsymbol{i_k^{21} \in I^{21} \wedge i_k^{31} \notin I^{31} \cup \{f^{31}\}}$

Theorem 5: If the network contains at most one fault and the observer at $m_1$ records $i_k^{21} \in I^{21}$ and $i_k^{31} \notin I^{31} \cup \{f^{31}\}$ then the fault is in $m_3$, $c_{13}$, or $c_{31}$.

The proof is analogous to case c).

f) $i_k^{21} = f^{21} \wedge i_k^{31} = f^{31}$

Theorem 6: If the network contains at most one fault and the observer at $m_1$ records $i_k^{21} = f^{21}$ and $i_k^{31} = f^{31}$ then the fault is in $m_1$.

Proof: If faults come from $(\text{—}m_2)$ and $(\text{—}m_3)$ then there are 2 faults, which are assumed not to happen (the single fault assumption). So, the only conclusion can be the machine $m_1$ sent an output tuple $(o_{k-1}^{12}, o_{k-1}^{13})$ where $o_{k-1}^{12} \in O_{k-1}^{12}$ and $o_{k-1}^{13} \in O_{k-1}^{13}$ but both of these output values were different than the specified outputs for $(i_{k-1}^{21}, i_{k-1}^{31})$. This affected both $m_2$ and $m_3$ to make them both to produce faulty output symbols, which were transmitted as the input $(f^{21}, f^{31})$ to $m_1$, i.e. the output tuple $o_{k-1}$ was wrong from $m_1$. Here, we conclude the single fault was in $m_1$.
Thus for this case, using the single fault assumption, we get some fault location information with possible overlap on $m_1$ (the cut).

g) $i_k^{21} = f^{21} \wedge i_k^{31} \notin I^{31} \cup \{f^{31}\}$

Theorem 7: If the observer at $m_1$ records $i_k^{21} = f^{21}$ and $i_k^{31} \notin I^{31} \cup \{f^{31}\}$ then a single fault in the network could not be the cause of the faulty behavior.

Proof: Since $L^{k-1} = \varphi$, the input/output tuple $(i_{k-1}^{21}, i_{k-1}^{31})/ (o_{k-1}^{12}, o_{k-1}^{13})$ was a specified *i/o* for some state of $m_1$ in $L^{k-2}$. Thus, no fault in $m_1$ had yet been observed. Although $o_{k-1}^{12} \in I^{12}$, it could have been a different output than specified causing $m_2$ to output $f^{21}$ as $i_k^{21}$. Also, some fault in $(m_1\text{—}m_2)$ could have caused $i_k^{21} = f^{21}$ so a fault is in $(m_1\text{—}m_2)$. Similarly, some fault in $(\text{—}m_3)$ could have caused $i_k^{31} \notin I^{31} \cup \{f^{31}\}$. Now a fault in $(m_1\text{—}m_2)$ means a fault in $m_1$, $m_2$, $c_{12}$, or $c_{21}$, and a fault in $(\text{—}m_3)$ means a fault in $m_3$, $c_{13}$, or $c_{31}$. There is no overlap here, so this could not have arisen from a single fault of the types we are assuming.

h) $i_k^{21} \notin I^{21} \cup \{f^{21}\} \wedge i_k^{31} = f^{31}$

Theorem 8: If the observer at $m_1$ records $i_k^{21} \notin I^{21} \cup \{f^{21}\} \wedge i_k^{31} = f^{31}$, a single fault in the network could not be the cause of the faulty behavior.

The proof is analogous to case g).

i) $i_k^{21} \notin I^{21} \cup \{f^{21}\} \wedge i_k^{31} \notin I^{31} \cup \{f^{31}\}$

Theorem 9: If the observer at $m_1$ records $i_k^{21} \notin I^{21} \cup \{f^{21}\} \wedge i_k^{31} \notin I^{31} \cup \{f^{31}\}$, a single fault in the network could not be the cause of the faulty behavior.

Proof: Since $L^{k-1} = \varphi$, the input/output tuple $(i_{k-1}^{21}, i_{k-1}^{31})/ (o_{k-1}^{12}, o_{k-1}^{13})$ was a specified *i/o* for some state of $m_1$ in $L^{k-2}$. Thus, no fault in $m_1$ had yet been observed. Some fault in $(\text{—}m_2)$ could have caused $\notin I^{21} \cup \{f^{21}\}$ so fault is in $(\text{—}m_2)$. Similarly, some fault in $(\text{—}m_3)$ could have caused $i_k^{31} \notin I^{31} \cup \{f^{31}\}$. Now a fault in $(\text{—}m_2)$ means a fault in in $m_2$, $c_{12}$, or $c_{21}$, and a fault in $(\text{—}m_3)$ means a fault in $m_3$, $c_{13}$, or $c_{31}$. There is no overlap here, so this cannot arise from a single fault of the types we consider.

| $\mathbf{Mo_{13} \setminus Mo_{12}}$ | $i_k^2 \in I^{21}$ | $i_k^2 = f^{21}$ | $i_k^2 \notin I^{21} \cup \{f^{21}\}$ |
|---|---|---|---|
| $i_k^3 \in I^{31}$ | Look for $o_k$ | $C_b \rightarrow (m_1 — m_2)$ | $C_a \rightarrow (— m_2)$ |
| $i_k^3 = f^{31}$ | $C_b \rightarrow (m_1 — m_3)$ | $m_1$ | X |
| $i_k^3 \notin I^{31} \cup \{f^{31}\}$ | $C_a \rightarrow (— m_3)$ | X | X |

**Table 1 Merging input-fault information**

Table 1 summarizes the results of theorems 2 through 9. Now Case 1(a) is considered further by looking at the observed output sequence of $(o_k^{12}, o_k^{13})$

2. Output faults: $L^k = \varphi$, the input tuple $(i_k^2, i_k^3) \in (I^{21}, I^{31})$ (otherwise type $C_a$ or $C_b$ should be detected) and one of the following:

***a) $o_k^{12} \in O^{21} \wedge o_k^{13} \in O^{31}$***

Theorem 10: If the network contains at most one fault and the observer at $m_1$ records $o_k^{12} \in O^{21} \wedge o_k^{13} \in O^{31}$ then the fault is in $m_1$.

Proof: Since the *i/o* sequence is observed from the implementation of $m_1$ we can conclude that it is a valid *i/o* for some state in the implementation of $m_1$. However, since $L^k$ is empty we conclude that the observed *i/o* sequence is not specified for any state in the specification of $m_1$. Therefore, the implementation $m_1$ is faulty.

***b) $o_k^{12} = f^{12} \wedge \exists$ no $s \in L^{k-1}$ has $\lambda(s, i_k^{21}) = f^{12} \wedge o_k^{13} \in O^{13}$***

Theorem 11: If the network contains at most one fault and the observer at $m_1$ records $o_k^{12} = f^{12}$ and $\exists$ no $s \in L^{k-1}$ has $\lambda(s, i_k^{21}) = f^{12}$ and $o_k^{13} \in O^{13}$ then the fault is in $m_1$.

Proof: Since there is no state $s \in L^{k-1}$ in the specification that has $\lambda(s, i_k^{21}) = f^{12}$ it must be that the specification has an output $(o_k^{12}, o_k^{13})$ defined under $(i_k^{21}, i_k^{31})$ for every state in $L^{k-1}$. However, the observed output of $o_k^{12} = f^{12}$ means that the implementation is faulty. Thus, $m_1$ is faulty.

***c) $o_k^{12} = f^{12} \wedge \exists s \in L^{k-1}$ has $\lambda(s, i_k^{21}) = f^{12} \wedge o_k^{13} \in O^{13}$***

Theorem 12: If the network contains at most one fault and the observer at $m_1$ records $o_k^{12} = f^{12}$ and $\exists s \in L^{k-1}$ has $\lambda(s, i_k^{21}) = f^{12}$ and $o_k^{13} \in O^{13}$ then the fault is in $m_1$, $m_2$, $c_{12}$, or $c_{21}$.

Proof: This is a typical $B_b$ case, where:

- The implementation of $m_1$ can be faulty. If the specification has $\delta(s, i_{k-1}) = s'$ and $\lambda(s, i_k^{21}) = f^{12}$, whereas the implementation has this transition could go erroneously to $\delta(s, i_{k-1}) = s''$ where $s'' \neq s'$. This leads to a fault in $m_1$, or
- The fault can be in $(— m_2)$. This fault can change the expected input $i'^{21}_k$ (according to the specification) to the observed input $i_k^2$, and for the specification $\lambda(s, i'^{21}_k) \in O^{12}$, while in the implementation $\lambda(s, i_k^{21}) = f^{12}$ since the transition was undefined in the specification.

So, the fault can be in either $m_1$ or in $(— m_2)$, i.e. there is a fault in $(m_1 — m_2)$.

So, again we have a fault location to one side of the cut (including the cut itself).

***d) $o_k^{12} \in O^{12} \wedge o_k^{13} = f^{13} \wedge \exists$ no $s \in L^{k-1}$ has $\lambda(s, i_k^{31})=f^{13}$***

Theorem 13: If the network contains at most one fault and the observer at $m_1$ records $o_k^{12} \in O^{12}$ and $o_k^{13} = f^{13}$ and $\exists$ *no* $s \in L^{k-1}$ *has* $\lambda(s, i_k^{31})=f^{13}$ then the fault is in $m_1$.

The proof is analogous to case b).

***e) $o_k^{12} \in O^{12} \wedge o_k^{13} = f^{13} \wedge \exists s \in L^{k-1}$ has $\lambda(s, i_k^{31})=f^{13}$***

Theorem 14: If the network contains at most one fault and the observer at $m_1$ records $o_k^{12} \in O^{12}$ and $o_k^{13} = f^{13}$ and $\exists s \in L^{k-1}$ has $\lambda(s, i_k^{31})=f^{13}$ then the fault is in $m_1$, $m_3$, $c_{13}$, or $c_{31}$.

The proof is analogous to case c).

***f) $o_k^{12} = f^{12} \wedge \exists$ no $s \in L^{k-1}$ has $\lambda(s, i_k^{21})=f^{12} \wedge o_k^{13} = f^{13} \wedge \exists$ no $s \in L^{k-1}$ has $\lambda(s, i_k^{31})=f^{13}$***

Theorem 15: If the network contains at most one fault and the observer at $m_1$ records $o_k^{12} = f^{12}$ and $\exists$ *no* $s \in L^{k-1}$ has $\lambda(s, i_k^{21})=f^{12}$ and $o_k^{13} = f^{13}$ and $\exists$ *no* $s \in L^{k-1}$ *has* $\lambda(s, i_k^{31})=f^{13}$ then the fault is in $m_1$.

Proof: In that case, we have for both modules (of the observer) reporting a type $B_a$ fault, meaning both modules indicate a fault in machine $m_1$. Thus, our final conclusion is that $m_1$ is faulty.

***g) $o_k^{12} = f^{12} \wedge \exists s \in L^{k-1}$ has $\lambda(s, i_k^{21})=f^{12} \wedge o_k^{13} = f^{13} \wedge \exists$ no $s \in L^{k-1}$ has $\lambda(s, i_k^{31})=f^{13}$***

Theorem 16: If the network contains at most one fault and the observer at $m_1$ records $o_k^{12} = f^{12}$ and $\exists s \in L^{k-1}$ *has* $\lambda(s, i_k^{21})=f^{12}$ and $o_k^{13} = f^{13}$ and $\exists$ *no* $s \in L^{k-1}$ *has* $\lambda(s, i_k^{31})=f^{13}$ then the fault is in $m_1$.

Proof: For the module taking care of the $(m_1 — m_2)$ side, it detects a type $B_b$ fault meaning it detects a fault in $(m_1 — m_2)$. For the other module taking care of the $(m_1 — m_3)$ side, it detects a type $B_a$ fault meaning it detects a fault in $m_1$. When the observer merges both results it indicates that there is a fault in $m_1$, since we assume only one fault occurred, according to the single fault assumption.

***h) $o_k^{12} = f^{12} \wedge \exists$ no $s \in L^{k-1}$ has $\lambda(s, i_k^{21})=f^{12} \wedge o_k^{13} = f^{13} \wedge \exists s \in L^{k-1}$ has $\lambda(s, i_k^{31})=f^{13}$***

Theorem 17: If the network contains at most one fault and the observer at $m_1$ records $o_k^{12} = f^{12}$ and $\exists$ *no* $s \in L^{k-1}$ *has* $\lambda(s, i_k^{21})=f^{12}$ and $o_k^{13} = f^{13}$ and $\exists s \in L^{k-1}$ *has* $\lambda(s, i_k^{31})=f^{13}$ then the fault is in $m_1$.

The proof is analogous to case g).

***i)*** $o_k^{12} = f^{12} \wedge \exists s \in L^{k-1}$ ***has*** $\lambda(s, i_k^{21}) = f^{12} \wedge o_k^{13} = f^{13} \wedge \exists s \in L^{k-1}$ ***has*** $\lambda(s, i_k^{31}) = f^{13}$

Theorem 18: If the network contains at most one fault and the observer at $m_1$ records $o_k^{12} = f^{12}$ and $\exists s \in L^{k-1}$ *has* $\lambda(s, i_k^{21}) = f^{12}$ and $o_k^{13} = f^{13}$ and $\exists s \in L^{k-1}$ *has* $\lambda(s, i_k^{31}) = f^{13}$ then the fault is in $(m_1 — m_2 — m_3)$.

Proof: This means that the module taking care of the $(m_1 — m_2)$ side, it detects a type $B_b$ fault meaning it detects a fault in $(m_1 — m_2)$. Similarly, the module taking care of the $(m_1 — m_3)$ side, it detects a type $B_b$ fault meaning it detects a fault in $(m_1 — m_3)$. For this case, $m_1$ can be faulty as in type $B_b$ faults. Also assuming $m_1$ is correct and observing $(i_k^{21}, i_k^{31})/(f^{12}, f^{13})$ would imply faults in both $(— m_2)$ and $(— m_3)$, which is not possible according to the single fault assumption. However, a fault in either $(— m_2)$ or $(— m_3)$ can change $i_k^{21}$ or $i_k^{31}$ respectively. Any of the three cases could occur and no determination is possible to determine if the fault is in $m_1$, $m_2$, or $m_3$. In this case fault location separation cannot be improved, and we cannot determine more accurately where the fault is located. The fault is located somewhere in the whole system, using the notation $(m_1 — m_2 — m_3)$.

One can note the similarity of the proofs of our theorems with those of [2]. One might consider the theorems in this paper as corollaries of the theorems in [2], with the addition of using overlaps of regions.

Table 2 summarizes the results of theorems 10 through 18.

| $\mathbf{Mo_{13} \setminus Mo_{12}}$ | $o_k^{12} \in O^{12}$ | $o_k^{12} = f^{12} \wedge \exists$ **no** $s \in L^{k-1}$ **has** $\lambda(s, i_k^{21}) = f^{12}$ | $o_k^{12} = f^{12} \wedge \exists$ ***state*** $s \in L^{k-1}$ **has** $\lambda(s, i_k^{21}) = f^{12}$ |
|---|---|---|---|
| $o_k^{13} \in O^{13}$ | $A \rightarrow (m_1)$ | $A(m_1) \cap B_a(m_1) \rightarrow (m_1)$ | $(m_1 — m_2)$ |
| $o_k^{13} = f^{13} \wedge \exists$ **no** $s \in L^{k-1}$ **has** $\lambda(s, i_k^{31}) = f^{13}$ | $B_a(m_1) \cap A(m_1) \rightarrow (m_1)$ | $B_a(m_1) \cap B_a(m_1) \rightarrow (m_1)$ | $B_a(m_1) \cap B_b(m_1 — m_2) \rightarrow (m_1)$ |
| $o_k^{13} = f^{13} \wedge \exists$ ***state*** $s \in L^{k-1}$ **has** $\lambda(s, i_k^{31}) = f^{13}$ | $(m_1 — m_3)$ | $B_b(m_1 — m_3) \cap B_a(m_1) \rightarrow (m_1)$ | $(m_1 — m_2 — m_3)$ |

**Table 2 Merging output-fault information**

We now have some fault location capabilities by analyzing a 3-node case. This can be viewed as a node cut through a large network splitting the network into three parts: The cut, represented by $m_1$; one side of the cut, represented by $(— m_2)$; and the other side of the cut, represented by $(— m_3)$. Now, if all channels capacities are bounded, this finite state abstraction is a valid abstraction, even though the machines may become large. Now to get finer location we 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 single region, assuming that the $(m_1 — m_2 — m_3)$ case does not occur. We now discuss how multiple cuts might be made to provide this "small regions" fault location capability.

## 5. Multiple Cuts and Location of Observers

Now we have to decide "where" to place the observers in our network. To determine this criteria many graph division techniques can be studied and compared. For the purpose of this study, we choose the "Node Cut" approach. Our observers will be located at the nodes on this node cut. This approach lets us use the passive fault detection approach of [1] since each node is specified as a FSM. Then using the fault location results of [2] and overlap arguments we succeed in obtaining more detailed fault location information.

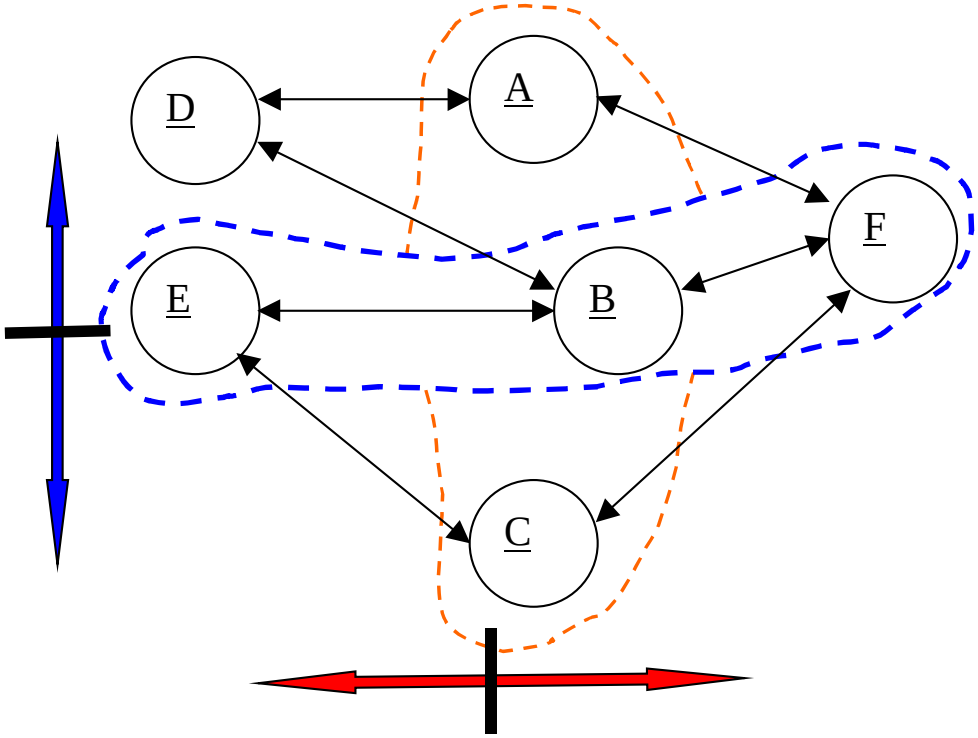



**Figure 3 A multiple-node-cut example**

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 to. For the above figure, the node cut passing through ABC can have 3 observers, one at each node over this node cut. By summing up 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 which node of that region is faulty as shown in the figure below.

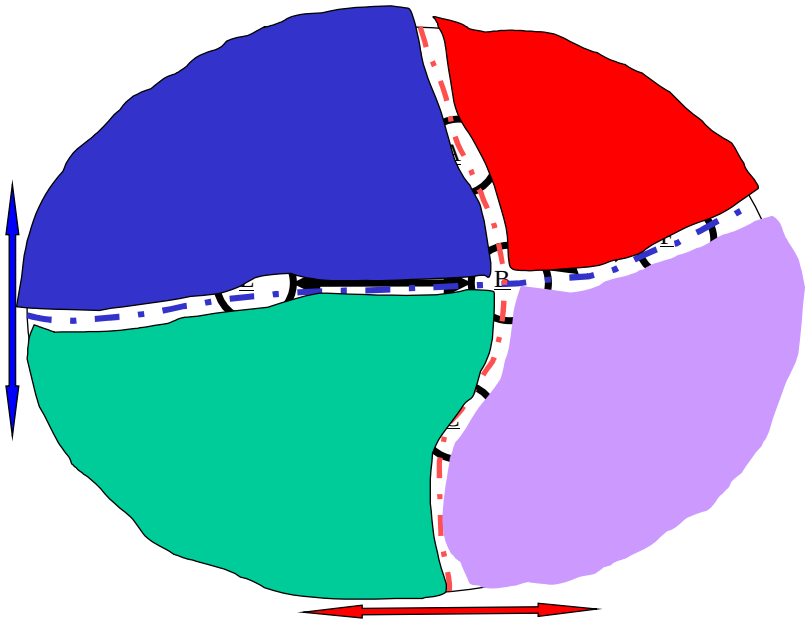


**Figure 4 The "small regions" resulting from 2 node cuts**

## 6. Abstraction to the Three Node Case

If we look at any node cut, we can abstract it to a 3-node configuration. Along the node cut, we will have two sub graphs, one on each side of the cut. We are calling the CFSM containing all nodes and channels -connecting these nodes with each other- on one side of the cut a "sub CFSM", and the other sub CFSM lies on the other side.

All nodes on the node cut is also a sub CFSM –i.e. with their internal channels included- can be abstracted as one node with the observer in it. All the nodes in the sub CFSM on one side of the node cut can be abstracted as one node to the left of the node cut node, and the sub CFSM to the right of the node cut as one node to the right of the node cut node.

We abstract also all channels connecting the sub CFSM nodes at the cut to the left sub CFSM as the abstract channel $C_{12}$. Similarly, the abstract channel $C_{13}$ represents all connections between sub CFSM nodes at the cut and the nodes of the right sub CFSM.

We have to add a new assumption here of "Bounded FIFO" Channel capacities so that we can represent them as FSM's. This reduces the problem to the simpler configuration of a 3-node case as described in section 4.

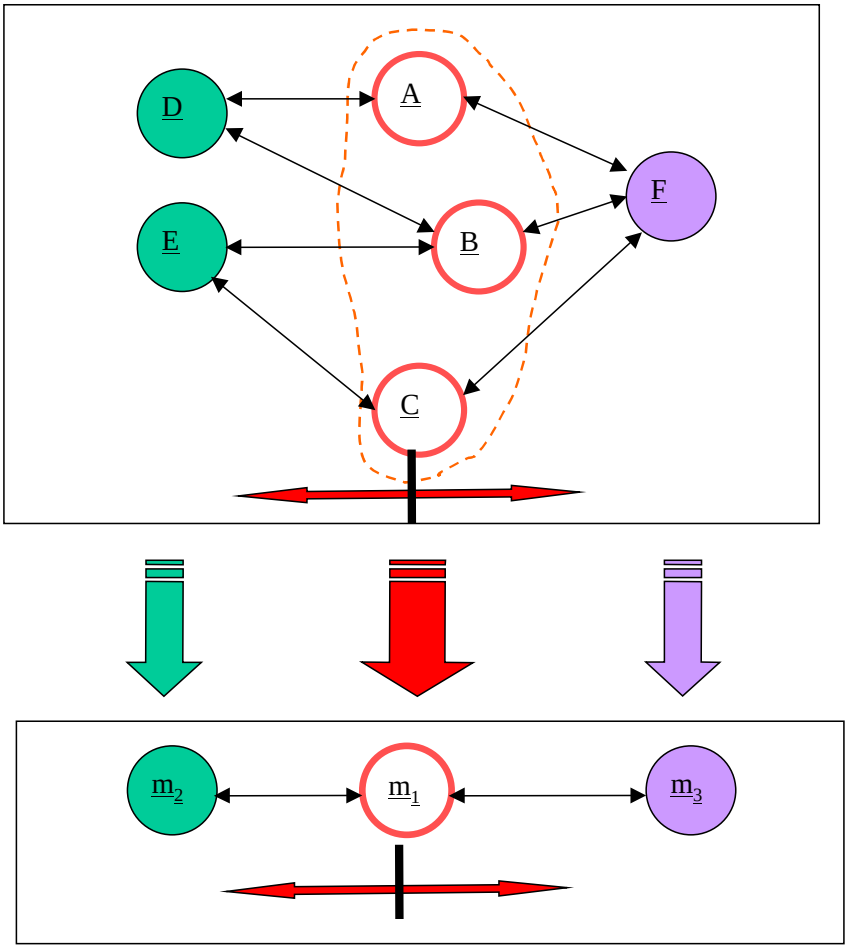



**Figure 5 Abstraction based on a node cut**

## 7. Experiment

We now investigate the effectiveness of our 3-node fault location approach with an experiment representing a practical network.

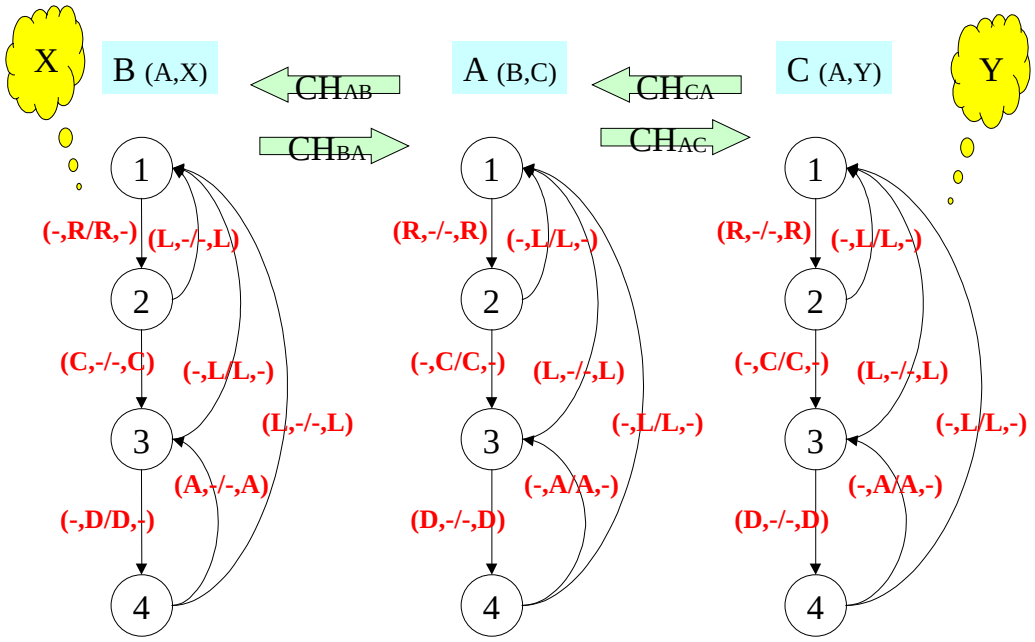



**Figure 6 A CFSM model for the experiment**

Our experiment is based upon a simulation of the 3-node case. The experiment is an abstraction of an X.25 network layer protocol [3] 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.

We place an observer at machine *A*. We inject faults randomly in the system. The random generator will choose:

- Fault class,
- Fault location (which machine, and which transition if needed,)
- Fault time (the time at which the fault so injected in the experiment).

Time is measured in atomic steps. One atomic step is equivalent to the time it takes a transition to be executed in one FSM.
The observer will report to the simulator when it detects the fault and what kind of fault location information it was able to deduce. Some details of the network and the experiment are given in appendix A.

The results from the experiment list the maximum, minimum and average time durations taken by the observer to detect the fault after it was injected into the system.

Simulation experiments have been run for quite a number of iterations and the summary of the results are as shown in the following table:

| Size of Experiment | Average Time | Maximum Time | Minimum Time |
|---|---|---|---|
| 50 | 7.987333 | 13.00000 | 3.800000 |
| 100 | 8.004706 | 13.70588 | 3.647059 |
| 500 | 7.847647 | 19.76471 | 1.058824 |
| 1000 | 8.095882 | 21.00000 | 0.941176 |
| 5000 | 8.105882 | 23.94118 | 0.941176 |
| 10000 | 8.050000 | 26.70588 | 0.941176 |
| 20000 | 8.014118 | 28.58824 | 0.941176 |
| 30000 | 8.114118 | 27.88235 | 0.941176 |

**Table 3 Experiment Result: Time to detect injected fault**

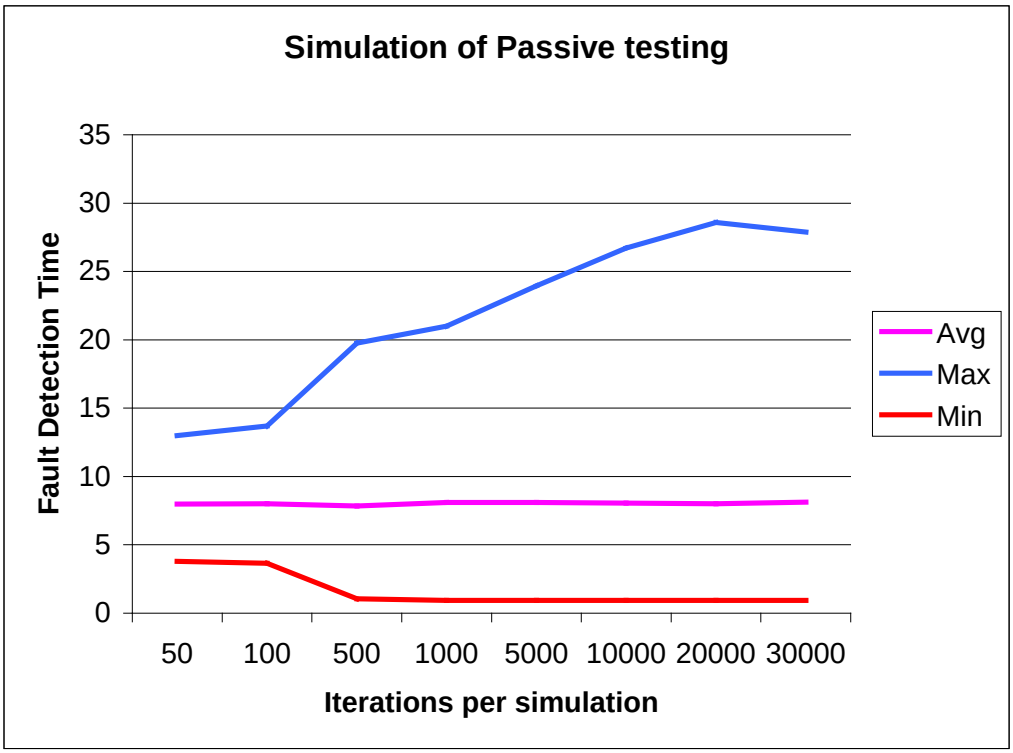



**Figure 7 Plot of Average, maximum and minimum fault detection time**

It can be noticed from the above table and chart that the average time to detect a fault in our experiment is quite low, i.e. it does not take long for the passive testing to detect a fault and to provide location information. The experiment also shows that fault detection using passive testing is quite stable over a wide range of experiment sizes.

## 8. Future 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 location since our technique may be confused on which side of the cut a fault exists. In addition to that, faults can "hide" each other so that it will appear to the observer as if the system is behaving properly. The multiple fault case may be handled under the condition that these faults are distant from each other, so that the effect of "fault hiding" does not confuse the observer.

How should a network be cut to provide best fault location? For this we propose that two approaches be investigated. The first is to apply cuts in the network to create equal (or nearly equal) parts, using a graph theoretical approach. The other is a more practical way; to apply the cuts at gateways of the network. These gateways represent logical separation between parts of the network, so division will have a practical value. This may also result in cuts that contain only a few nodes each.

Non determinism: This means that $L^k$ 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.

Study of an abstracted network vs. a detailed network: This is a very interesting point. It means whether to abstract the sub-CFSM as one node, or to study the sub-system keeping the original physical configuration. We need to investigate whether techniques can be developed to make abstract 3-node results useable for larger cuts.

More than one I/O pair to be processed (previous to $i_k/o_k$ in the past): This will be considered in a future extension for erroneous behavior, rather than the case *A* through $C_b$, i.e. taking into consideration both $i_{k-1}/o_{k-1}$ and $i_k/o_k$ together to characterize the fault types.

## 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. Miller, "Passive Testing of Networks using a CFSM Specification," 1998 IEEE International Performance Computing and Communications Conference, pp. 111-116, February 1998.

[3] ISO/IEC 8208, Third Edition, August 1995.

# 1. Appendix A: Conventions for experiment

Convention of state meaning:

1) Initial, no connection yet.
2) Connection is requested, still no connection yet.
3) Connection is established, no packets waiting for acknowledgement.
4) Data packet sent, and waiting for acknowledgement.

Naming of signals:

R: Connection Request
C: Connection Accepted
L: Connection Cleared (Terminated)
D: Data packet
A: Acknowledgement
F: Fault symbol
X: unacceptable symbol ($X \notin I \wedge X \neq F$)

Fault lists:

| Fault ID | Original faulty machine | Fault Type |
|---|---|---|
| 0 | B | $C_a$ |
| 1 | B | $C_a$ |
| 2 | C | $C_a$ |
| 3 | C | $C_a$ |
| 4 | $CH_{BA}$ | $C_a$ |
| 5 | $CH_{CA}$ | $C_a$ |
| 10 | B | $C_b$ |
| 11 | A → B | $C_b$ |
| 12 | C | $C_b$ |
| 13 | A → C | $C_b$ |
| 20 | A | A |
| 21 | A | A |
| 30 | A | $B_a$ |
| 31 | A | $B_a$ |
| 40 | A | $B_b$ |
| 41 | B → A | $B_b$ |
| 42 | C → A | $B_b$ |

**Table 4 Test cases: ID, original fault location and detected location**

General assumptions:

- Time is measured in terms of atomic transition units at any machine.
- Injection of faults: we choose first the type of fault {$C_a$, $C_b$, A, $B_a$, or $B_b$} randomly. Then, we select one fault from the group of representing faults to inject. Finally we choose the time randomly {0 to ½ maximum simulation loop duration} when the fault should take place.

For machine B:

- Inputs "R" (Call request) is always available from end user X whenever B is at state 1.
- Clearing "L" at state 2 means non-acceptance of connection request, as a response from peer machine C.

- Clearing “L” at state 3 is sent from end user X, and triggered by last data packet acknowledged.
- Clearing “L” at state 4 is sent from peer machine C.
- Only one data packet is waiting for acknowledgement, i.e. stop and wait style.

For machine C:

- The clearing “L” (refusal of connection request) that is sent from state 2, will be chosen against the acceptance “C”, by random arbitration.
- The clearing “L” that is sent from state 4, will be chosen against the acknowledgement “A”, by random arbitration.